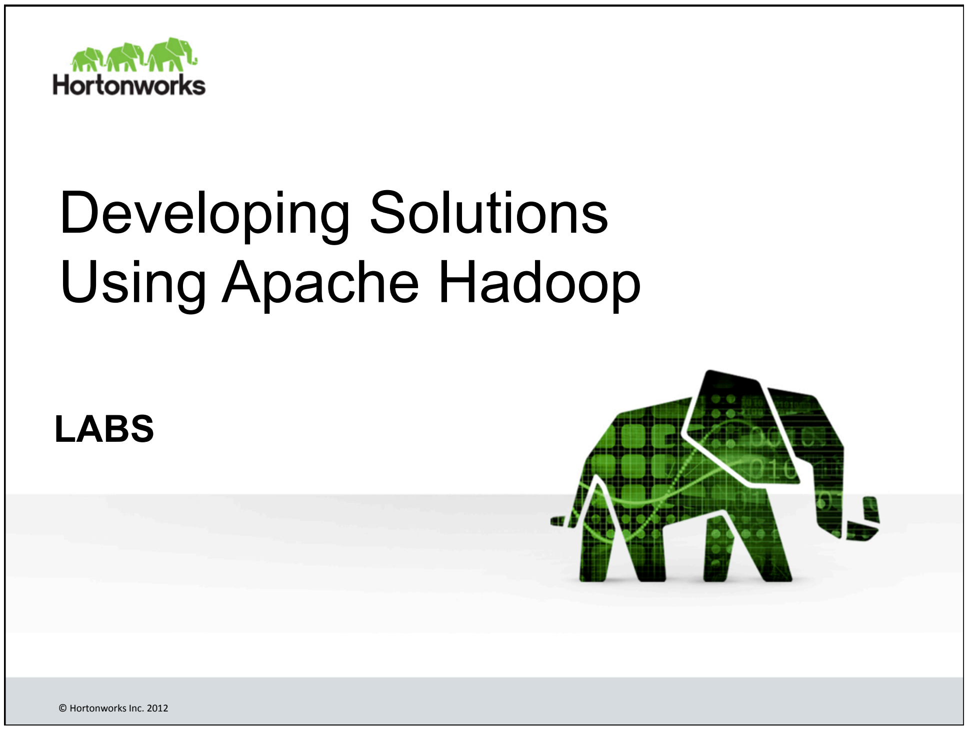


Developing Solutions Using Apache Hadoop

LABS

Lab 1.1: Login to Amazon EC2 Cluster

Description:

- Getting a terminal window
- Logging into lab
- Going to the right directory

Task 1: Opening a Terminal Window

Activity Procedure

For a MAC:

Step 1. Go to far upper right hand corner of your screen and click on the magnifying glass icon.

Step 2. Type in terminal and select terminal

For a PC:

Step 1. You will need a ssh client, your instructor will provide you a copy of putty to install.

Step 2. Invoke Putty and specify an ssh connection

Activity Verification

When the job completes, you will see similar to the following on your screen:

```
You have a terminal window.
```

Task 2: Logging into the lab

Activity Procedure

For a MAC

Step 1. In your terminal window type in:

```
$ ssh train@[EC2 Instance Name provided by instructor]
```

Step 2. To “Are you sure you want to continue connecting” type in:

```
yes
```

Step 3. For the password enter:

```
$ [to be provided by the instructor]
```

For the PC

Step 1. Bring up a terminal window in putty (ssh port 22 and the EC2 Instance provided by the instructor)

You should save the configuration so that you won't have to retype the host name every time that you use putty.

Step 2. Enter the username `train`

Step 3. For the password enter:

```
$ [to be provided by the instructor]
```

Activity Verification

When the job completes, you will a screen similar to the following:

```
When you are logged in and see the prompt change to:
   [train@/home/train]$
```

NOTE: Throughout this course, the lab instructions will instruct you to use vi to edit text files. If you are not familiar with the vi text editor, then you may want to use emacs instead. Ask your instructor if you have any questions or concerns about editing the lab files.

Task 3: Change to Your Student Directory

Activity Procedure

Step 1. Check that you are in the right directory. Type in:

```
$ pwd
Your directory should be /home/train
```

Step 2. Check that you have the correct directories. Type in:

```
$ ls
```

Activity Verification

When the job completes, you will see similar to the following on your screen:

```
The directory you are in is:
   /home/train
The files/directories in your directory are:
  Backup      labs
```

Lab 2.1: Use HDFS File System Commands

Task 1: File listing related commands

Activity Procedure and Verification

Step 1. Let's browse the hdfs directory structure. Note that by default it shows the listing of your home directory, which is /user/train

```
$ hadoop fs -ls
```

Step 2. To check what other directories are available in HDFS:

```
$ hadoop fs -ls /
```

Step 3. To check directory structure reclusively:

```
$ hadoop fs -lsr /user/train
```

Task 2: File/Directory Creation/deletion commands

Step 1. Make a directory in HDFS (We will delete it later.)

```
$ hadoop fs -mkdir test
```

Step 2. Make a sub-directory in HDFS (We will delete it later.)

```
$ hadoop fs -mkdir test/test1
$ hadoop fs -mkdir test/test2/test3
```

Step 3. Make sure the directory has been created. Let's do a recursive directory listing:

```
$ hadoop fs -lsr /user/train/
```

You should see:

```
drwx------   - train hdfs          0 2012-04-12 14:35 /user/train/exampledir
drwx------   - train hdfs          0 2012-06-03 16:14 /user/train/test
drwx------   - train hdfs          0 2012-06-03 16:13 /user/train/test/test1
drwx------   - train hdfs          0 2012-06-03 16:14 /user/train/test/test2
drwx------   - train hdfs          0 2012-06-03 16:14 /user/train/test/test2/test3
```

Step 4. Copy input data file to this directory

```
$ cd ~/labs/hdfsbasic
$ hadoop fs -put 10000.txt test
```

Step 5. Check that the file was copied to HFDS:

```
$ hadoop fs -ls test
```

```
Found 3 items
-rw-------   1 train hdfs     509785 2012-06-03 16:15 /user/train/test/10000.txt
drwx------   - train hdfs          0 2012-06-03 16:13 /user/train/test/test1
drwx------   - train hdfs          0 2012-06-03 16:14 /user/train/test/test2
```

Step 6. Make a copy of the file in HDFS to another file within HDFS (we will delete it later):

```
$ hadoop fs -cp test/10000.txt test/10000_1.txt
```

Step 7. Check that both files exist:

```
$ hadoop fs -ls test
```

You should see:

```
Found 4 items
-rw-------   1 train hdfs     509785 2012-06-03 16:15 /user/train/test/10000.txt
-rw-------   1 train hdfs     509785 2012-06-03 16:16 /user/train/test/10000_1.txt
drwx------   - train hdfs          0 2012-06-03 16:13 /user/train/test/test1
drwx------   - train hdfs          0 2012-06-03 16:14 /user/train/test/test2
```

Step 8. Now we will remove the copied file we just created:

```
$ hadoop fs -rm test/10000_1.txt
```

Step 9. Check that the file was removed:

```
$ hadoop fs -ls test
```

Step 10. Remove the entire directory:

```
$ hadoop fs -rmr test
```

Step 11. Check that the directory was removed and a new directory .Trash is created. You can find all deleted folders and files here.

```
$ hadoop fs -lsr /user/train/
```

```
drwxrwxrwx   - train hdfs          0 2012-06-03 16:18 /user/train/.Trash
drwxrwxrwx   - train hdfs          0 2012-06-03 16:18 /user/train/.Trash/Current
drwxrwxrwx   - train hdfs          0 2012-06-03 16:18 /user/train/.Trash/Current/user
drwxrwxrwx   - train hdfs          0 2012-06-03 16:19
    /user/train/.Trash/Current/user/train
```

```
drwx------   - train hdfs          0 2012-06-03 16:18
    /user/train/.Trash/Current/user/train/test
-rw-------   1 train hdfs     509785 2012-06-03 16:16
    /user/train/.Trash/Current/user/train/test/10000_1.txt
drwx------   - train hdfs          0 2012-06-03 16:18
    /user/train/.Trash/Current/user/train/test.1
-rw-------   1 train hdfs     509785 2012-06-03 16:15
    /user/train/.Trash/Current/user/train/test.1/10000.txt
drwx------   - train hdfs          0 2012-06-03 16:13
    /user/train/.Trash/Current/user/train/test.1/test1
drwx------   - train hdfs          0 2012-06-03 16:14
    /user/train/.Trash/Current/user/train/test.1/test2
drwx------   - train hdfs          0 2012-06-03 16:14
    /user/train/.Trash/Current/user/train/test.1/test2/test3
drwx------   - train hdfs          0 2012-04-12 14:35 /user/train/exampledir
```

Task 3: File/Directory permissions related commands

Step 1. Copy the test file again from local file system to HDFS:

```
$ hadoop fs -copyFromLocal 10000.txt test/10000.txt
$ hadoop fs -ls test
```

Step 2. Change the file permission on the HDFS file:

```
$ hadoop fs -chmod 777 test/10000.txt
```

Step 3. Look at the permissions on the file that you just changed the permissions for:

```
$ hadoop fs -ls test/10000.txt
```

Task 4: Commands to read a file

Step 1. Check the content of the HDFS file:

```
$ hadoop fs -cat test/10000.txt
```

Step 2. To check the content of the HDFS file, but page wise:

```
$ hadoop fs -cat test/10000.txt | more
NOTE: CTRL + C will bring you back to command prompt
```

Step 3. To check the content of the HDFS file, but last 20 lines:

```
$ hadoop fs -tail test/10000.txt
```

Task 5: Other useful commands

Step 1. To find out the disk usage in your hdfs directory, the following statement is very useful. You should have access permission to be able to view this.

```
$ hadoop fs -du
```

Step 2. To count the number of directories, files and bytes under the paths that match the specified file pattern. The output columns are DIR_COUNT, FILE_COUNT, CONTENT_SIZE, FILE_NAME

```
$ hadoop fs -count /user/train
```

Step 3. To Display help for a given command or all commands if none is specified

```
$ hadoop fs -help [cmd]
```

Task 6: Exercise – Learning 'getmerge' command

Step 1. Create 2 different data files (5 records each) using following fields:

Name, Age, Zip, Salary

e.g.: File1.txt

```
Tom,21,94085,5000

John,45,95014,25000

Joe,21,94085,5000

Larry,45,95014,25000

Hans,21,94085,5000
```

e.g.: File2.txt

```
T1,21,94085,5000

T2,45,95014,25000

T3,21,94085,5000

T4,45,95014,25000

T5,21,94085,5000
```

Step 2. Copy these files to HDFS in a new directory **/user/train/merge**

Step 3. Get help on **getmerge** command and use it for these 2 data files.

Step 4. Review the output file.

(NOTE: The documentation states that getmerge sorts the data, but in actuality the data does not get sorted.)

Task 7: Optional: Challenge Questions?

Find out the difference between the following commands:

```
i> $ hadoop fs –ls *
   $ hadoop fs -ls "*"

ii> $ hadoop fs -dus
    $ hadoop fs -du

iii> $ hadoop fs -get
     $ Hadoop fs -copyToLocal

iv> $ hadoop fs -put
    $ hadoop fs -copyFromLocal
    $ hadoop fs -moveFromLocal
```

Lab 2.2: Program HDFS

Task 1: Check Java CLASSPATH

Activity Procedure and Verification

Step 1. Make sure that $CLASSPATH has needed JARs.

```
$ echo $CLASSPATH
```

You should see the following jar files, which are needed for this lab:

```
commons-configuration-1.6.jar
commons-logging-1.1.1.jar
log4j-1.2.15.jar
```

Task 2: Create (or modify) input file

Step 1. Change your working directory:

```
$ cd /home/train/labs/hdfs-p
```

Step 2. Locate the **`input-file.txt`** that we have provided for your work:

```
$ ls -l

-rw-rw-r-- 1 train train 141 Jun  1 14:07 input-file.txt
```

You should see the file as shown above. Feel free to edit it's content.

Task 3: Write a Java class to explore programming HDFS

For those who are Java developers, we provide a detailed description of what the class must accomplish. For those who are less comfortable with Java, we provide a step-by-step list of instructions below the description.

Class	ShowAPI
Responsibilities	The purpose of this application is to encapsulate read and write functionality with HDFS. Take as input two parameters, an inputPath and and outputPath. The inputPath will be used to read data from the Local Filesystem. The outputPath will be used to write to HDFS. The primary behavior is to take these two parameters and wrap them with appropriate input and output streams.

	Use the public static factory methods to get an instance of the LocalFileSystem (Linux) and the FileSystem (HDFS). Both require an instance of the Configuration object. Make a directory on HDFS to which you will write your file. Work with public factory methods on the provided FileSystem and LocalFileSystem types to obtain the input and output streams (see: open() and create() methods which require the inputPath and outputPath respectively). Use a byte array as a buffer to hold the file input data and write it to the HDFS output file. The input stream has a read method (inherited from DataInputStream, which simplifies read, and returns -1 when EOF is reached). The output stream has a simple write method which requires three parameters to write the output data: the byte array (buffer), an offset and the length of the buffer to write. Don't forget to close your streams ☺ Exception handling code is required.
Collaborators	An input file: input-file.txt (or another source) on the local Linux filesystem. Hadoop API types: org.apache.hadoop.fs.Path org.apache.hadoop.conf.Configuration org.apache.hadoop.fs.FileSystem & org.apache.hadoop.fs.LocalFileSystem org.apache.hadoop.fs.FSDataInputStream & org.apache.hadoop.fs.FSDataOutputStream An HDFS output directory "/user/train/hdfs-p" An HDFS output file
Other	byte[] to use as a buffer. java.io.IOException

Step 1. Write a class named **`ShowAPI`** with two methods. A skeleton version of this class has been provided for you (see the directory **`/home/train/labs/hdfs-p/ShowAPI.java`**).

Step 2. Double check that required imports are present to reach the HDFS package:

```
import org.apache.hadoop.fs.*;

import org.apache.hadoop.conf.*;

import java.io.*;
```

Step 3. FIRST METHOD. Write a public method to encapsulate working with local and HDFS filesystems, and pass in two arguments of type **`Path`**, for example:

```
public void showHDFS(Path inPath, Path outPath) throws IOException
```

1. If you don't add the **`throws IOException`** clause you will have to deal with exceptions within the method itself, your decision.
2. One **`Path`** is for the input path, which is the file we want to read.
3. The other **`Path`** is for the output path, which is the file we want to write.

Step 4. Within this method, instantiate a default **`Configuration`** object, this is needed for our work with the **`FileSystem`** factory methods later. For example:

```
Configuration config = new Configuration();
```

Step 5. We want to write a file to HDFS. Acquire a **`FileSystem`** object which represents HDFS, by invoking the public static **`FileSystem`** factory method **`get(Configuration conf)`** such as:

```
FileSystem hdfs = FileSystem.get(config);
```

Step 6. We want to read a file from the Linux filesystem. Acquire a **`LocalFileSystem`** object which represents the local Linux filesystem, by invoking the public static **`FileSystem`** factory method **`getLocal(Configuration conf)`** such as:

```
LocalFileSystem local = FileSystem.getLocal(config);
```

Step 7. We want to read the data from the local file system using a filesystem data input stream. The input stream we want is one wrapped around our input **`Path`** argument to this method, we can just pass that directly to the **`FileSystem open()`** factory method, such as:

```
FSDataInputStream inStream = local.open(inPath);
```

Step 8. We want to write the data to the HDFS using a filesystem data output stream. The output stream we want is one wrapped around our output **`Path`** argument to this method, we can pass that directly to the **`FileSystem create()`** factory method, such as:

```
FSDataOutputStream outStream = hdfs.create(outPath);
```

Step 9. For this to work, when we read from the file we need to hold the file's data in memory in a **`byte`** array buffer. We just picked 1000 bytes as a general number:

```
byte[] fromFile = new byte[1000];
```

Step 10. Read the input file. This can be achieved by a simple **`read()`** method inherited from the **`java.io.DataInputSteam.`** This **`read()`** method requires that **`byte`** buffer to be passed as an argument, to get it's work done. Also, **`read()`** stores the bytes into the array, and returns each byte read (as an **`int`**) from the file. When it reaches the end of file (EOF), a value of **`-1`** is returned. Loop over the input steam's **`read()`** method until you reach the end of the file, for example:

```
int data = 0;

while((data = inStream.read(fromFile)) > 0)

{ /* perhaps report using System.out that you are reading… */ }
```

Step 11. Write the buffer data out to HDFS. The output steam has a **`write()`** method which requires three arguments respectively, a **`byte`** array (our buffer "fromFile"), an offset into that array, and lastly the length (number of bytes to write):

```
outStream.write(fromFile, 0, 1000);
```

Step 12. Close the streams:

```
inStream.close();

outStream.close();
```

Step 13. SECOND METHOD. Write a main() method which invokes the method you just completed. Main always has the same signature:

```
public static void main(String[] args)
```

Step 14. We need an inputPath and an outputPath. If the length of the **`args`** array is < 2, send a usage message to **`System.out`** letting the user know what you are expecting. Take the first argument, knowns as **`args[0]`** and use it for the inputPath value. Take the second argument, **`args[1]`** and use it for the outputPath value. The **`args`** will be of type **`String`**. Consider the following code, where we check for the length of the incoming arguments to our Java application, and finding two we commence to work with the data provided (from the command line):

```
if(args.length == 2) {

     String inputPath = args[0];

     String outputPath = args[1];

     try { //because our method might throw an exception

             ShowAPI showMe = new ShowAPI(); //instantiate our object

             //invoke our method, wrapping Path around the input/output
```

```
            showMe.showHDFS(new Path(inputPath), new Path(outputPath));

        }catch (Exception e){ /* … */ }

    }
```

Step 15. Compile your Java class from the command line.

```
$ javac ShowAPI.java
```

Step 16. We must send this JAR to Hadoop to run it on HDFS, and provide the arguments for the inputFile and outputFile, try this:

```
$ java ShowAPI input-file.txt /user/train/hdfs-p/output.txt
```

Step 17. Ask Hadoop to show you what has been created in the HDFS by invoking the **–lsr** command (list recursively) this way:

```
$ hadoop fs –ls hdfs-p

-rw-------   1 train hdfs       1000 2012-06-03 17:18
/user/train/hdfs-p/output.txt
```

You will see that first the **hdfs-p** directory was created under **/user/train/**, and then the file was placed in that directory. Success!

Step 18. LAB RESULTS: Read the content of the file you just programmatically copied from the Linux filesytem to the HDFS with the **–cat** command:

```
[train@/home/train]$ hadoop fs -cat /user/train/hdfs-p/output.txt

www.yahoo.com news sports finance email celebrity
www.amazon.com shoes books jeans
www.google.com news finance email search
www.microsoft.com operating-system productivity search
www.target.com shoes books jeans groceries
```

Lab 2.3: Monitor HDFS

We will cover following in this lab:

1. HDFS Configuration Files
2. HDFS Web GUIs
3. Starting/Stopping services

Task 1: Check Hadoop configuration files

There are two kinds of configuration files. Default configuration files, which reside in the hadoop-core jar file. The other kind is user editable configuration files, which can be found (usually) in the **`/etc/hadoop`** directory.

Activity Procedure and Verification

Step 1. First we will look into editable configuration files. Check for the XML files in following locations

```
$ ls -l /etc/hadoop/*.xml
```

You should see a list of xml files, which Hadoop uses for the configuration:

```
-rw-r--r-- 1 root root  2082 Apr  3 13:02 /etc/hadoop/capacity-
scheduler.xml

-rw-r--r-- 1 root root  7372 Apr  3 13:02 /etc/hadoop/core-site.xml

-rw-r--r-- 1 root root  5001 Apr  3 13:02 /etc/hadoop/hadoop-policy.xml

-rw-r--r-- 1 root root 11137 Apr  3 13:02 /etc/hadoop/hdfs-site.xml

-rw-r--r-- 1 root root   402 Apr  3 13:02 /etc/hadoop/mapred-queue-
acls.xml

-rw-r--r-- 1 root root 14315 Apr  3 13:02 /etc/hadoop/mapred-site.xml
```

Step 2. Let's check Namenode and Datanode configuration files first. Change directory to where these configuration files are located:

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

Step 3. Let's view the configuration file and review various parameters.

```
$ less hdfs-site.xml
```

Step 4. Notice that for the dfs.http.address property the default port is 50070

```
 <property>
    <name>dfs.http.address</name>
    <value>hortonworks-sandbox.localdomain:50070</value>
<description>The name of the default file system.  Either the
literal string "local" or a host:port for NDFS.</description>
<final>true</final>
</property>
```

Step 5. Check for what is the HDFS location for data as defined in the configuration file.

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

<value>/hdp/disk0/data/HDP/hadoop/datanode_dir,/hdp/disk1/data/HDP/hadoo
p/datanode_dir</value>
    <description>Determines where on the local filesystem an DFS data
node
  should store its blocks.  If this is a comma-delimited
  list of directories, then data will be stored in all named
  directories, typically on different devices.
  Directories that do not exist are ignored.
  </description>
    <final>true</final>
  </property>
```

Note: The final property set top true makes this property unchangeable from other sources.

Step 6. Check who is the administrator for the cluster?

```
<property>
 <name>dfs.cluster.administrators</name>
 <value> hdfs</value>
 <description>ACL for who all can view the default servlets in the
HDFS</description>
 </property>
```

Step 7. Check that value for super-group of the administrator for the cluster.

```
<property>

<name>dfs.permissions.supergroup</name>

<value>hdfs</value>

<description>The name of the group of super-users.</description>

</property>
```

Step 8. Let's view configuration file for the JobTracker and TaskTracker and review various parameters.

```
 $ less mapred-site.xml
```

Step 9. Notice that for the `mapred.job.tracker.http.address` property the default port is `50030`. Which is the setting of the port address for the jobtacker.

```
<property>
    <name>mapred.job.tracker.http.address</name>
    <value>hortonworks-sandbox.localdomain:50030</value>
    <description>No description</description>
    <final>true</final>
  </property>
```

Step 10. Check for the location where job related data is being written:

```
<property>
    <!-- cluster specific -->
    <name>mapred.local.dir</name>

<value>/hdp/disk0/data/HDP/hadoop/mapred_dir,/hdp/disk1/data/HDP/hadoop/m
apred_dir</value>
    <description>No description</description>
    <final>true</final>
</property>
```

Step 11. Check for following properties for maximum number of Mappers and Reducers:

```
<property>
    <name>mapred.tasktracker.map.tasks.maximum</name>
    <value>4</value>
    <description>No description</description>
  </property>

  <property>
    <name>mapred.tasktracker.reduce.tasks.maximum</name>
    <value>2</value>
    <description>No description</description>
  </property>
```

Step 12. To view the hadoop-env file bring it up in an editor:

```
$ less /etc/hadoop/hadoop-env.sh
```

This file is the place where you can set all your variables centrally.

```
  # The java implementation to use.
  export JAVA_HOME=/usr/jdk1.6.0_26
  export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-"/etc/hadoop"}


# Where log files are stored.  $HADOOP_HOME/logs by default.

export HADOOP_LOG_DIR=/hdp/disk0/data/HDP/hadoop/log_dir/$USER
```

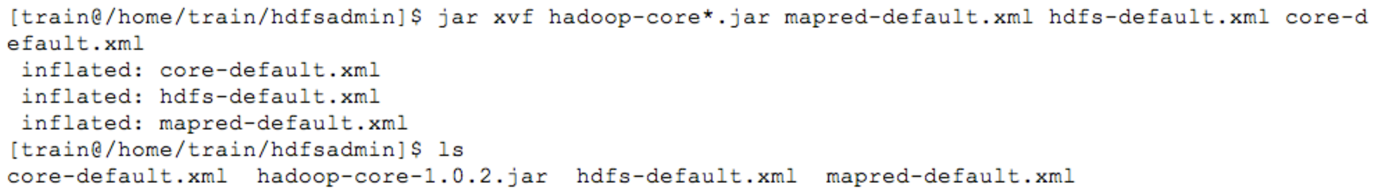
Step 13. Accessing read-only configuration files:

```
$ mkdir ~/labs/hdfsadmin
$ cd ~/labs/hdfsadmin

$ cp /usr/share/hadoop/hadoop-core-*.jar .
$ jar xvf hadoop-core-*.jar mapred-default.xml hdfs-default.xml core-
   default.xml
```

You will see that the following three files have been extracted (or inflated) from the Hadoop jar files:

`mapred-default.xml, hdfs-default.xml, core-default.xml`



```
[train@/home/train/hdfsadmin]$ jar xvf hadoop-core*.jar mapred-default.xml hdfs-default.xml core-d
efault.xml
 inflated: core-default.xml
 inflated: hdfs-default.xml
 inflated: mapred-default.xml
[train@/home/train/hdfsadmin]$ ls
core-default.xml  hadoop-core-1.0.2.jar  hdfs-default.xml  mapred-default.xml
```

Review these files and you will see almost the same settings as what you have seen in earlier steps for hdfs-site.xml, mapred-site.xml and core-site.xml. The value of these variables can be changed in the corresponding *–site.xml file.

Task 2: Hadoop Primary daemons and Environment Variables

Activity Procedure and Verification

Step 1. To view a list of primary daemons execute following

```
$ ps -ef | grep "Dproc" | awk '{print $1  "\t"  $2  "\t" $9}'
```

You should see a list of daemon process:

```
hdfs    6333    -Dproc_namenode
hdfs    6659    -Dproc_secondarynamenode
hdfs    6871    -Dproc_datanode
mapred  11252   -Dproc_jobtracker
mapred  11542   -Dproc_historyserver
mapred  11743   -Dproc_tasktracker
hcat    12038   -Dproc_jar
2001    12950   -Dproc_jar
root    14762   Dproc
```

Step 2. Let's find out where the hadoop binaries are located.

```
$ which hadoop
```

You should see:

```
/usr/bin/hadoop
```

Task 3: Checking HDP web based GUI:

You can browse the following web based UIs in HDP1.0.x versions:

1. For the Namenode GUI:

   URL: **http://<EC2 Instance Name>:50070**

2. For the JobTracker GUI:

   URL: **http://<EC2 Instance Name>:50030**

3. For the Hortonworks Data Platform:

   URL: **http://<EC2 Instance Name>/hdp/dashboard/ui/home.html**

4. For Nagios:

   URL: **http://<EC2 Instance Name>/nagios**

   - User name: `nagiosadmin`
   - Password: `admin`

   If that fails, ask your instructor for these values.

5. For Ganglia:

   URL: **http://<EC2 Instance Name>/ganglia**

Please note: If any Link does not work on these pages, fix the URL by replacing hostname with your EC2 Instance external hostname.

For example, if the URL is: **`http://hortonworks-sandbox.localdomain:50030/logs`**
Then replace it with:

```
http:// ec2-23-22-167-119.compute-1.amazonaws.com:50030/logs
```

Task 4: Stopping and Starting HDP:

Activity Procedure and Verification

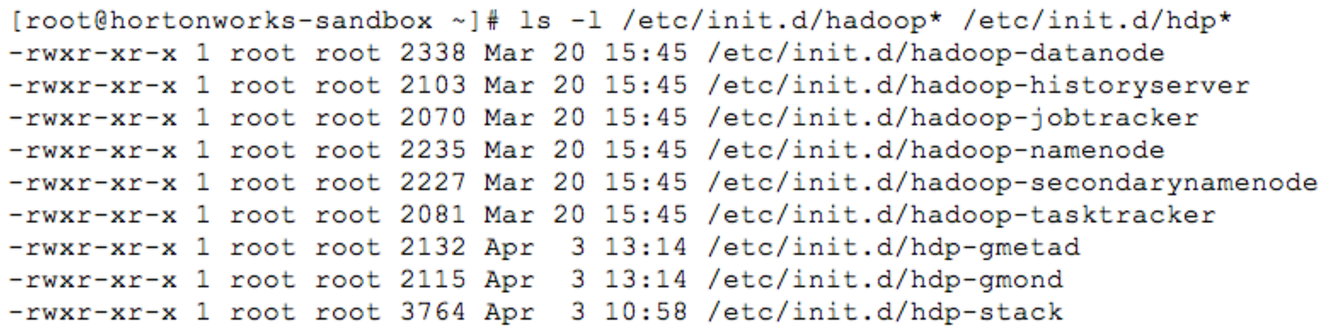
Step 1. To stop and Start services you need to login as ‘root’ user. Only ‘root’ user can do most of the administration work. Let’s login as root by executing following command. Please note the password for root user is same as ‘train’ user.

```
$ su – root
```

Password: **`train`**

```
$ ls -l /etc/init.d/hadoop* /etc/init.d/hdp*
```

You should see a list of scripts to stop/start different hadoop daemons:



```
[root@hortonworks-sandbox ~]# ls -l /etc/init.d/hadoop* /etc/init.d/hdp*
-rwxr-xr-x 1 root root 2338 Mar 20 15:45 /etc/init.d/hadoop-datanode
-rwxr-xr-x 1 root root 2103 Mar 20 15:45 /etc/init.d/hadoop-historyserver
-rwxr-xr-x 1 root root 2070 Mar 20 15:45 /etc/init.d/hadoop-jobtracker
-rwxr-xr-x 1 root root 2235 Mar 20 15:45 /etc/init.d/hadoop-namenode
-rwxr-xr-x 1 root root 2227 Mar 20 15:45 /etc/init.d/hadoop-secondarynamenode
-rwxr-xr-x 1 root root 2081 Mar 20 15:45 /etc/init.d/hadoop-tasktracker
-rwxr-xr-x 1 root root 2132 Apr  3 13:14 /etc/init.d/hdp-gmetad
-rwxr-xr-x 1 root root 2115 Apr  3 13:14 /etc/init.d/hdp-gmond
-rwxr-xr-x 1 root root 3764 Apr  3 10:58 /etc/init.d/hdp-stack
```

Step 2. Now let’s stop all hadoop services:

```
$ /etc/init.d/hdp-stack stop
```

You should see all the HDP services are being stop one by one. This will take some time. Once it is done check for all hadoop processes:

```
$ ps -ef | grep "Dproc" | awk '{print $1  "\t"  $2  "\t" $9}'
```

You should see there are no service daemons running:

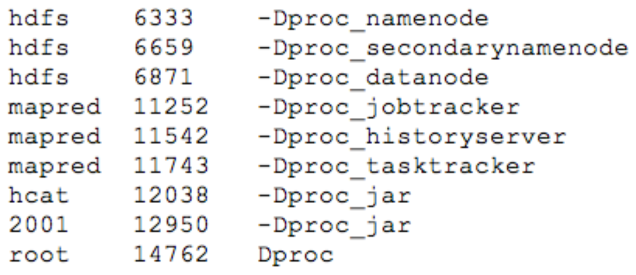
Step 3. Restart HDP and check again for the running processes and services. This will take some time, please be patient:

```
$ /etc/init.d/hdp-stack start
```

It takes about 5-10 minutes to restart all the services. Once it is done check for all hadoop process:

```
$ ps -ef | grep "Dproc" | awk '{print $1  "\t"  $2  "\t" $9}'
```

You should see a list of daemon process:



```
hdfs    6333    -Dproc_namenode
hdfs    6659    -Dproc_secondarynamenode
hdfs    6871    -Dproc_datanode
mapred  11252   -Dproc_jobtracker
mapred  11542   -Dproc_historyserver
mapred  11743   -Dproc_tasktracker
hcat    12038   -Dproc_jar
2001    12950   -Dproc_jar
root    14762   Dproc
```

Step 4. Logout from ‘root’ user now for further labs.

```
$ exit
```

Lab 3.1: Run An Existing MapReduce Job

Description:

- Load input data into HDFS
- Run a Map Reduce job to create an inverted index of that data.

Task 1: Loading data into HDFS

Activity Procedure

Step 1. Change to the directory where the data is located:

```
$ cd ~/labs/mrintro
```

Step 2. In the Hadoop file system, create a directory as follows:

```
$ hadoop fs -mkdir mrintro/input/
```

Step 3. Into the Hadoop file system, put a file as follows:

```
$ hadoop fs -put in-0000* mrintro/input/
```

Step 4. List the files in the hdfs directory

```
$ hadoop fs -ls mrintro/input/
```

You should see the in-00000 and the in-00001 files listed.

Step 5. To see the content in this file type in:

```
$ hadoop fs -cat mrintro/input/in-00000
```

Activity Verification

When the job completes, you should see the **in-00000** file listed and the content of the data file.

```
www.yahoo.com news sports finance email celebrity

www.amazon.com shoes books jeans

www.google.com news finance email search

www.microsoft.com operating-system productivity search

www.target.com shoes books jeans groceries
```

Task 2: Building and Running the Map Reduce Job

Activity Procedure

Step 1. Compile the java file

```
$ javac ProductSearchIndexer.java
```

Step 2. Jar all the class files

```
$ jar cvf mrintro.jar *.class
```

Step 3. Next, let's run the above Map Reduce Job as follows:

```
$ hadoop jar mrintro.jar ProductSearchIndexer mrintro/input
   mrintro/output
```

Activity Verification

When the job completes, you will see something similar to the following on your screen:

```
Job started: Wed Jan 11 22:18:31 EST 2012
12/01/11 22:18:32 INFO mapred.FileInputFormat: Total input paths to
process : 2
12/01/11 22:18:32 INFO mapred.JobClient: Running job:
……………………..
……………………………

12/01/11 22:19:02 INFO mapred.JobClient:     Map input records=8
12/01/11 22:19:02 INFO mapred.JobClient:     Reduce shuffle bytes=639
12/01/11 22:19:02 INFO mapred.JobClient:     Spilled Records=48
12/01/11 22:19:02 INFO mapred.JobClient:     Map output bytes=693
12/01/11 22:19:02 INFO mapred.JobClient:     Total committed heap usage
(bytes)=809369600
12/01/11 22:19:02 INFO mapred.JobClient:     CPU time spent (ms)=3890
12/01/11 22:19:02 INFO mapred.JobClient:     Map input bytes=350
12/01/11 22:19:02 INFO mapred.JobClient:     SPLIT_RAW_BYTES=408
12/01/11 22:19:02 INFO mapred.JobClient:     Combine input records=30
12/01/11 22:19:02 INFO mapred.JobClient:     Reduce input records=24
12/01/11 22:19:02 INFO mapred.JobClient:     Reduce input groups=14
12/01/11 22:19:02 INFO mapred.JobClient:     Combine output records=24
12/01/11 22:19:02 INFO mapred.JobClient:     Physical memory (bytes)
snapshot=657444864
12/01/11 22:19:02 INFO mapred.JobClient:     Reduce output records=14
12/01/11 22:19:02 INFO mapred.JobClient:     Virtual memory (bytes)
snapshot=4096884736
12/01/11 22:19:02 INFO mapred.JobClient:     Map output records=30
Job ended: Wed Jan 11 22:19:02 EST 2012
The job took 30 seconds.
```

Task 3: Viewing the Output

Activity Procedure

Step 1. List the files in the HDFS directory

```
$ hadoop fs -ls mrintro/output/
```

Step 2. Save a copy of the output hdfs file in local file system as follows:

```
$ hadoop fs -get mrintro/output/part-00000 ~/labs/mrintro/inverted.txt
```

Step 3. View the output of your MapReduce job

```
$ cat inverted.txt
```

Activity Verification

When the job completes, you will see something similar to the following on your screen:

```
books     www.walmart.com  www.amazon.com  www.target.com
celebrity      www.yahoo.com
email     www.google.com www.yahoo.com  www.facebook.com
finance   www.yahoo.com www.google.com
groceries      www.walmart.com  www.target.com
jeans     www.target.com  www.amazon.com  www.walmart.com
news      www.facebook.com www.linkedin.com  www.yahoo.com www.google.com
operating-system    www.microsoft.com
productivity   www.microsoft.com
recruitment    www.linkedin.com
search    www.google.com www.microsoft.com
shoes     www.amazon.com  www.target.com  www.walmart.com
social    www.facebook.com
sports    www.facebook.com www.walmart.com  www.yahoo.com
```

Lab 3.2: Write a Simple MapReduce Program

Description:

- First Create a test input file
- Develop code for the word-count MapReduce program
- Enhance code by using existing methods in Hadoop API.

Task 1: Create an input file and copy it to HDFS

Activity Procedure

Step 1. Change to the lab directory where we will create the data:

```
$ cd ~/labs/mrsimple
```

Step 2. Create your own input file (input.txt) having some random text in written in multiple lines. Make sure some words are repeating. The MapReduce program will count words in this input file and will provide you an output having count for each word.

```
$ vi input.txt
```

Step 3. Copy the input file into HDFS

```
$ hadoop fs -put input.txt mrsimple/input.txt
```

Task 2: Step-by-step development of your first MapReduce program

Activity Procedure

Step 1. Write a class named **`MyFirstMR`**. A skeleton version of this class has been provided for you in the current working directory(see the directory **`/home/train/labs/mrsimple`**).

```
$ vi MyFirstMR.java
```

Please note it has JAVA mandatory standard imports and also Hadoop imports required to execute the code. It also has class declaration with no code inside. It has one mandatory **`main()`** method which invokes your public class **`MyFirstMR`** at runtime.

Step 2. To create a new job, first we need to instantiate an object of Hadoop JobConf class in the main method by passing it your class name. Add below line in the **`main()`** method.

```
final JobConf conf = new JobConf(MyFirstMR.class);
```

Step 3. Set the data type for the output Key and Value in the main method. In this exercise, the output is a Key-Value pair having Text & LongWritable data types respectively. Add below lines in the main method:

```
conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(LongWritable.class);
```

Step 4. We are using in-built classes to count words from the input file. The TokenCountMapper.class is being treated as mapper class. This class breaks all the sentences into words and LongSumReducer.class count the occurrences of each word. Add below lines in the main method.

```
conf.setMapperClass(TokenCountMapper.class);

conf.setReducerClass(LongSumReducer.class);
```

Step 5. Read command line arguments to identify input file and the output directory. The first argument is input file and the second one is output directory. Add below lines in the main method.

```
FileInputFormat.setInputPaths(conf, new Path(args[0]));

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

Step 6. The final step to add the statement, which executes the job. Add below line in the main method.

```
JobClient.runJob(conf);
```

Save your work.

Task 3: Compile and run the MapReduce program

Activity Procedure

Step 1. Let's first review what JAR files are included in the CLASSPATH. Your Java program uses **`hadoop-core-1.0.2.jar`** file for the compilation of the code.

```
$ echo $CLASSPATH
```

You will notice that **`hadoop-core-1.0.2.jar`** is already included in the Java CLASSPATH.

Step 2. Let's compile the code and make a jar file from the CLASS files.

```
$ javac MyFirstMR.java
$ jar cvf mr.jar *.class
```

Step 3. It is time to run your first program. Use the following command to see your program in action. It will take some time to complete running:

```
$ hadoop jar mr.jar MyFirstMR mrsimple/input.txt mrsimple/output
```

Step 4. Let's check the output of the program:

```
$ hadoop fs -cat mrsimple/output/part-00000
```

Task 4: Exercise – Write an Inverted Index MapReduce Program

Activity Procedure

Step 1. You can reuse the Task 3 code for this exercise. Try to do the following yourself:

1. Make a copy of the MyFirstMR.java by saving it as MyInvertedIndex.java
2. Open the file and change the class name as MyInvertedIndex (everywhere)
3. Change Mapper class to InverseMapper.class
4. Change Reducer class to IdentityReducer.class
5. Save the file
6. Compile the code and create a jar file

   ```
   $ javac MyInvertedIndex.java
   $ jar cvf mr.jar *.class
   ```

7. Run your program using same input file. But make sure your output directory must be different (e.g. output1):

   ```
   $ hadoop jar mr.jar MyInvertedIndex mrsimple/input.txt
   mrsimple/output1
   ```

8. Review the output:

   ```
   $ hadoop fs -cat mrsimple/output1/part-00000
   ```

   The output would be something like the following (a word or a sentence followed by a number):

   ```
   Testing       76
   Inverted      21
   Index output.      0
   ```

Note: The numbers are the keys created by the MapReduce program for each line. Your inverted index program inverts keys into data and data into keys.

Task 5: Optional – Write a word count MapReduce Program using your own Mapper Class and Reducer Class

Activity Procedure

Step 1. Make a copy of the `MyFirstMR.java` by saving it as `MyWordCount.java`

Step 2. Add an implementation of the Mapper interface as an inner-class in the MapReduce public class `MyWordCount`. This inner-class has only one method called `map(…)`.

A. This method expects four data types.

B. The first two data types are for input Key & Value.

C. The remaining two data types are for the output Key & Value.

D. In this exercise, the `map(…)` method expects input Key-Value as a LongWritable and Text respectively.

E. The output key-value must be Text & IntWritable respectively.

```
public static class MyMap extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable>{
      public void map(LongWritable key, Text value,
  OutputCollector<Text, IntWritable> collector, Reporter reporter)
  throws IOException {
        /* code here */
  }
}
```

Step 3. Add a Reducer inner-class in the MapReduce public class. This class has only one method called `reduce()`.

This method expects four data types. The first two data types are for input Key & Value and the remaining two data types are for the output Key & Value. In this exercise, the reduce method expects input Key-Value as Text and IntWritable respectively. The output Key-Value must be Text & IntWritable respectively.

```
public static class MyReduce extends MapReduceBase implements
```

```
Reducer<Text, IntWritable, Text, IntWritable> {
      public void reduce(Text key, Iterator<IntWritable> values,
  OutputCollector<Text, IntWritable> output, Reporter reporter)
  throws IOException { /* code here */ } }
```

Step 4. Write your own logic in **`map()`** and **`reduce()`** methods to achieve the word count functionality.

Step 5. If you need help with the logic, you can refer to the working solution in the same directory. The file name is **`WordCount.java`**.

Lab 4.1: Create a Custom Writable Type

Description:

- Create a custom data type that can store a pair of values. We will simply store the values that we receive from our default **`InputFormat`** (the **`Long`** associated with the **`LongWritable`** and the **`String`** associated with **`Text`**) into our custom data type.
- We will store those values as we receive them in the **`map()`** method, and pass our custom data type to the reduce as a value. We will use a **`null`** "dummy" key. This will cause all our values to be sent to the same reducer. (Not generally a good idea, but okay for our code).
- On the reduce side, we will extract our **`Long`** and **`String`** from our custom data type, and write them as output to HDFS.

Task 1: Open and review lab sample Java file

Activity Procedure

Step 1. Change directory to where you will edit the java code:

```
$ cd ~/labs/writable
```

Step 6. Open and start editing your Custom Writable sample file

```
$ vi MyWritable.java
```

This file already contains prewritten code from a previous lab to save the time. We will add additional code, which is specific to the new **`Writable`** type.

Task 2: Add code for the new Writable Type

Activity Procedure

Step 1. We are creating a new inner-class **`PairWritable`** in **`MyWritable`** class, which will implement the **`Writable`** interface. The code is prewritten to create this inner-class.

Step 2. We need 2 private variables for the subclass PairWritable, which we make a new Writable type. Add these variables in the PairWritable subclass

```
//Private Variables for PairWritable datatype
private Long myLong;
private String myString;
```

Step 3. We will override existing methods in Writable interface by writing our own code for these methods. We read value to the variables from input using 'readFields' method & the write their value using 'write' method. Add following code in the subclass PairWritable to override these methods.

```
/* Override the Hadoop serialization/Writable
interface methods */
@Override
public void readFields(DataInput in) throws IOException {
     myLong = in.readLong();
     myString = in.readUTF();
}

@Override
public void write(DataOutput out) throws IOException {
     out.writeLong(myLong);
     out.writeUTF(myString);
}
//End of Implementation
```

Step 4. We have a getter & setter method for each of the variables `myLong` and `myString`. These methods help to initialize and get values for these variables in the code. The code for getters and setters is prewritten.

Step 5. Let's work on the `Mapper` implementation `MyMapper`. It expects two sets of data types: Input Key & Value and Output Key & Value. The default data type for input Key is `LongWritable` and for input Value is the `Text` type.

As discussed at the start, the data type for the output key is `NullWritable` and for the output value is `PairWritable`.

The code for `Mapper` class is prewritten.

Step 6. Finish the `map()` method to read data from input and transform it in `NullWritable` & `PairWritable` format by using the following code:

```
PairWritable mypair = new PairWritable();
mypair.set(inputKey.get(),inputValue.toString());
output.collect(NullWritable.get(), mypair);
```

Step 7. Now we will work on the Reducer implementation `MyReducer`. It also expects two sets of data types: Input Key & Value and Output Key & Value. In this case, the data type for input Key is `NullWritable` and the data type for the input Value is `PairWritable`.

As discussed at the start, the data type for the output key is `LongWritable` and for the output value is `Text`.

The code for `Reducer` implementation class is prewritten.

Step 8. Write code in `reduce()` method to read data from input and transform it into `LongWritable` & `Text` format by using following code:

```
while(myPair.hasNext()) {

     PairWritable myPairWritable = myPair.next();

     Long longValue = myPairWritable.getLong();
     String stringValue = myPairWritable.getString();
     output.collect(new LongWritable(longValue),
     new Text(stringValue));
}
```

Step 9. In the `MyWritable main()` method, we need to set Output of Mapper and Reducer using the following code:

```
     /* Default map output data types are insufficient.
     Must specify.*/

     conf.setMapOutputKeyClass(NullWritable.class);
     conf.setMapOutputValueClass(PairWritable.class);

     // Set HDFS output key/value data types
     conf.setOutputKeyClass(LongWritable.class);
     conf.setOutputValueClass(Text.class);
```

Task 3: Compile and run the MapReduce job

Activity Procedure

Step 1. Let's compile the code and make a jar file from the CLASS files.

```
$ javac MyWritable.java

$ jar cvf wr.jar *.class
```

Step 2. We will use your input file, which you created in the previous lab. So copy it to the current directory and also to HDFS:

```
$ cp ~/labs/mrsimple/input.txt .

$ hadoop fs -put input.txt writable/input.txt
```

Step 3. Use following command to see your program in the action, this will take some time to run:

```
$ hadoop jar wr.jar MyWritable writable/input.txt writable/output
```

Step 4. Let's check the output of the program:

```
$ hadoop fs -cat writable/output/part-00000
```

Lab 4.2: Create a Custom InputFormat

Description:

- The basic InputFormat class itself is already implemented in the sample code. The real work is in the implementation of interface RecordReader. We will implement the next() method of RecordReader in this lab.
- The next() method is called by Hadoop in order to get the next key and value pair. You will implement that code.
- RecordReader is implemented as a nested class within the InputFormat, and has a sufficiently functional method to read the block of data (our input split) and parse to a ":" delimiter. Within next(), you will call that method to acquire the next value from the block.
- You must also keep a counter to keep track of the file offset. The initial offset will have been calculated in the constructor.
- The actual code you will create is fairly trivial, as java goes. The real work is in understanding how RecordReader is called and what it is expected to do.
- The code that reads to the ":" delimiter does not check previous blocks to see if a record is continued into the current block, but does read into the next block to get the current record. This is "broken", but not really worth fixing for classroom purposes. The extra 30 lines of code to do that would obscure the basic operation of the code. The Hadoop source tree implements this functionality properly. You should expect to see the last record of a block twice.

Task 1: Open and review lab sample java file

Activity Procedure

Step 1. Change directory to where you will edit the java code. You will find 2 files in this directory.

```
$ cd ~/labs/iformat
$ ls -l ~/labs/iformat

-rw-r--r-- 1 train train 5852 May 21 14:59 MyTextInputFormat.java
-rw-r--r-- 1 train train 2547 May 21 14:59 UseMyTextInputFormat.java
```

MyTextInputformat.java is a class where you will define a new input format. **UseMyTextInputFormat.java** is the main java program, which will use this new input format. We will modify both the files in this exercise.

Step 2. First look into the new input format class.

```
$ vi MyTextInputFormat.java
```

This file has already prewritten code. We will add the code, which is relevant for this exercise.

Task 2: Add code for the new Input Format

Activity Procedure

Step 1. Review the java method "**readToDelim**" in MyTextInputFormat class.

It delimits input data using COLON (":") as a delimiter. Usually, it is developer's responsibility to write his own method to tell the program how to delimit input data. The code and logic could be tricky. That's why we are not writing the logic in this exercise.

Step 2. We will override only one method "**next**" in the RecordReader interface. Look for "**next**" method in the MyTextInputFormat class. Add following code inside this method:

```
while (pos < end) {  // still bytes in split
  key.set(String.valueOf(pos));    // set key to byte offset
  // should req read of no more than end-pos
  int nChar = readToDelim(value);  // read to colon delim
  if (nChar == 0) { return false; }
  pos += nChar;
   return true;
 }
 return false;  // we must have hit End of Split
```

We are reading data character by character from first line to the last line in this method and also looking for ":" as record breaker.

Step 3. Open the main java file (**UseMyTextInputFormat.java**) where we are implementing the new input format.

```
$ vi UseMyTextInputFormat.java
```

Step 4. Add following line in the main method to set a new input format:

```
conf.setInputFormat(MyTextInputFormat.class);
```

Task 3: Compile and run the MapReduce job

Activity Procedure

Step 1. Let's compile the code and make a jar file from the CLASS files.

```
$ javac UseMyTextInputFormat.java MyTextInputFormat.java
$ jar cvf iformat.jar *.class
```

Step 2. Create your input file (input.txt) having a sentence delimited by ":"

```
e.g.
This: is: my: first: custom: Input: Format
```

Step 3. Copy your input file to HDFS.

```
$ hadoop fs -put input.txt iformat/input.txt
```

Step 4. Use following command to see your job in the action

```
$ hadoop jar iformat.jar UseMyTextInputFormat iformat/input.txt
iformat/output
```

Step 5. Let's check the output of the program:

```
$ hadoop fs -cat iformat/output/part-00000
```

You will see that input data is delimited using ":" and the output shows that end of the line is each ":".

```
0 This:
13 first:
20 custom:
21 custom:
28 Input:
35 Format
5  is:
9  my:
```

Note: Notice at least one word is repeating in the output. In this case 'custom' is repeating. The default setting for this environment is two Mappers per job. So there is at least one overlapped word on which both the mappers worked. The code is not optimized enough to take care of this exception.

Lab 4.3: Use Combiner with MapReduce

Description:

- Run a simple MapReduce job
- Add a combiner to enhance performance of the job
- Compare outputs and runtimes

We will use the New York Stock Exchange records of stock prices from 1962-2010. We will edit a MapReduce job that finds the maximum high price for each stock over that time.

Task 1: Locate the Input Data into your Cluster

Activity Procedure

Step 1. Change to the directory where the data is located.

```
$ cd ~/labs/mrcombiner
```

Step 2. Create a directory in HDFS

```
$ hadoop fs -mkdir mrcombiner/input
```

Step 3. Copy data from this directory to HDFS

```
$ hadoop fs -put ./n* mrcombiner/input/
```

Step 4. Check that all of the files are there. From your command line type:

```
$ hadoop fs -ls mrcombiner/input/nyse/
```

You should see one .csv file for each letter of the alphabet.

Task 2: Compile and Run The Job

Activity Procedure

Now let's see the initial code in action. The map function is processing one line at a time. Since this file contains more data than we need, we will need to parse out the fields we're interested in and send those to the reducer. The stock symbol field will be our key and the stock high price field will be our value.

Step 1. Look over the Java source (especially the Mapper and Reducer code)

```
$ vi Combiner.java  OR  $ cat Combiner.java
```

Step 2. Compile the java source

```
$ javac Combiner.java
```

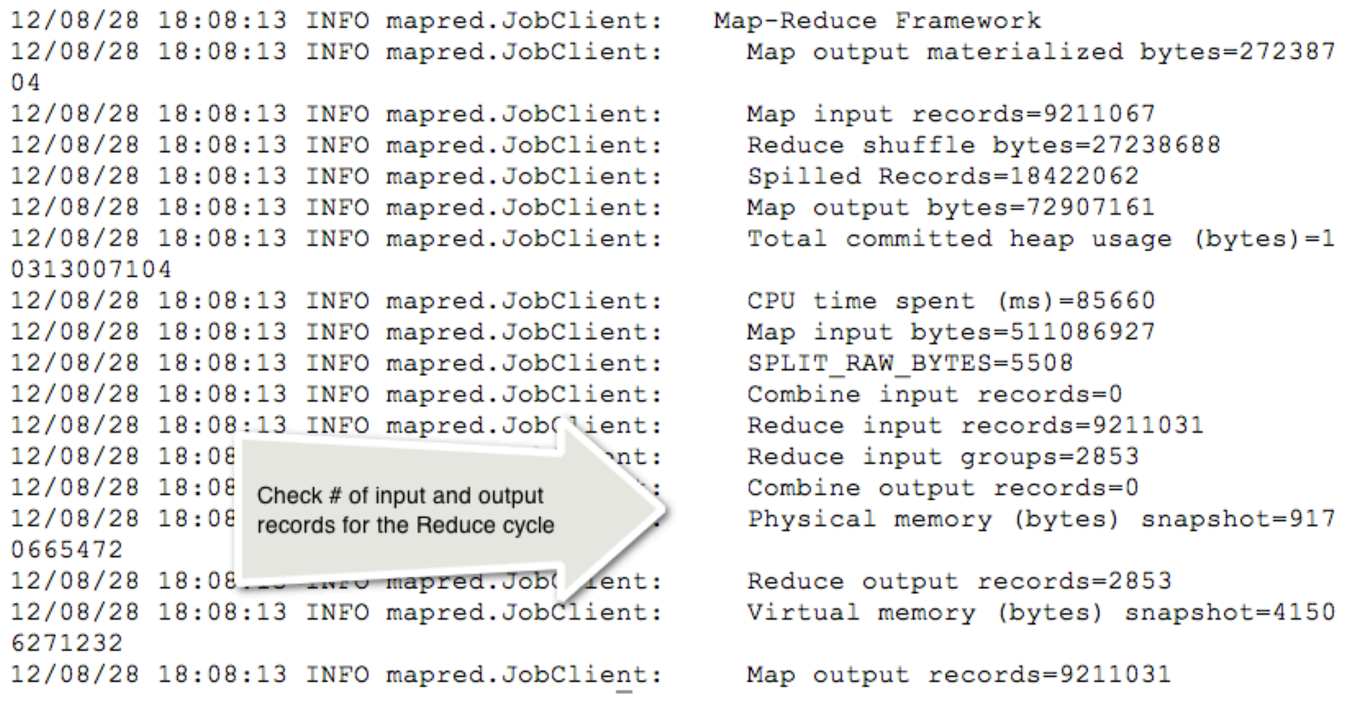
Step 3. Create a jar from the class file

```
$ jar cvf myjar.jar *.class
```

Step 4. Run the MapReduce job by doing:

```
$ hadoop jar myjar.jar Combiner mrcombiner/input/nyse mrcombiner/output
```

You will see diagnostics from Hadoop as it runs the job. When it is finished it will tell you how many output records it has written.



```
12/08/28 18:08:13 INFO mapred.JobClient:   Map-Reduce Framework
12/08/28 18:08:13 INFO mapred.JobClient:     Map output materialized bytes=272387
04
12/08/28 18:08:13 INFO mapred.JobClient:     Map input records=9211067
12/08/28 18:08:13 INFO mapred.JobClient:     Reduce shuffle bytes=27238688
12/08/28 18:08:13 INFO mapred.JobClient:     Spilled Records=18422062
12/08/28 18:08:13 INFO mapred.JobClient:     Map output bytes=72907161
12/08/28 18:08:13 INFO mapred.JobClient:     Total committed heap usage (bytes)=1
0313007104
12/08/28 18:08:13 INFO mapred.JobClient:     CPU time spent (ms)=85660
12/08/28 18:08:13 INFO mapred.JobClient:     Map input bytes=511086927
12/08/28 18:08:13 INFO mapred.JobClient:     SPLIT_RAW_BYTES=5508
12/08/28 18:08:13 INFO mapred.JobClient:     Combine input records=0
12/08/28 18:08:13 INFO mapred.JobClient:     Reduce input records=9211031
12/08/28 18:08                        nt:     Reduce input groups=2853
12/08/28 18:08                         :     Combine output records=0
12/08/28 18:08                         :     Physical memory (bytes) snapshot=917
0665472
12/08/28 18:08       mapred.Job    ent:     Reduce output records=2853
12/08/28 18:08:13 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=4150
6271232
12/08/28 18:08:13 INFO mapred.JobClient:     Map output records=9211031
```

Task 3: Add a Combiner

Activity Procedure

The reduce function is presented each key and an iterator that will produce all values associated with that key. We will need to determine the maximum value for each key.

Step 1. Comment out the Reducer assignment. Then, add a Combiner to the map-side of the job. Use the following method:

```
//conf.setReducerClass(Reduce.class);
conf.setCombinerClass(Reduce.class);
```

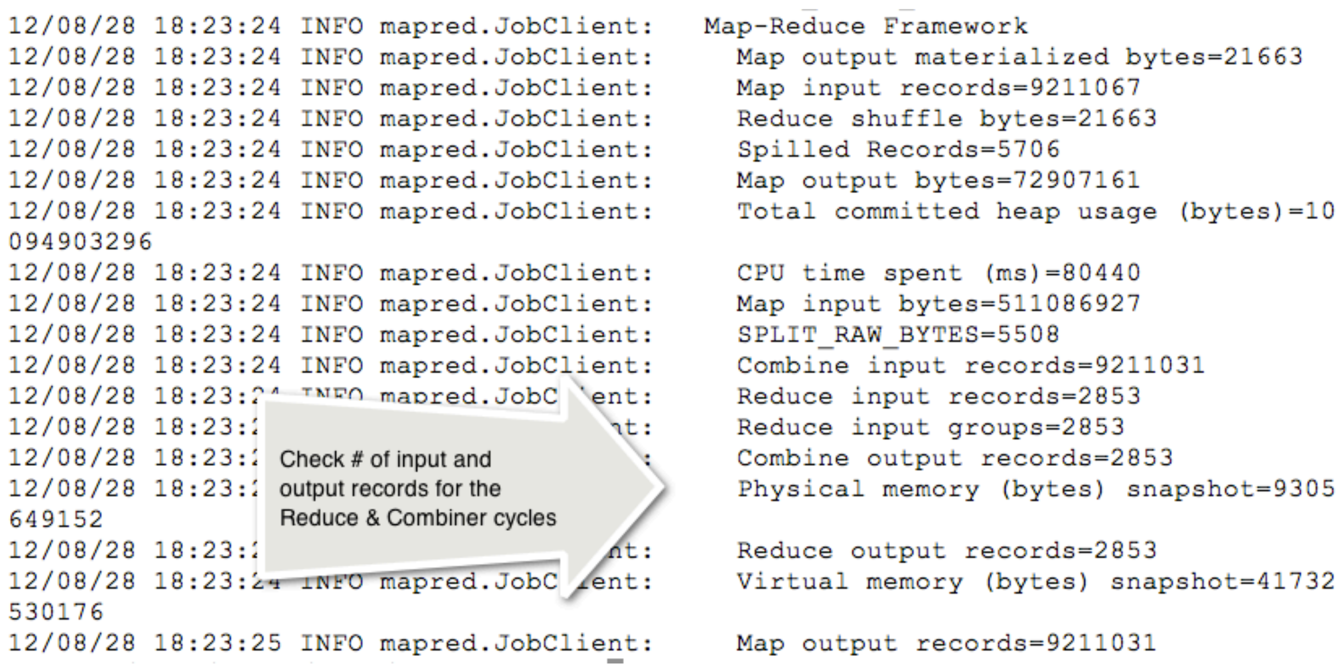
Step 2. Compile, jar and run your new job.

```
$ javac Combiner.java
$ jar cvf myjar.jar *.class
```

Step 3. Run the MapReduce job by doing:

```
$ hadoop jar myjar.jar Combiner mrcombiner/input/nyse mrcombiner/output1
```

You will see diagnostics from Hadoop as it runs the job. When it is finished it will tell you how many output records were written.



```
12/08/28 18:23:24 INFO mapred.JobClient:   Map-Reduce Framework
12/08/28 18:23:24 INFO mapred.JobClient:     Map output materialized bytes=21663
12/08/28 18:23:24 INFO mapred.JobClient:     Map input records=9211067
12/08/28 18:23:24 INFO mapred.JobClient:     Reduce shuffle bytes=21663
12/08/28 18:23:24 INFO mapred.JobClient:     Spilled Records=5706
12/08/28 18:23:24 INFO mapred.JobClient:     Map output bytes=72907161
12/08/28 18:23:24 INFO mapred.JobClient:     Total committed heap usage (bytes)=10
094903296
12/08/28 18:23:24 INFO mapred.JobClient:     CPU time spent (ms)=80440
12/08/28 18:23:24 INFO mapred.JobClient:     Map input bytes=511086927
12/08/28 18:23:24 INFO mapred.JobClient:     SPLIT_RAW_BYTES=5508
12/08/28 18:23:24 INFO mapred.JobClient:     Combine input records=9211031
12/08/28 18:23:   INFO mapred.JobC  ent:     Reduce input records=2853
12/08/28 18:23:                       t:     Reduce input groups=2853
12/08/28 18:23:                        :     Combine output records=2853
12/08/28 18:23:                        :     Physical memory (bytes) snapshot=9305
649152
12/08/28 18:23:                       t:     Reduce output records=2853
12/08/28 18:23:   INFO mapred.JobC  ent:     Virtual memory (bytes) snapshot=41732
530176
12/08/28 18:23:25 INFO mapred.JobClient:     Map output records=9211031
```

Note: this time combiner at Map side processed all the records. The number of input and output records for reducer is same.

Lab 4.4: Create a Custom Partitioner

Description:

- Write a MapReduce Java program that uses a custom partitioner
- Run the resulting program

We will use the New York Stock Exchange records of stock prices from 1962-2010. We will write a Map-Reduce program that sorts the data by stock ticker symbol, and places all of the stocks with ticker symbols that begin with A in the first part file, those beginning with B in the second part file, etc.

Task 1: Set the Partitioner Class in `main()`

Activity Procedure

A partitioner is provided with each record coming out of the map and asked to indicate which partition the record should be sent to. An important feature of a partitioner is it should be consistent. All records with the same key must go to the same reducer. The partitioner is also told how many reducers there are.

For those who are Java developers, we provide a detailed description of what the **`main()`** method of this class must accomplish. For those who are less comfortable with Java, we provide a step-by-step list of instructions below the description.

Class	MRPartitioner
Responsibilities	For this lab exercise only modify the **`main()`** method, nothing else. We have provided an alphabetic partitioner for you, with a single **`getPartition()`** method. It works as is so don't modify this method, but feel free to read it. In the **`main()`** method, simply set the map reduce job configuration partitioner class to use the alphabetic partitioner. In the **`main()`** method, set the number of reduce tasks to twenty six.
Collaborators	org.apache.hadoop.mapred.JobConf AlphabeticPartitioner
Other	NA

Step 1. Change to the directory where the program resides:

```
$ cd ~/labs/mrpart
```

Step 2. Open Partitioner.java in your editor (using `vi` or nano):

```
$ vi MRPartitioner.java
```

Step 3. Modify the `main()` method code to set the partitioner class. See the

/* TODO: … */

…comments in the code. Leave your editor up for the next task.

Activity Verification

You have completed this task when you attain these results:

```
conf.setPartitionerClass(AlphabeticPartitioner.class);
```

Task 2: Configure Hadoop to use your custom Partitioner

By default Hadoop uses the `org.apache.hadoop.mapred.lib.HashPartitioner` (which partitions your data by the results of calls to `hashCode()` on it) and has only one reducer. In this case we want to use the custom partitioner, and we want to set the **number of reducers to 26**, so that we get one file for each letter of the alphabet.

Activity Procedure

Step 1. Modify the `main()` method to set the number of reducers which will be used. See the

/* TODO: … */

….comments in the code.

Activity Verification

You have completed this task when you attain these results:

```
conf.setNumReduceTasks(26);
```

Task 4: Compile and Run Your Code

Activity Procedure

Now let's see your code in action.

Step 1. Compile the java file

```
$ javac MRPartitioner.java
```

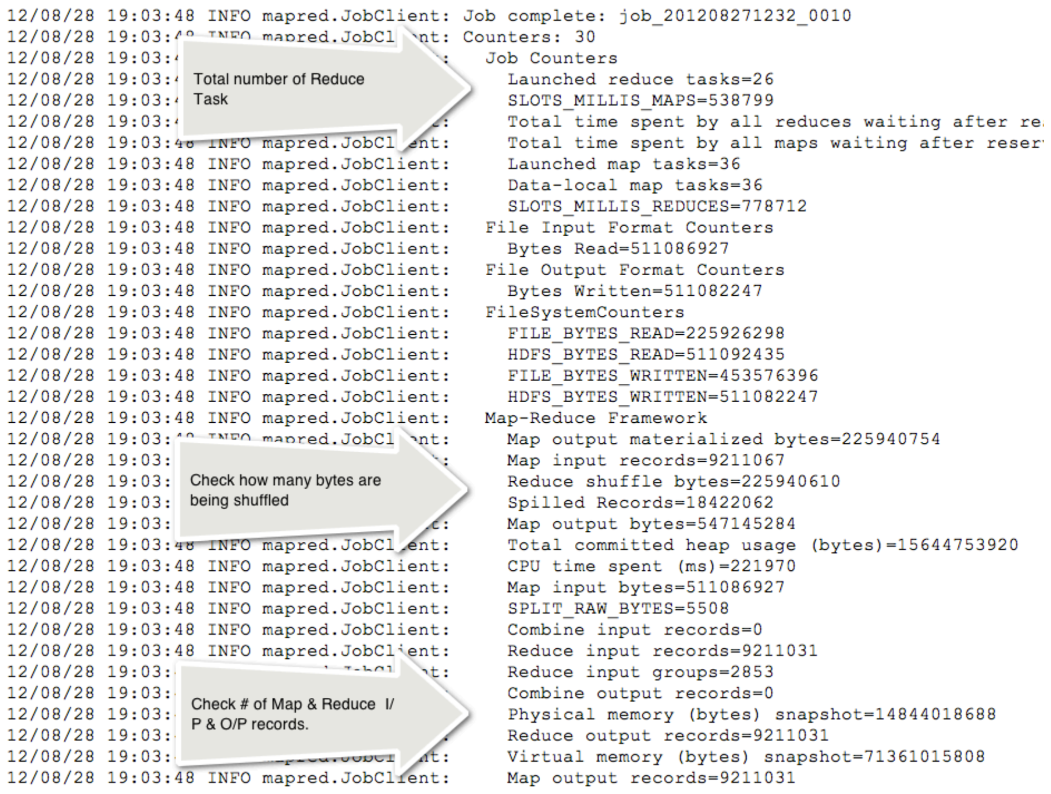
Step 2. Jar all the class files

```
$ jar cvf mrpart.jar *.class
```

Step 3. Run your new program on Hadoop:

```
$ hadoop jar mrpart.jar MRPartitioner mrcombiner/input/nyse mrpart/output
```

You will see diagnostics from Hadoop as it runs the job:

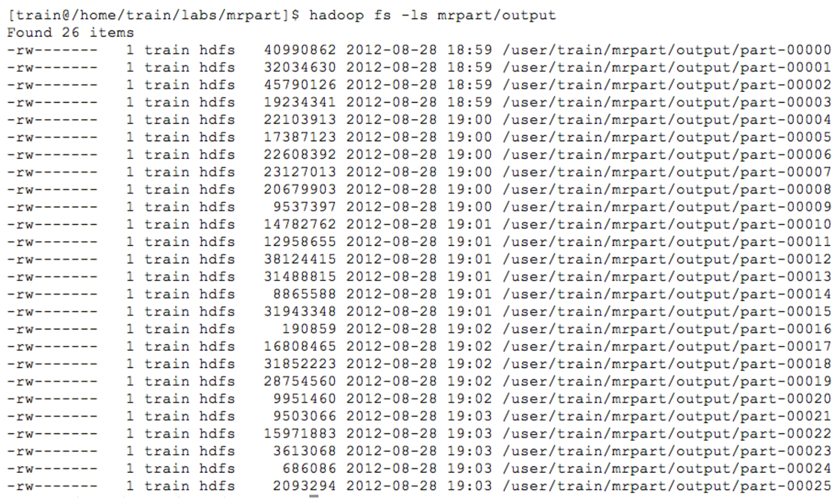

Total number of Reduce Task
Check how many bytes are being shuffled
Check # of Map & Reduce I/P & O/P records.

```
12/08/28 19:03:48 INFO mapred.JobClient: Job complete: job_201208271232_0010
12/08/28 19:03:          mapred.JobCl   nt: Counters: 30
12/08/28 19:03:                               Job Counters
12/08/28 19:03:                                 Launched reduce tasks=26
12/08/28 19:03:                                 SLOTS_MILLIS_MAPS=538799
12/08/28 19:03:                                 Total time spent by all reduces waiting after re
12/08/28 19:03:4       mapred.JobCl  ent:       Total time spent by all maps waiting after reser
12/08/28 19:03:48 INFO mapred.JobClient:        Launched map tasks=36
12/08/28 19:03:48 INFO mapred.JobClient:        Data-local map tasks=36
12/08/28 19:03:48 INFO mapred.JobClient:        SLOTS_MILLIS_REDUCES=778712
12/08/28 19:03:48 INFO mapred.JobClient:      File Input Format Counters
12/08/28 19:03:48 INFO mapred.JobClient:        Bytes Read=511086927
12/08/28 19:03:48 INFO mapred.JobClient:      File Output Format Counters
12/08/28 19:03:48 INFO mapred.JobClient:        Bytes Written=511082247
12/08/28 19:03:48 INFO mapred.JobClient:      FileSystemCounters
12/08/28 19:03:48 INFO mapred.JobClient:        FILE_BYTES_READ=225926298
12/08/28 19:03:48 INFO mapred.JobClient:        HDFS_BYTES_READ=511092435
12/08/28 19:03:48 INFO mapred.JobClient:        FILE_BYTES_WRITTEN=453576396
12/08/28 19:03:48 INFO mapred.JobClient:        HDFS_BYTES_WRITTEN=511082247
12/08/28 19:03:48 INFO mapred.JobClient:      Map-Reduce Framework
12/08/28 19:03:        mapred.JobCl   nt:       Map output materialized bytes=225940754
12/08/28 19:03:                                 Map input records=9211067
12/08/28 19:03:                                 Reduce shuffle bytes=225940610
12/08/28 19:03:                                 Spilled Records=18422062
12/08/28 19:03:                                 Map output bytes=547145284
12/08/28 19:03:4       mapred.JobCl  ent:       Total committed heap usage (bytes)=15644753920
12/08/28 19:03:48 INFO mapred.JobClient:        CPU time spent (ms)=221970
12/08/28 19:03:48 INFO mapred.JobClient:        Map input bytes=511086927
12/08/28 19:03:48 INFO mapred.JobClient:        SPLIT_RAW_BYTES=5508
12/08/28 19:03:48 INFO mapred.JobClient:        Combine input records=0
12/08/28 19:03:48 INFO mapred.JobClient:        Reduce input records=9211031
12/08/28 19:03:                                 Reduce input groups=2853
12/08/28 19:03:                                 Combine output records=0
12/08/28 19:03:                                 Physical memory (bytes) snapshot=14844018688
12/08/28 19:03:                                 Reduce output records=9211031
12/08/28 19:03:                                 Virtual memory (bytes) snapshot=71361015808
12/08/28 19:03:48 INFO mapred.JobClient:        Map output records=9211031
```

Step 4. Take a look at your resulting data:

```
$ hadoop fs -ls  mrpart/output
```

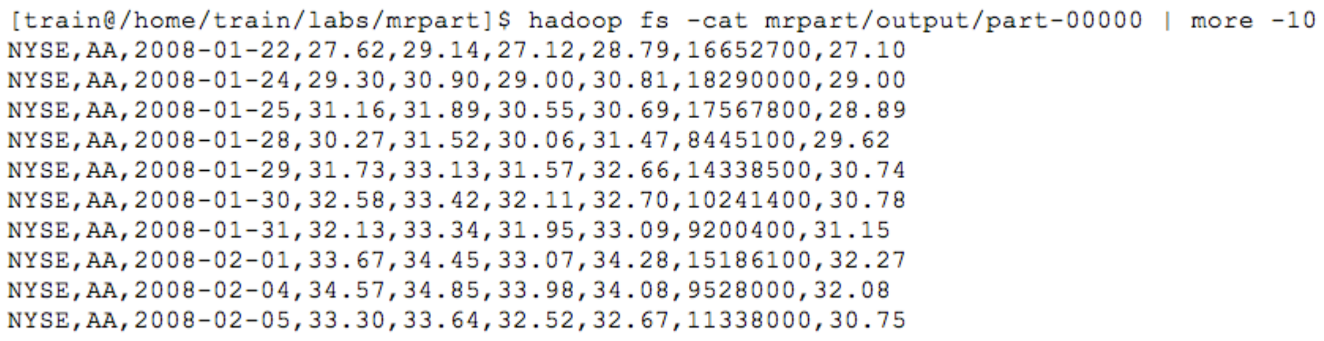
You will see 26 output files (part-000xx) files (one for each reducer):



```
[train@/home/train/labs/mrpart]$ hadoop fs -ls mrpart/output
Found 26 items
-rw-------   1 train hdfs   40990862 2012-08-28 18:59 /user/train/mrpart/output/part-00000
-rw-------   1 train hdfs   32034630 2012-08-28 18:59 /user/train/mrpart/output/part-00001
-rw-------   1 train hdfs   45790126 2012-08-28 18:59 /user/train/mrpart/output/part-00002
-rw-------   1 train hdfs   19234341 2012-08-28 18:59 /user/train/mrpart/output/part-00003
-rw-------   1 train hdfs   22103913 2012-08-28 19:00 /user/train/mrpart/output/part-00004
-rw-------   1 train hdfs   17387123 2012-08-28 19:00 /user/train/mrpart/output/part-00005
-rw-------   1 train hdfs   22608392 2012-08-28 19:00 /user/train/mrpart/output/part-00006
-rw-------   1 train hdfs   23127013 2012-08-28 19:00 /user/train/mrpart/output/part-00007
-rw-------   1 train hdfs   20679903 2012-08-28 19:00 /user/train/mrpart/output/part-00008
-rw-------   1 train hdfs    9537397 2012-08-28 19:00 /user/train/mrpart/output/part-00009
-rw-------   1 train hdfs   14782762 2012-08-28 19:01 /user/train/mrpart/output/part-00010
-rw-------   1 train hdfs   12958655 2012-08-28 19:01 /user/train/mrpart/output/part-00011
-rw-------   1 train hdfs   38124415 2012-08-28 19:01 /user/train/mrpart/output/part-00012
-rw-------   1 train hdfs   31488815 2012-08-28 19:01 /user/train/mrpart/output/part-00013
-rw-------   1 train hdfs    8865588 2012-08-28 19:01 /user/train/mrpart/output/part-00014
-rw-------   1 train hdfs   31943348 2012-08-28 19:01 /user/train/mrpart/output/part-00015
-rw-------   1 train hdfs     190859 2012-08-28 19:02 /user/train/mrpart/output/part-00016
-rw-------   1 train hdfs   16808465 2012-08-28 19:02 /user/train/mrpart/output/part-00017
-rw-------   1 train hdfs   31852223 2012-08-28 19:02 /user/train/mrpart/output/part-00018
-rw-------   1 train hdfs   28754560 2012-08-28 19:02 /user/train/mrpart/output/part-00019
-rw-------   1 train hdfs    9951460 2012-08-28 19:02 /user/train/mrpart/output/part-00020
-rw-------   1 train hdfs    9503066 2012-08-28 19:03 /user/train/mrpart/output/part-00021
-rw-------   1 train hdfs   15971883 2012-08-28 19:03 /user/train/mrpart/output/part-00022
-rw-------   1 train hdfs    3613068 2012-08-28 19:03 /user/train/mrpart/output/part-00023
-rw-------   1 train hdfs     686086 2012-08-28 19:03 /user/train/mrpart/output/part-00024
-rw-------   1 train hdfs    2093294 2012-08-28 19:03 /user/train/mrpart/output/part-00025
```

```
$ hadoop fs -cat mrpart/output/part-00000 | more -10
```

This file should only contain stocks with ticker symbols beginning with A. Also all of the entries in the file should be sorted by ticker symbol.



```
[train@/home/train/labs/mrpart]$ hadoop fs -cat mrpart/output/part-00000 | more -10
NYSE,AA,2008-01-22,27.62,29.14,27.12,28.79,16652700,27.10
NYSE,AA,2008-01-24,29.30,30.90,29.00,30.81,18290000,29.00
NYSE,AA,2008-01-25,31.16,31.89,30.55,30.69,17567800,28.89
NYSE,AA,2008-01-28,30.27,31.52,30.06,31.47,8445100,29.62
NYSE,AA,2008-01-29,31.73,33.13,31.57,32.66,14338500,30.74
NYSE,AA,2008-01-30,32.58,33.42,32.11,32.70,10241400,30.78
NYSE,AA,2008-01-31,32.13,33.34,31.95,33.09,9200400,31.15
NYSE,AA,2008-02-01,33.67,34.45,33.07,34.28,15186100,32.27
NYSE,AA,2008-02-04,34.57,34.85,33.98,34.08,9528000,32.08
NYSE,AA,2008-02-05,33.30,33.64,32.52,32.67,11338000,30.75
```

Activity Verification

You have completed this task when you attain these results:

- You have compiled the **`MRPartitioner.jar`** with your changes
- Hadoop has run your program.
- You have verified that the data is sorted and each file contains only stocks with ticker symbols starting with the appropriate letter.

Lab 4.5: Demo MapReduce Streaming Using Python

Description:

- Write a Map-Reduce Streaming program

Streaming is a way to use Map-Reduce without needing to write Java. It can be used with existing executables (such as shell tools like wc) or with ones you write, in any language that can be run on the cluster. We will use the New York Stock Exchange records of stock prices from 1962-2010. We will write a streaming application program that prints the unique list of ticker symbols available in the data.

Task 1: Examine the Python map application

Activity Procedure

Step 1. Change directory to the location of python code that will be used:

```
cd ~/labs/mrstream
```

Step 2. View the python script

```
$ less parse.py
```

Note: This implements the equivalent of awk -F, {'print $2'}

Activity Verification

You have completed this task when you attain these results;

- You've located and examined the example code.

Task 2: Ensure you have input data

Activity Procedure

Step 1. Examine the input data from command line:

```
$ hadoop fs -cat mrcombiner/input/nyse/NYSE_daily_prices_A.csv | head -10
```

Activity Verification

You have completed this task when you attain these results:

- You've located and examined the data files.

Task 3: Run the Streaming Job

Activity Procedure

We have already looked at the Python script we will use as a map function. For the reduce function we will use the shell utility uniq, which when provided sorted output returns a unique set of values. Since our Python program is not already on the cluster we will need to tell Hadoop to ship it as part of our job. uniq will already be present on the cluster machines, so there will be no need to ship it.

Step 1. Run the streaming example. The command line (ALL OF THIS IS TO BE ON ONE LINE) will be:

```
$ hadoop jar /usr/share/hadoop/contrib/streaming/hadoop-streaming-
   1.0.2.jar \
-input mrcombiner/input/nyse \
-output mrstream/output \
-mapper parse.py \
-reducer /usr/bin/uniq -file parse.py
```

Task 4: Display the Streaming Job

Activity Procedure

Step 1. Display the job output:

```
$ hadoop fs -cat mrstream/output/part-00000
```

This file should only contain one record for each stock ticker symbol.

Activity Verification

You have completed this task when you attain these results;

- Hadoop has run your streaming job.
- You have verified that the data contains one record for each stock ticker symbol.

```
YPF
YSI
YUM
YZC
ZAP
ZEP
ZF
ZLC
ZMH
ZNH
ZNT
ZQK
ZTR
ZZ
stock_symbol
```

Lab 4.6: Use Distributed Cache in MapReduce

Description:

- Write a Map-Reduce Java program that uses the distributed cache.
- Run the resulting program

We will use the New York Stock Exchange records of stock prices from 1962-2010. We will write a Map-Reduce program that finds the adjusted closing price for each day that a stock reported dividends. This means that we will need to join the daily prices data with the dividends data, which is in a separate file. The dividends data is small, and can easily fit into memory. So rather than use the reducer to do our join, which you would do if the data was large, we will load the dividends data into the distributed cache. Each map task can then read it into a hash table and do the join. The purpose of the distributed cache is to efficiently distribute data, jar files, scripts, libraries, etc. to every map or reduce task in a job.

Task 1: Copy the Example to a Java File

Activity Procedure

Step 1. Change directory to where your Java code is located:

```
$cd ~/labs/mrdcache
```

Step 2. Bring up DCache.java in your editor:

```
$ vi DCache.java
```

Examine the program and leave your editor up.

Activity Verification

You have completed this task when you attain these results;

- You've located the example code.

Task 2: Configure your Job to Get the Local Cache File for your Mapper

For those who are Java developers, we provide a detailed description of what the class must accomplish. For those who are less comfortable with Java, we provide a step-by-step list of instructions below the description.

Classes	**`DCache`** **`Map`**
Responsibilities	Within the **`Map configure()`** method work with the public static **`DistributedCache`** method **`getLocalCacheFiles(JobConf job)`** to set the value of the **`localFiles Path`** array. Within the **`Map configure()`** method work with the **`FileReader`**, to wrap the

	first **Path** in the **localFiles Path** array. Within the **DCache main()** method, work with the public static **DistributedCache addCacheFile(…)** method to add the new **URI** (which **URI** is wrapped around the third argument submitted to the application, or **args[2]**).
Collaborators	**org.apache.hadoop.filecache.DistributedCache** **org.apache.hadoop.mapred.JobConf** **org.apache.hadoop.fs.Path** **java.net.URI**
Other	NA

Activity Procedure

Step 1. Find the method public void configure(JobConf job){ /* … */ }

Step 2. In **configure(…)**, where you see the first **/* TODO: … */** invoke the public static **getLocalCacheFiles(…)** method on the **DistributedCache** and store the value returned in the instance variable crated for this purpose.

You have completed this task when you attain these results:

```
localFiles = DistributedCache.getLocalCacheFiles(job);
```

Task 3: In your Map Class Open the File from the Distributed Cache

Activity Procedure

The map function will need to open the file from the distributed cache and load its contents into a hash table. Most of the code to do this has been added to the configure method where you just added a line to get the list of files in the distributed cache. Files in the distributed cache are placed on the local file system of the task, so they can be read using regular file reading tools.

Step 1. In **configure(…)**, where you see the second **/* TODO: … */** instantiate the **FileReader** and wrap it around the **localFiles Path** array at index zero as a String, by using a **toString()** invocation on that index.

You have completed this task when you attain these results:

```
FileReader fr = new FileReader(localFiles[0].toString());
```

Task 4: Tell conf to add the cache file

Activity Procedure

Step 1. Where you see the third `/* TODO: … */` in the `main()` method invoke the public static `DistributedCache` method `addCacheFile(…)` passing two arguments, a new `URI` instance which is wrapped around `args[2]`, and the reference to the `JobConf` object.

You have completed this task when you attain these results;

```
DistributedCache.addCacheFile(new URI(args[2]), conf);
```

Task 5: Compile and Run Your Code

Activity Procedure

Step 1. Compile the java and create a jar file

```
$ javac DCache.java
$ jar cvf mrdcache.jar *.class
```

Step 2. Load the dividend file into HDFS

```
$ hadoop fs –mkdir mrdcache/input
$ hadoop fs -put dividends mrdcache/input/
```

Step 3. Run the program using following command:

```
$ hadoop jar mrdcache.jar DCache mrcombiner/input/nyse mrdcache/output
mrdcache/input/dividends
```

Step 4. Look at the directory where you data was written.

```
$ hadoop fs -ls  mrdcache/output
```

You should see the file part-00000

Step 5. Take a look at your result data:

```
$hadoop fs -cat mrdcache/output/part-00000
```

This file should only contain entries for dates when a dividend was paid.

Activity Verification

You have completed this task when you attain these results;

- You have compiled the tutorial.jar with your changes
- Hadoop has run your program.
- You have verified that the data now contains only record for dates when dividends were paid.

```
…
CRT,1992-11-23,2.68
CRT,1992-10-26,2.73
CRT,1992-09-24,2.44
CRT,1992-08-25,2.22
CRT,1992-07-27,2.06
CRT,1992-06-24,1.91
CRT,1992-05-22,1.81
CRT,1992-04-24,1.77
CRT,1992-03-25,1.75
```

Lab 5.1: Monitoring a Job with JobTracker

Task 1: Map Reduce – Execution Phase

Activity Procedure and Verification

Step 1. Change to the directory with the scripts

```
$ cd ~/labs/mroperation
```

Step 2. Copy input data file to this directory

```
$ hadoop fs -put visit_5000000.txt mroperation/input/visit_5000000.txt
```

Step 3. Examine the input file

```
$ hadoop fs -ls mroperation/input

Found 1 items
-rw-------   1 train hdfs  254918253 2012-08-29 13:22
   /user/train/mroperation/input/visit_5000000.txt
```

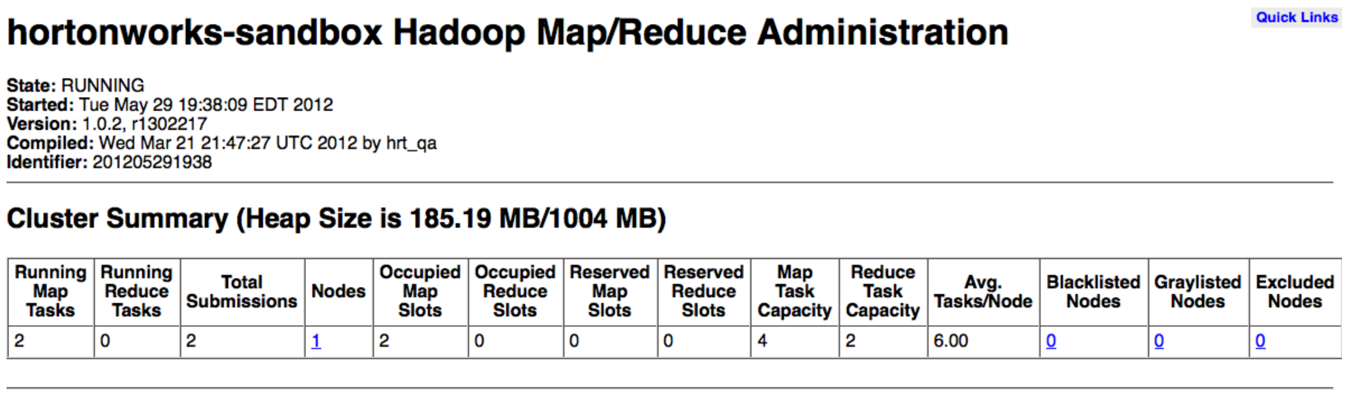
Step 4. Compile the java file & create a jar file

```
$ javac PageStat.java
$ jar cvf mroperation.jar *.class
```

Step 5. Open a browser to **http://<job_tracker_node>:50030** which is the jobtracker Web UI admin. To be able to view all these steps, the map reduce job needs to be running before completion. If it is completed already, you can re-run the above map_reduce job.b from step 8.

NOTE: if any Link does not work on these pages, fix the URL by replacing servername with your EC2 hostname.

e.g: if the URL is: `http://hortonworks-sandbox.localdomain:50030/logs` Then replace it with `http://<EC2 Cluster Name>:50030/logs`



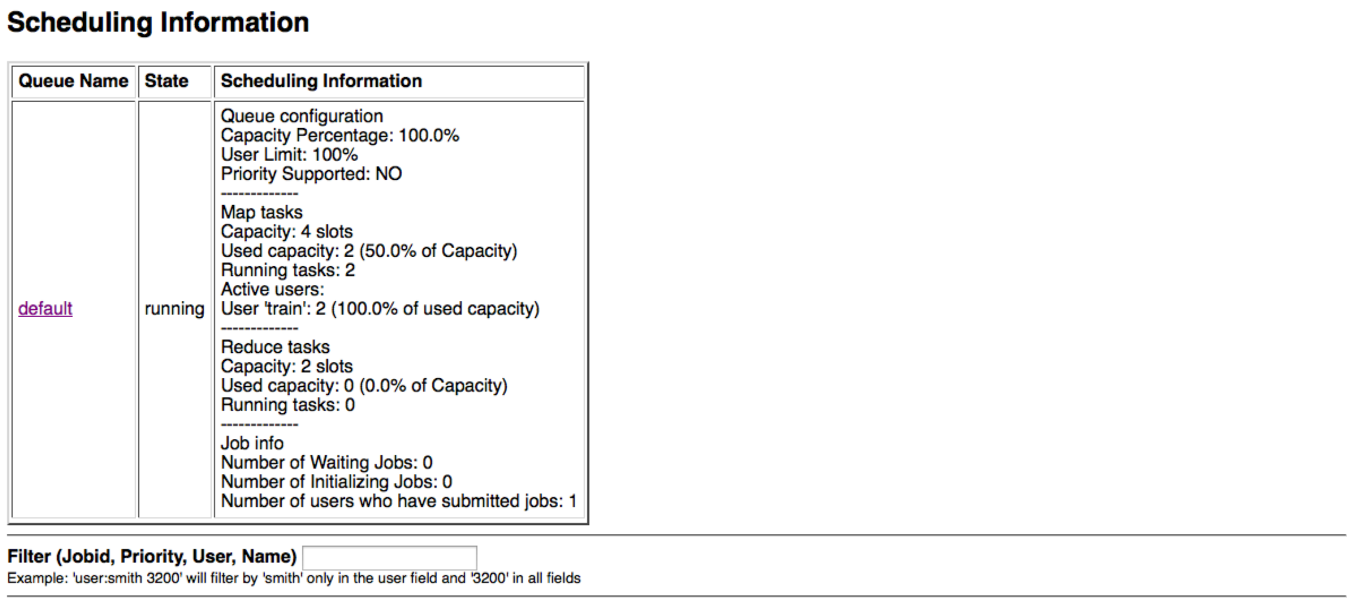
Quick Links

hortonworks-sandbox Hadoop Map/Reduce Administration

State: RUNNING
Started: Tue May 29 19:38:09 EDT 2012
Version: 1.0.2, r1302217
Compiled: Wed Mar 21 21:47:27 UTC 2012 by hrt_qa
Identifier: 201205291938

Cluster Summary (Heap Size is 185.19 MB/1004 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Graylisted Nodes	Excluded Nodes
2	0	2	1	2	0	0	0	4	2	6.00	0	0	0



Scheduling Information

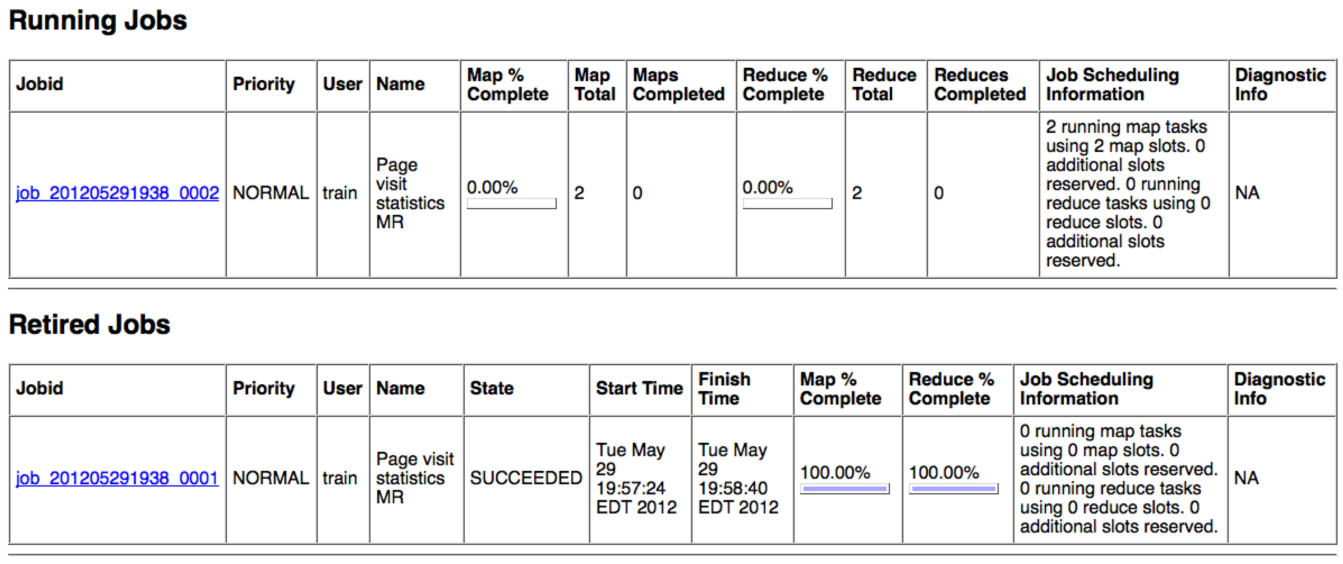
Queue Name	State	Scheduling Information
default	running	Queue configuration Capacity Percentage: 100.0% User Limit: 100% Priority Supported: NO ------------- Map tasks Capacity: 4 slots Used capacity: 2 (50.0% of Capacity) Running tasks: 2 Active users: User 'train': 2 (100.0% of used capacity) ------------- Reduce tasks Capacity: 2 slots Used capacity: 0 (0.0% of Capacity) Running tasks: 0 ------------- Job info Number of Waiting Jobs: 0 Number of Initializing Jobs: 0 Number of users who have submitted jobs: 1

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Step 6. Run your new hadoop job:

```
$ hadoop jar mroperation.jar PageStat -Dnum.reducer=2 -Dpage.stat=total
   mroperation/input mroperation/output
```

Step 7. Back in the browser, when you scroll down, you will see running jobs as follows:



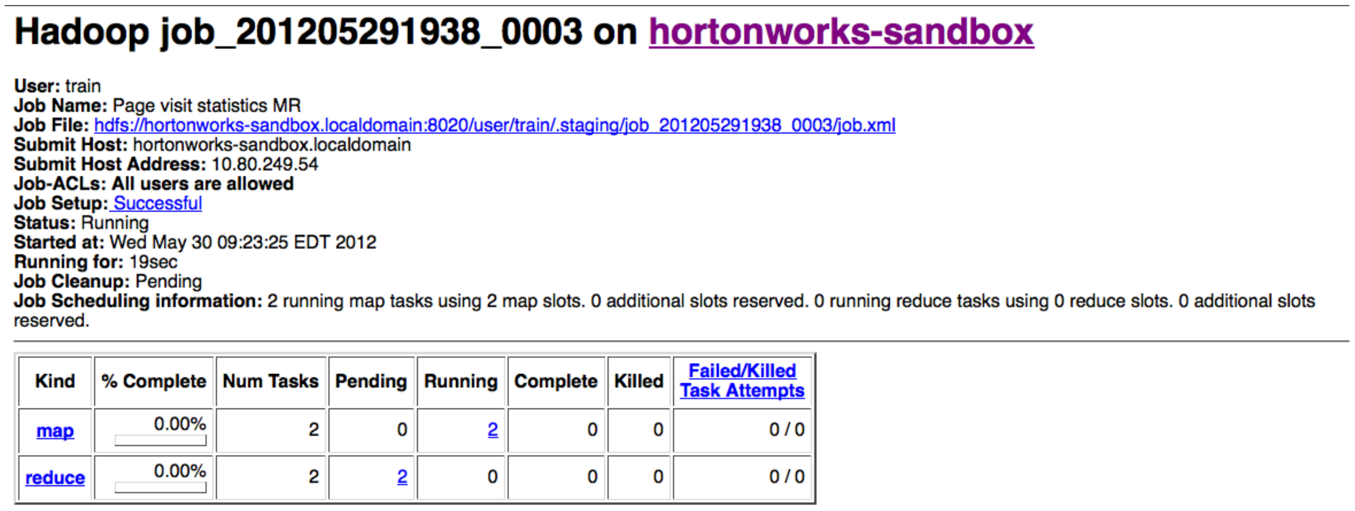
Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201205291938_0002	NORMAL	train	Page visit statistics MR	0.00%	2	0	0.00%	2	0	2 running map tasks using 2 map slots. 0 additional slots reserved. 0 running reduce tasks using 0 reduce slots. 0 additional slots reserved.	NA

Retired Jobs

Jobid	Priority	User	Name	State	Start Time	Finish Time	Map % Complete	Reduce % Complete	Job Scheduling Information	Diagnostic Info
job_201205291938_0001	NORMAL	train	Page visit statistics MR	SUCCEEDED	Tue May 29 19:57:24 EDT 2012	Tue May 29 19:58:40 EDT 2012	100.00%	100.00%	0 running map tasks using 0 map slots. 0 additional slots reserved. 0 running reduce tasks using 0 reduce slots. 0 additional slots reserved.	NA

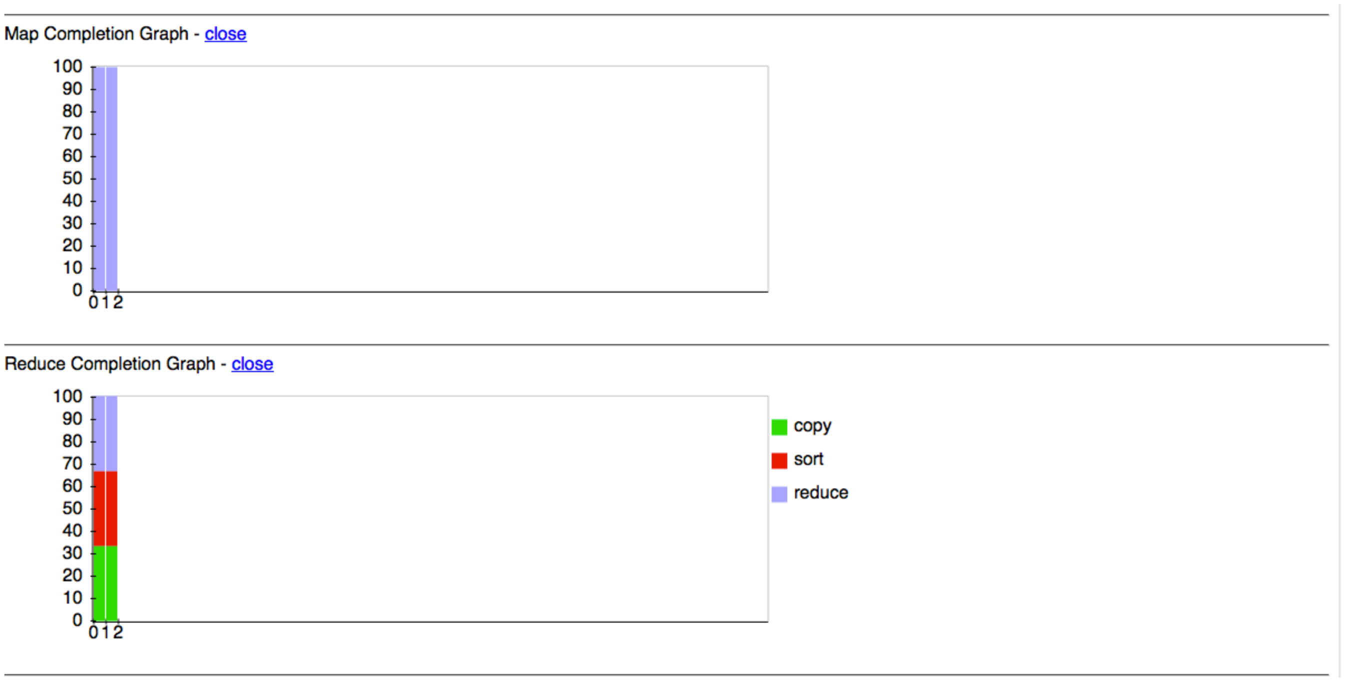
Step 8. Click on the running job. Many of the links will only work while the job is running, not after the job has ended. It will show you the map and reduce tasks as follows:



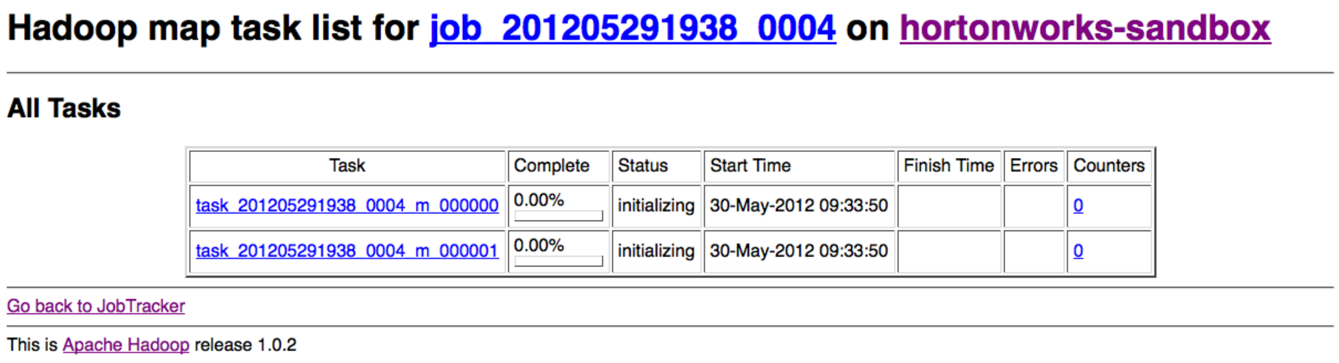
Hadoop job_201205291938_0003 on hortonworks-sandbox

User: train
Job Name: Page visit statistics MR
Job File: hdfs://hortonworks-sandbox.localdomain:8020/user/train/.staging/job_201205291938_0003/job.xml
Submit Host: hortonworks-sandbox.localdomain
Submit Host Address: 10.80.249.54
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Running
Started at: Wed May 30 09:23:25 EDT 2012
Running for: 19sec
Job Cleanup: Pending
Job Scheduling information: 2 running map tasks using 2 map slots. 0 additional slots reserved. 0 running reduce tasks using 0 reduce slots. 0 additional slots reserved.

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	0.00%	2	0	2	0	0	0 / 0
reduce	0.00%	2	2	0	0	0	0 / 0



Step 9. In the Kind category, click on the word "map". It will show you the map tasks as follows:



Hadoop map task list for job_201205291938_0004 on hortonworks-sandbox

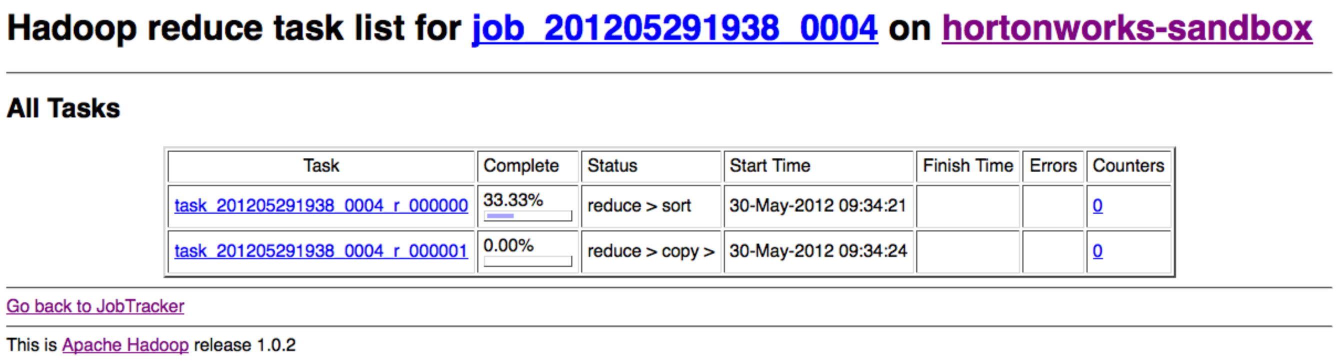
All Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
task_201205291938_0004_m_000000	0.00%	initializing	30-May-2012 09:33:50			0
task_201205291938_0004_m_000001	0.00%	initializing	30-May-2012 09:33:50			0

Go back to JobTracker

This is Apache Hadoop release 1.0.2

Step 10. Go back to the job tracker page and click on reduce. It will show you all the reduce tasks.



Hadoop reduce task list for job_201205291938_0004 on hortonworks-sandbox

All Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
task_201205291938_0004_r_000000	33.33%	reduce > sort	30-May-2012 09:34:21			0
task_201205291938_0004_r_000001	0.00%	reduce > copy >	30-May-2012 09:34:24			0

Go back to JobTracker

This is Apache Hadoop release 1.0.2

Step 11. When the job completes, check the output by running the following. You will see 2 output files, because we set the **number of reducers to 2**

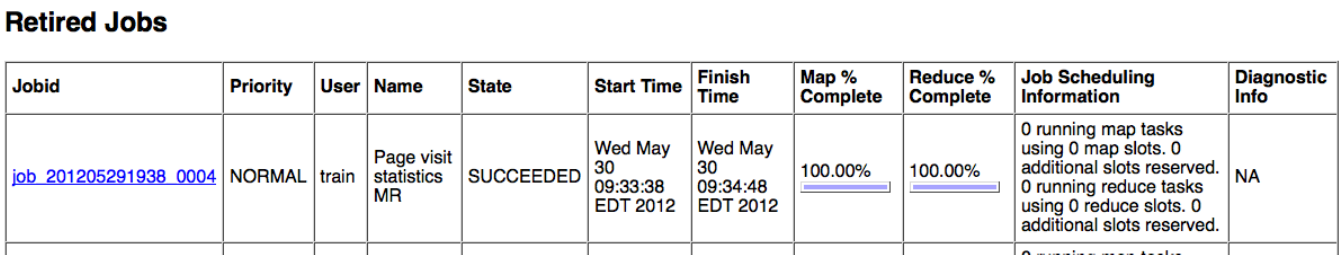
```
$ hadoop fs -ls mroperation/output
```

Step 12. Merge all output files and copy to file system

```
$ hadoop fs -getmerge mroperation/output
   ~/labs/mroperation/mergedPageStatOut.txt
```

Task 2: Deep Analysis of the Retired Job

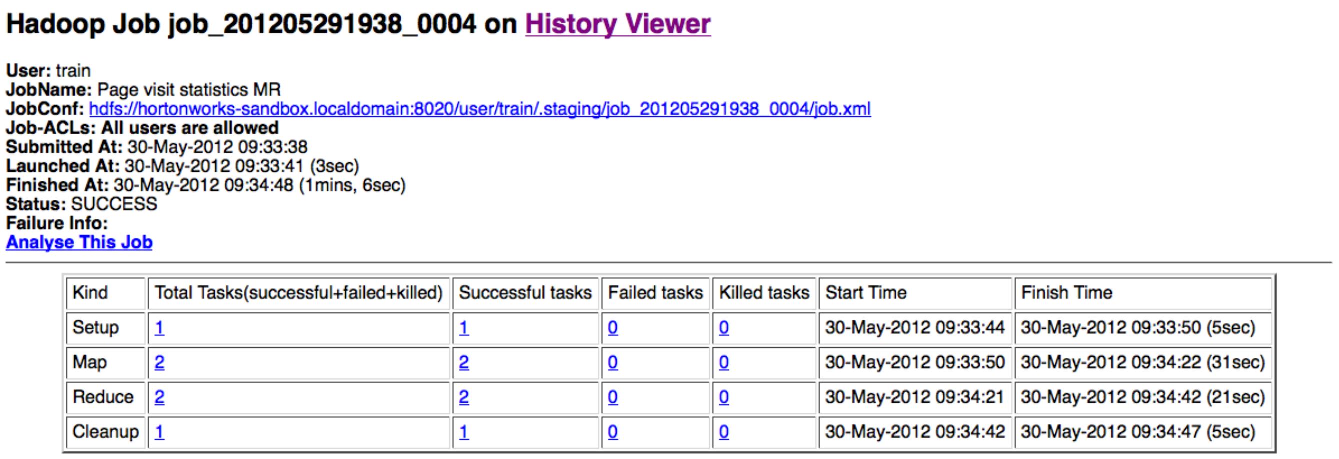
Step 1. Go back to the job tracker page and refresh the page and check if the job is completed. If completed, it will be listed under retired jobs.



Retired Jobs

Jobid	Priority	User	Name	State	Start Time	Finish Time	Map % Complete	Reduce % Complete	Job Scheduling Information	Diagnostic Info
job_201205291938_0004	NORMAL	train	Page visit statistics MR	SUCCEEDED	Wed May 30 09:33:38 EDT 2012	Wed May 30 09:34:48 EDT 2012	100.00%	100.00%	0 running map tasks using 0 map slots. 0 additional slots reserved. 0 running reduce tasks using 0 reduce slots. 0 additional slots reserved.	NA

Step 2. Click on the retired job. It shows 'History View' of the job. It also shows what kind of tasks were performed on the job.

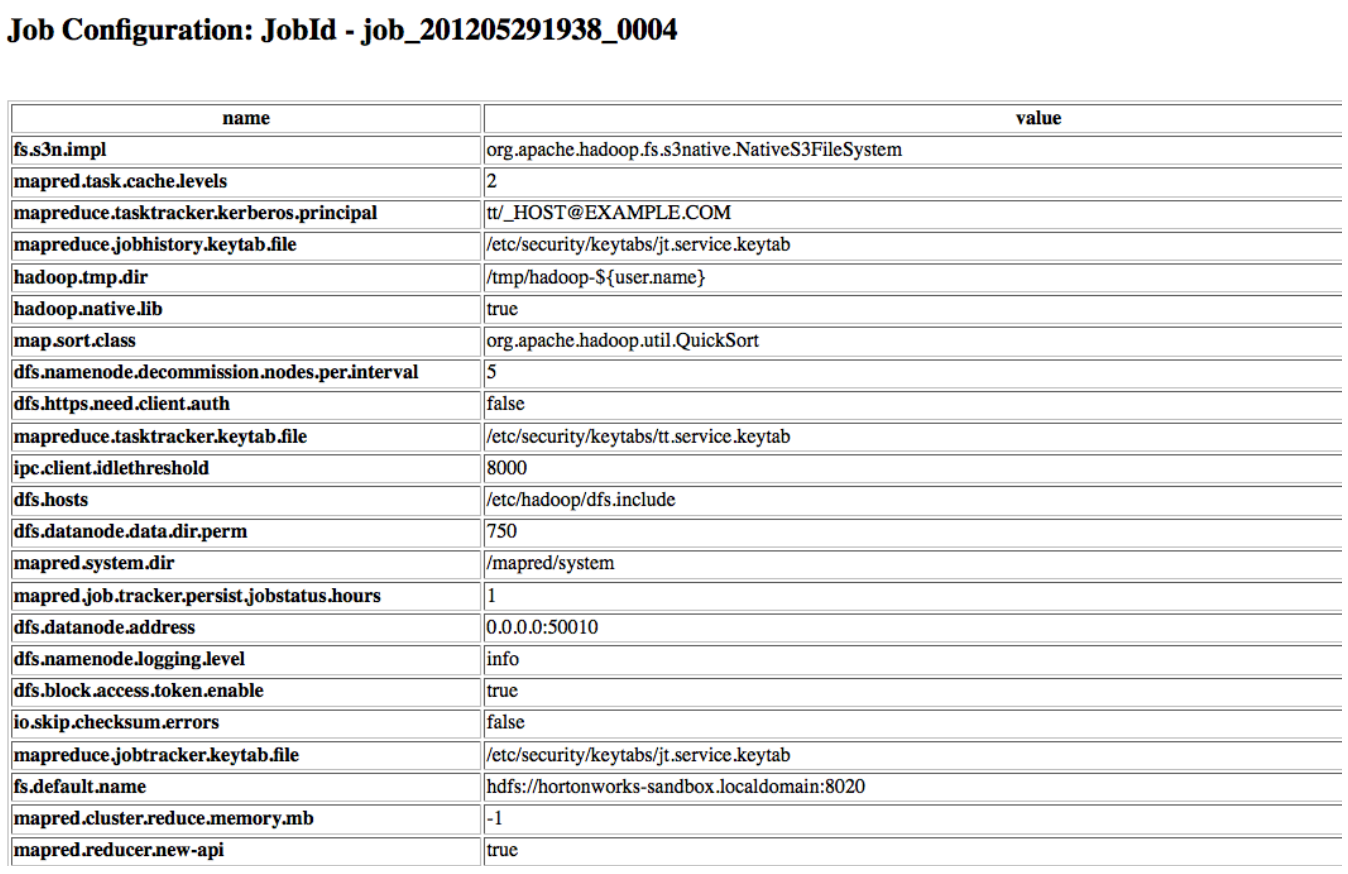


Hadoop Job job_201205291938_0004 on History Viewer

User: train
JobName: Page visit statistics MR
JobConf: hdfs://hortonworks-sandbox.localdomain:8020/user/train/.staging/job_201205291938_0004/job.xml
Job-ACLs: All users are allowed
Submitted At: 30-May-2012 09:33:38
Launched At: 30-May-2012 09:33:41 (3sec)
Finished At: 30-May-2012 09:34:48 (1mins, 6sec)
Status: SUCCESS
Failure Info:
Analyse This Job

Kind	Total Tasks(successful+failed+killed)	Successful tasks	Failed tasks	Killed tasks	Start Time	Finish Time
Setup	1	1	0	0	30-May-2012 09:33:44	30-May-2012 09:33:50 (5sec)
Map	2	2	0	0	30-May-2012 09:33:50	30-May-2012 09:34:22 (31sec)
Reduce	2	2	0	0	30-May-2012 09:34:21	30-May-2012 09:34:42 (21sec)
Cleanup	1	1	0	0	30-May-2012 09:34:42	30-May-2012 09:34:47 (5sec)

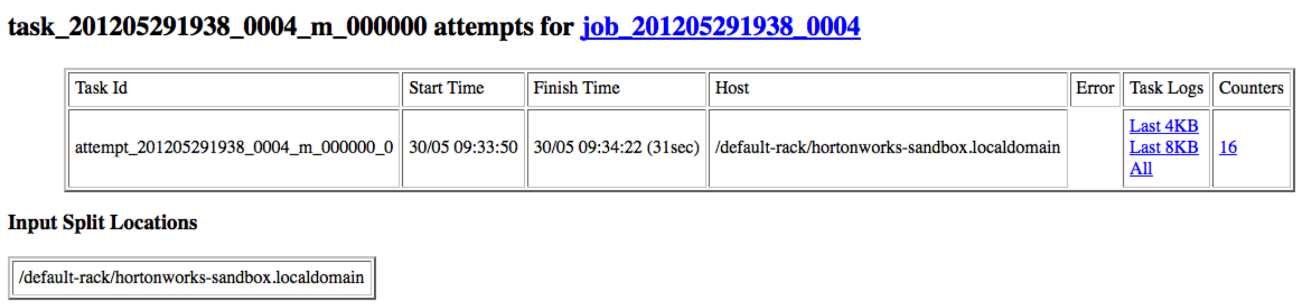
Step 3. Click on the JobConf link (The very first link on the page). It shows the complete Hadoop configuration for this job. It is another good way to learn about all Hadoop configuration variables being used for a job.



Job Configuration: JobId - job_201205291938_0004

name	value
fs.s3n.impl	org.apache.hadoop.fs.s3native.NativeS3FileSystem
mapred.task.cache.levels	2
mapreduce.tasktracker.kerberos.principal	tt/_HOST@EXAMPLE.COM
mapreduce.jobhistory.keytab.file	/etc/security/keytabs/jt.service.keytab
hadoop.tmp.dir	/tmp/hadoop-${user.name}
hadoop.native.lib	true
map.sort.class	org.apache.hadoop.util.QuickSort
dfs.namenode.decommission.nodes.per.interval	5
dfs.https.need.client.auth	false
mapreduce.tasktracker.keytab.file	/etc/security/keytabs/tt.service.keytab
ipc.client.idlethreshold	8000
dfs.hosts	/etc/hadoop/dfs.include
dfs.datanode.data.dir.perm	750
mapred.system.dir	/mapred/system
mapred.job.tracker.persist.jobstatus.hours	1
dfs.datanode.address	0.0.0.0:50010
dfs.namenode.logging.level	info
dfs.block.access.token.enable	true
io.skip.checksum.errors	false
mapreduce.jobtracker.keytab.file	/etc/security/keytabs/jt.service.keytab
fs.default.name	hdfs://hortonworks-sandbox.localdomain:8020
mapred.cluster.reduce.memory.mb	-1
mapred.reducer.new-api	true

Step 4. Now go back and click on Total task for MAP and then click on any task:



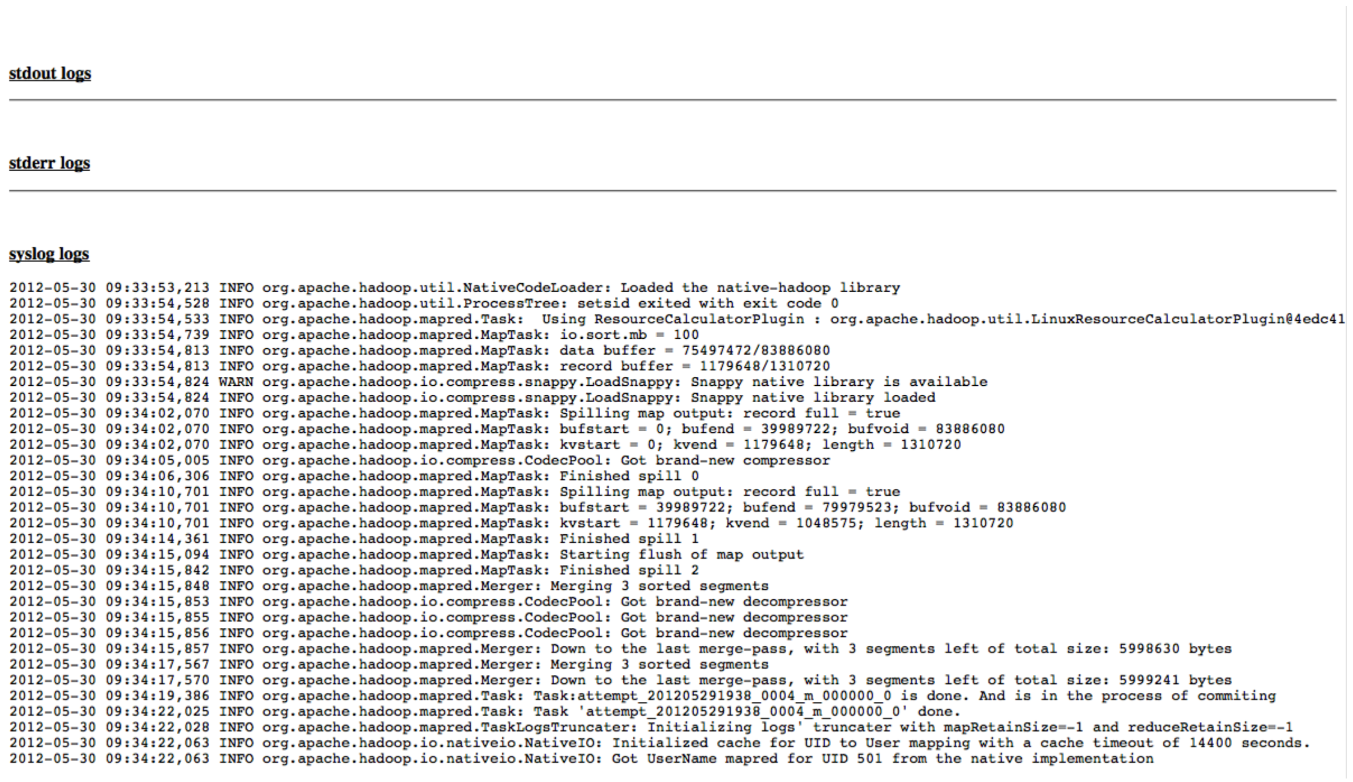
task_201205291938_0004_m_000000 attempts for job_201205291938_0004

Task Id	Start Time	Finish Time	Host	Error	Task Logs	Counters
attempt_201205291938_0004_m_000000_0	30/05 09:33:50	30/05 09:34:22 (31sec)	/default-rack/hortonworks-sandbox.localdomain		Last 4KB Last 8KB All	16

Input Split Locations

/default-rack/hortonworks-sandbox.localdomain

Step 5. In the task log column, click on ‘All’. It shows contents of logs file generated for this job:



stdout logs

stderr logs

syslog logs

```
2012-05-30 09:33:53,213 INFO org.apache.hadoop.util.NativeCodeLoader: Loaded the native-hadoop library
2012-05-30 09:33:54,528 INFO org.apache.hadoop.util.ProcessTree: setsid exited with exit code 0
2012-05-30 09:33:54,533 INFO org.apache.hadoop.mapred.Task:  Using ResourceCalculatorPlugin : org.apache.hadoop.util.LinuxResourceCalculatorPlugin@4edc41
2012-05-30 09:33:54,739 INFO org.apache.hadoop.mapred.MapTask: io.sort.mb = 100
2012-05-30 09:33:54,813 INFO org.apache.hadoop.mapred.MapTask: data buffer = 75497472/83886080
2012-05-30 09:33:54,813 INFO org.apache.hadoop.mapred.MapTask: record buffer = 1179648/1310720
2012-05-30 09:33:54,824 WARN org.apache.hadoop.io.compress.snappy.LoadSnappy: Snappy native library is available
2012-05-30 09:33:54,824 INFO org.apache.hadoop.io.compress.snappy.LoadSnappy: Snappy native library loaded
2012-05-30 09:34:02,070 INFO org.apache.hadoop.mapred.MapTask: Spilling map output: record full = true
2012-05-30 09:34:02,070 INFO org.apache.hadoop.mapred.MapTask: bufstart = 0; bufend = 39989722; bufvoid = 83886080
2012-05-30 09:34:02,070 INFO org.apache.hadoop.mapred.MapTask: kvstart = 0; kvend = 1179648; length = 1310720
2012-05-30 09:34:05,005 INFO org.apache.hadoop.io.compress.CodecPool: Got brand-new compressor
2012-05-30 09:34:06,306 INFO org.apache.hadoop.mapred.MapTask: Finished spill 0
2012-05-30 09:34:10,701 INFO org.apache.hadoop.mapred.MapTask: Spilling map output: record full = true
2012-05-30 09:34:10,701 INFO org.apache.hadoop.mapred.MapTask: bufstart = 39989722; bufend = 79979523; bufvoid = 83886080
2012-05-30 09:34:10,701 INFO org.apache.hadoop.mapred.MapTask: kvstart = 1179648; kvend = 1048575; length = 1310720
2012-05-30 09:34:14,361 INFO org.apache.hadoop.mapred.MapTask: Finished spill 1
2012-05-30 09:34:15,094 INFO org.apache.hadoop.mapred.MapTask: Starting flush of map output
2012-05-30 09:34:15,842 INFO org.apache.hadoop.mapred.MapTask: Finished spill 2
2012-05-30 09:34:15,848 INFO org.apache.hadoop.mapred.Merger: Merging 3 sorted segments
2012-05-30 09:34:15,853 INFO org.apache.hadoop.io.compress.CodecPool: Got brand-new decompressor
2012-05-30 09:34:15,855 INFO org.apache.hadoop.io.compress.CodecPool: Got brand-new decompressor
2012-05-30 09:34:15,856 INFO org.apache.hadoop.io.compress.CodecPool: Got brand-new decompressor
2012-05-30 09:34:15,857 INFO org.apache.hadoop.mapred.Merger: Down to the last merge-pass, with 3 segments left of total size: 5998630 bytes
2012-05-30 09:34:17,567 INFO org.apache.hadoop.mapred.Merger: Merging 3 sorted segments
2012-05-30 09:34:17,570 INFO org.apache.hadoop.mapred.Merger: Down to the last merge-pass, with 3 segments left of total size: 5999241 bytes
2012-05-30 09:34:19,386 INFO org.apache.hadoop.mapred.Task: Task:attempt_201205291938_0004_m_000000_0 is done. And is in the process of commiting
2012-05-30 09:34:22,025 INFO org.apache.hadoop.mapred.Task: Task 'attempt_201205291938_0004_m_000000_0' done.
2012-05-30 09:34:22,028 INFO org.apache.hadoop.mapred.TaskLogsTruncater: Initializing logs' truncater with mapRetainSize=-1 and reduceRetainSize=-1
2012-05-30 09:34:22,063 INFO org.apache.hadoop.io.nativeio.NativeIO: Initialized cache for UID to User mapping with a cache timeout of 14400 seconds.
2012-05-30 09:34:22,063 INFO org.apache.hadoop.io.nativeio.NativeIO: Got UserName mapred for UID 501 from the native implementation
```

Lab 5.2: Debug MapReduce Code

Description:

- We will copy an existing MapReduce code to local directory
- Add code to the Mapper and Reducer to produce debug output

Task 1: Open and review lab sample java file

Activity Procedure

Step 1. Change directory to where you will edit the java code:

```
$ cd ~/labs/logger
```

Step 2. Review the code in the sample file

```
$ vi MyLogger.java
```

NOTE: This file has already prewritten code from previous lab to save time. We will write only additional code, to enable logging using Log4j. ***Look for "/* TODO" comments and add code as instructed below***.

Task 2: Add code to enable logging

Activity Procedure

Step 1. First "/* TODO" item is for declaring the logger. Use following code under the comment. We are creating a new instance of Log using LogFactory's `getLog()` method by passing the class name as input.

```
private static final Log LOG =
 LogFactory.getLog(MyLogger.class);
```

Step 2. Second "/* TODO" item is for capturing an error message in standard error (stderr). Use following code under the comment. We are printing key and value for the input file in the standard error consol. We are doing it in 'map' method. So this message will show up in Mapper's log.

```
System.err.println ("M: LineOffset:<"+lineOffset+">,
 Line:<"+line+">");
```

Step 3. Third "/* TODO" item is for printing a debug message using log4j. Use following code under the comment. We are printing key and value for the input file to the log at info level. We are doing it in 'reduce' method. So this message will show up in Reducer's log

```
LOG.info("R: Line:<"+line+">, LineOffset:<"+lineOffset+">");
```

Task 3: Compile your MapReduce job

Activity Procedure

Step 1. Let's compile the code and make a jar file from the CLASS files.

```
$ javac MyLogger.java
$ jar cvf log.jar *.class
```

Step 2. Copy your input file to current directory and also on hdfs.

```
$ cp ../mrsimple/input.txt .
$hadoop fs -put input.txt logger/input.txt
```

Step 3. Run the job.

```
$hadoop jar log.jar MyLogger logger/input.txt logger/output
```

Exercise – Use JobTracker GUI To Track your Job

Step 1. Point your browser to http://<job_tracker_node>:50030 and start tracking the job progress as mentioned in the previous lab.

Step 2. You can see stderr messages printing under Map task log page. Check for **stderr logs** for your output message.

Step 3. You can see the log4j output under **Reduce Task** log page, displayed under **syslog logs**

NOTE: if any Link does not work on these pages, fix the URL by replacing servername with your EC2 cluster name.

e.g: if the URL is: `http://hortonworks-sandbox.localdomain:50030/logs` Then replace it with `http://`*`<EC2 Cluster Name>`*`:50030/logs`

Additional information:

Normally, the Hadoop JobTracker GUI is used to view the logs, since the logs are stored on whichever TaskTracker happens to run the associated map() or reduce(). To see a log for a particular job on the command prompt, follow these steps:

```
$ su -
Password: train

$ su - mapred
$ cd /hdp/disk0/data/HDP/hadoop/log_dir/mapred/userlogs
```

Look for your job based on its ID.

```
$ ls -l <JOB ID FOLDER>
e.g.
  $ ls -l job_201208271232_0009
```

You will see many sub-folders for each Map and Reduce task. Pick the right task to track your logs and check the contents in that sub-folder

```
$ cd <JOB ID FOLDER>/<MAP or Reduce Task ID Folder>
e.g.
$ cd job_201208271232_0009/attempt_201208271232_0009_r_000000_0; ls -l

[mapred@hortonworks-sandbox userlogs]$ cd
   job_201208271232_0009/attempt_201208271232_0009_r_000000_0; ls -l
total 20
-rw-r--r-- 1 mapred hadoop   154 Aug 28 18:23 log.index
-rw-r--r-- 1 mapred hadoop     0 Aug 28 18:21 stderr
-rw-r--r-- 1 mapred hadoop     0 Aug 28 18:21 stdout
-rw-r--r-- 1 mapred hadoop 12979 Aug 28 18:23 syslog
```

Lab 5.3: MapReduce Use Case: Baseball Stats

Description:

- You will be given a fairly large set of statistics (over 95,000 rows)
- All stats for each American baseball player by year 1871-2011
- You will write a MapReduce application that will determine the total number of runs for all teams during the year 1956
- You should accomplish this with minimal instruction

Task 1: Review and understand Baseball statistics data files

Activity Procedure

Step 1. Create a new directory 'baseball' in the 'labs' folder:

Step 2. Get the latest baseball stats zip file from Sean Lahman's web site (version may change):

```
$ wget 'http://seanlahman.com/files/database/lahman591-csv.zip'
```

Step 3. Unzip the file. Many statistics files will unpack from the file.

Step 4. Examine following files

a. **`Batting.csv`**

Step 5. Here is column definition for both the files

Batting.csv

playerID	Player ID code
yearID	Year
stint	player's stint (order of appearances within a season)
teamID	Team
lgID	League
G	Games
G_batting	Game as batter
AB	At Bats
R	Runs

H	Hits
2B	Doubles
3B	Triples
HR	Homeruns
RBI	Runs Batted In
SB	Stolen Bases
CS	Caught Stealing
BB	Base on Balls
SO	Strikeouts
IBB	Intentional walks
HBP	Hit by pitch
SH	Sacrifice hits
SF	Sacrifice flies
GIDP	Grounded into double plays
G_Old	Old version of games (deprecated)

Reference website

Step 6. We need to consider following field/columns

a. **Batting.csv**

i. Column # 1 (Player ID)

ii. Column # 2 (Year)

iii. Column # 9 (Runs)

Step 7. Put the file Batting.csv into Hadoop under an appropriate directory.

Task 2: Create the shell class code and test

Activity Procedure

Step 1. Your teacher may have already given you the Map/Reduce job class file (Batting.java). Open it in your favorite editor. If you do not have a sample class, create Batting.java and copy the shell code from the back of this Lab.

Step 2. Compile, jar and run the class file as-is to test the output of the default Mapper and Reducer in Hadoop. Use the input file you put into Hadoop in Task 1.

Task 3: Edit the Map code to create an intermediate file

Activity Procedure

Step 1. This is where we'll tell you very little about what to do. You have several choices; you may filter the data in the map phase, or leave it un-filtered. You may simply inverse the key value with a key of your choice, leaving all rows intact otherwise.

Step 2. Compile, jar, and run the Baseball class again. Remember to delete the output directory from your previous trial run.

Task 4: Edit the Reduce code to create the desired result

Activity Procedure

Step 1. Now you'll implement the appropriate Reduce code to aggregate data in order to produce all runs for the target year. If you filtered data in your map phase, you may end up with only one row. If you did not, you can scroll through the result set to find the appropriate data row with the target year's result.

Step 2. If you achieve the result before anyone else does, claim your prize!

Task 5: Optional: trending analysis

Activity Procedure

Step 1. Can you notice something very interesting about runs in recent years, versus a decade ago?

Begin with this class (OR copy ~/labs/mrsimple/MyFirstMR.java OR WordCount.java to the baseball directory and modify it):

```
//Standard Java imports
import java.util.Iterator;
import java.io.IOException;

//Standard Hadoop imports
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.mapred.lib.*;

public class Batting  {
           public static class MyMap extends MapReduceBase implements
                                   Mapper<LongWritable, Text, LongWritable, Text>{

                   private Text word = new Text();

                   public void map(LongWritable key, Text value,
                                   OutputCollector<LongWritable, Text> output, Reporter
    reporter) throws IOException {
                   }
```

```
        }

        public static class MyReduce extends MapReduceBase implements
                                Reducer<LongWritable, Text, LongWritable, LongWritable> {

                public void reduce(LongWritable key, Iterator<Text> values,
    OutputCollector<LongWritable, LongWritable> output, Reporter reporter) throws IOException
    {               }
        }

        public static void main(String[] args) throws IOException {
                //Code to create a new Job specifying the MapReduce class
                final JobConf conf = new JobConf(Batting.class);

                //call different set methods to set the configuration
                conf.setInputFormat(TextInputFormat.class);
                conf.setOutputFormat(TextOutputFormat.class);

                //add appropriate custom mappers/reducers here

                //conf.setOutputKeyClass(Text.class);
                //conf.setOutputValueClass(IntWritable.class);

                //File Input/Output argument passed as a command line argument

                FileInputFormat.setInputPaths(conf, new Path(args[0]));
                FileOutputFormat.setOutputPath(conf, new Path(args[1]));

                //statement to execute the job
                JobClient.runJob(conf);
        }
}
```

Hints:

- There's a sweet little Java String method called `split(delimiter)` that might save you some effort.
- Be careful if you decide to start changing the signature of the Mapper or Reducer interfaces in the class. You'll have to make that change in several places.
- Check for unexpected missing data in some rows. Also, remember there's a header row.

Lab 6.1: Introduction to Pig

Description:

- Upload data to HDFS
- Load to specify input in Pig
- Dump
- Store into HDFS
- Describe
- Load directory and multiple files to Pig
- Create Pig Script

Task 1: Necessary File Location for Pig Lab

Activity Procedure

Step 1. Examine data files in ~/labs/pigintro

Task 2: Upload Data to hdfs

Activity Procedure

Step 1. Go to the directory

```
$cd ~/labs/pigintro
```

Step 2. In shell type the following create statement in HDFS system

```
$hadoop fs -mkdir pigintro/input
```

Step 3. put the file into HDFS

```
$hadoop fs -put ~/labs/pigintro/sample.data pigintro/input
```

Activity Verification

Check files in hdfs

```
$ hadoop fs -ls pigintro/input/
```

Task 3: Connecting to Pig Shell

Activity Procedure

Step 1. Start the Pig shell by typing the following at the command prompt.

```
$ pig
```

On successful completion you should see the following output:

```
grunt>
```

NOTE: To exit from Pig Shell, use the following command (but do not enter it now):

```
$ quit;
```

Task 4: Load Data

Activity Procedure

Step 1. In the grunt shell type the following statement which will load sample data as "A".

```
grunt> A = LOAD 'pigintro/input/sample.data' using PigStorage(',') AS
    (gender:chararray, age:int, income:int, zip:chararray);
grunt > describe A;
```

Activity Verification

- On completion you should see the following output,

```
A: {gender: chararray,age: int,income: int,zip: chararray}
```

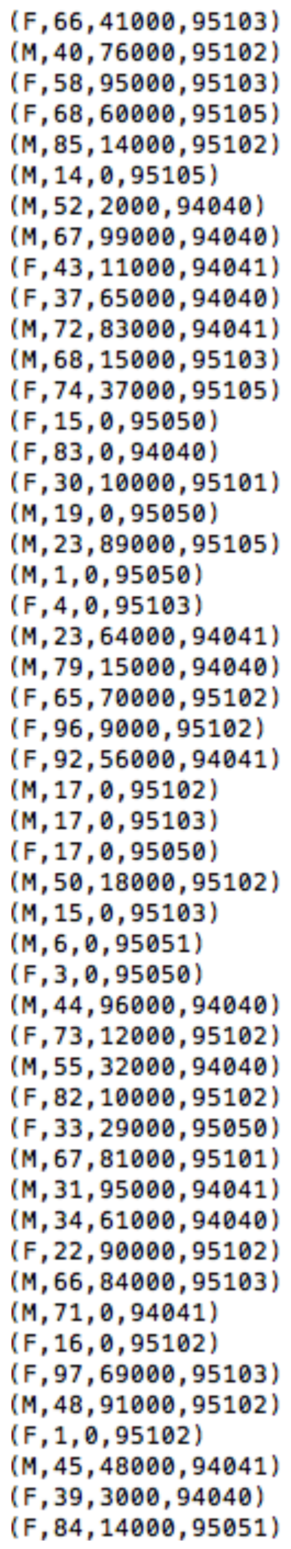
Task 5: Dump Data

Step 1. View the data of A:

```
grunt > DUMP A;
```

Activity Verification

- You will see Pig spinning up a MapReduce job. Wait a few seconds as it will take some time.
- You will see results as follows:



```
(F,66,41000,95103)
(M,40,76000,95102)
(F,58,95000,95103)
(F,68,60000,95105)
(M,85,14000,95102)
(M,14,0,95105)
(M,52,2000,94040)
(M,67,99000,94040)
(F,43,11000,94041)
(F,37,65000,94040)
(M,72,83000,94041)
(M,68,15000,95103)
(F,74,37000,95105)
(F,15,0,95050)
(F,83,0,94040)
(F,30,10000,95101)
(M,19,0,95050)
(M,23,89000,95105)
(M,1,0,95050)
(F,4,0,95103)
(M,23,64000,94041)
(M,79,15000,94040)
(F,65,70000,95102)
(F,96,9000,95102)
(F,92,56000,94041)
(M,17,0,95102)
(M,17,0,95103)
(F,17,0,95050)
(M,50,18000,95102)
(M,15,0,95103)
(M,6,0,95051)
(F,3,0,95050)
(M,44,96000,94040)
(F,73,12000,95102)
(M,55,32000,94040)
(F,82,10000,95102)
(F,33,29000,95050)
(M,67,81000,95101)
(M,31,95000,94041)
(M,34,61000,94040)
(F,22,90000,95102)
(M,66,84000,95103)
(M,71,0,94041)
(F,16,0,95102)
(F,97,69000,95103)
(M,48,91000,95102)
(F,1,0,95102)
(M,45,48000,94041)
(F,39,3000,94040)
(F,84,14000,95051)
```

Task 6: Store data in HDFS

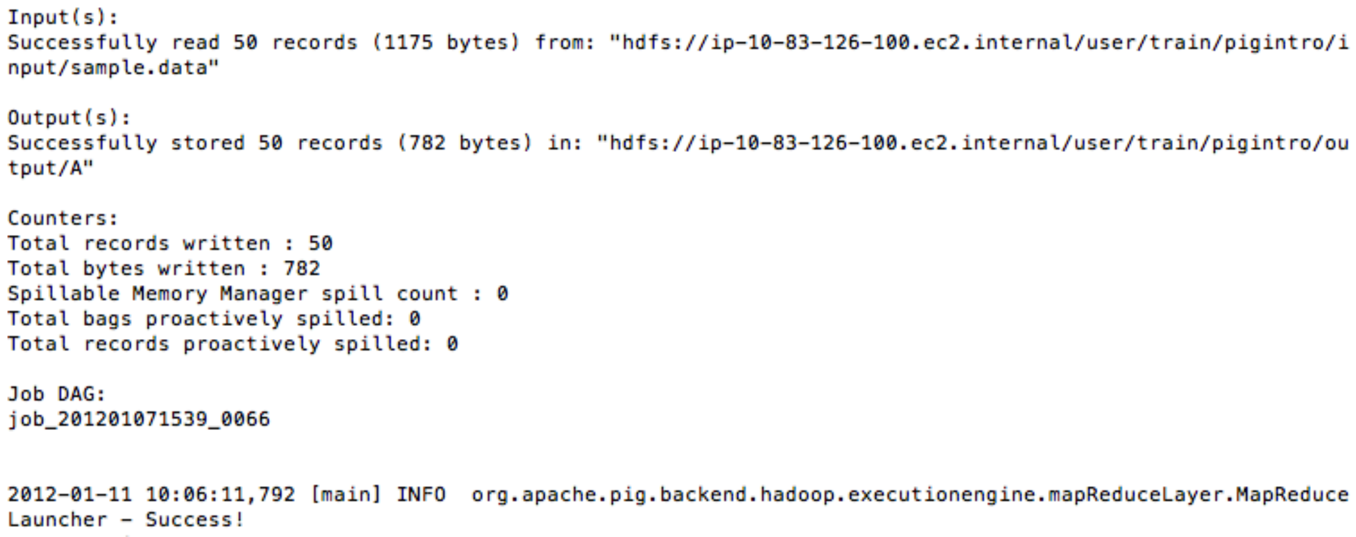
Activity Procedure

Step 1. In the grunt shell type the following

```
grunt >   STORE A INTO 'pigintro/output/A';
```

Activity Verification

Again you will see Pig running MapReduce job. On successful completion you should see the following output:



```
Input(s):
Successfully read 50 records (1175 bytes) from: "hdfs://ip-10-83-126-100.ec2.internal/user/train/pigintro/i
nput/sample.data"

Output(s):
Successfully stored 50 records (782 bytes) in: "hdfs://ip-10-83-126-100.ec2.internal/user/train/pigintro/ou
tput/A"

Counters:
Total records written : 50
Total bytes written : 782
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_201201071539_0066


2012-01-11 10:06:11,792 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduce
Launcher - Success!
```

Step 2. Exit grunt.

```
grunt> quit
```

Task 7: Query newly created table

Activity Procedure

Step 1. Lets examine the saved data in HDFS in shell:

```
$ hadoop fs -ls pigintro/output
$ hadoop fs -ls pigintro/output/A
```

Activity Verification

- On successful completion you should see the following output,

```
-rw-------   3 train supergroup        782 2012-01-13 17:44
/user/train/pigintro/output/A/part-m-00000
```

- The file 'part-m-00000' will contain our records contained in A. You can view the content using the following statement.

```
$ hadoop fs -cat pigintro/output/A/*
```

Task 8: Loading a directory or multiple files

Activity Procedure

Step 1. In shell, use the following statement to insert into hdfs file system.

```
$ hadoop fs -put *.data pigintro/input
```

Note: Since you put a file in here before you will receive an error message

Step 2. Let's load a directory into hdfs. In pig shell, use the following statement to load.

- `grunt> pig;`

```
grunt> A = LOAD 'pigintro/input';
```

- This loads the directory

Note that we will get schema unknown at this point because have not defined it

```
grunt> DESCRIBE A;
```

You will see Schema Unknown message.

Step 3. Now let's examine A

```
grunt > DUMP A;
```

Task 9: Creating Pig Script

Activity Procedure

Step 1. Create a file named myfirstpig.pig in ~/labs/pigintro at the shell prompt:

```
$ vi myfirstpig.pig
```

Step 2. Enter the following commands:

```
A = LOAD 'pigintro/input/sample.data' using PigStorage(',');
DUMP A;
```

Step 3. Let's run the file

```
$pig myfirstpig.pig
```

Activity Verification

You will see a map reduce job is running with results similar to the following:

```
(F,22,90000,95102)
(M,66,84000,95103)
(M,71,0,94041)
(F,16,0,95102)
```

```
(F,97,69000,95103)
(M,48,91000,95102)
(F,1,0,95102)
(M,45,48000,94041)
(F,39,3000,94040)
(F,84,14000,95051)
```

Lab 6.2: Pig Data Operations

Description:

In this lab, we will be going over various ETL Features of Pig to do the following operations.

- Join
- Filter
- Foreach
- Explain
- Illustrate

Task 1: Necessary File Location for Pig Lab

Activity Procedure

Step 1. If zip-city.data is not copied into your HDFS yet, put the file into HDFS now:

```
$ hadoop fs -put ~/labs/pigintro/zip-city.data pigintro/input
```

Task 2: Connect to Pig Shell

Activity Procedure

Step 1. Start the Pig shell by typing the following at the command prompt.

```
$ pig
```

Task 3: FILTER in Pig

The FILTER keyword is typically used to eliminate data from a dataset.

Activity Procedure

Step 1. In the Pig shell type the following statement to load data into A.

```
grunt > A = LOAD 'pigintro/input/sample.data' USING PigStorage(',') AS
    (gender:chararray, age:int, income:int, zip:chararray);
```

Step 2. Use the following statement, which will report the data on your screen.

```
grunt >  DUMP A;
```

Step 3. Let’s filter the dataset in A by zipcode = ‘95102’

```
grunt > B = FILTER A BY zip == '95102';
```

Step 4. Now let's dump filtered data from step 3 on your screen.

```
grunt >  DUMP B;
```

You will see that the result set will be a filtered set of only Zip = 95102.

Step 5. To add more filters to the above dataset, defined the following relation:

```
grunt > B = FILTER A BY (zip == '95102' AND gender == 'M');
```

Step 6. Let's verify:

```
grunt >  DUMP B;
```

You should see that the data is filtered now by zip = '95102' and gender = 'M' as follows:

```
(M,40,76000,95102)
(M,85,14000,95102)
(M,17,0,95102)
(M,50,18000,95102)
(M,48,91000,95102)
```

Task 4: STORE in Pig

Step 1. Now store the output to a HDFS file

```
grunt >  STORE B INTO 'pigintro/output2' USING PigStorage(',');
grunt > fs -cat pigintro/output2/part-m-00000
M,40,76000,95102
M,85,14000,95102
M,17,0,95102
M,50,18000,95102
M,48,91000,95102
```

Task 5: FOREACH in Pig

Step 1. Use FOREACH to create a new dataset

```
grunt> C = FOREACH A GENERATE income, zip;
grunt> DUMP C;
```

Notice the results are only the income and zip fields:

```
(95000,94041)
(61000,94040)
(90000,95102)
(84000,95103)
(3000,94040)
(14000,95051)
```

Step 2. Add some more operation while using FOREACH. This will overwrite content of the previous C.

```
grunt> C = FOREACH A GENERATE income/1000, zip;
```

Step 3. Verify data by using the following statement.

```
grunt> DUMP C;
```

You will see results as follows:

```
(95,94041)
(61,94040)
(90,95102)
(84,95103)
(3,94040)
(14,95051)
```

Task 6: FOREACH with FILTER

Activity Procedure

Step 1. In the grunt shell type the following commands.

```
grunt > C = FOREACH A GENERATE income/1000 AS income_k, zip;
grunt > D = FILTER C BY income_k > 0;
grunt> DUMP D;
```

On successful completion you should see output similar to:

```
(84,95103)
(69,95103)
(91,95102)
(48,94041)
```

```
(3,94040)
(14,95051)
```

Task 7: JOIN in Pig

Activity Procedure

Step 1. In the grunt shell type the following commands.

```
grunt> E = LOAD 'pigintro/input/zip-city.data' USING PigStorage(',') AS
(zip:chararray, city:chararray);
grunt> jnd = JOIN A by zip, E by zip;
grunt> DUMP jnd;
```

Activity Verification

- On successful completion, you should see output similar to the following:

```
(M,15,0,95103,95103,san jose)
(F,97,69000,95103,95103,san jose)
(M,17,0,95103,95103,san jose)
(F,58,95000,95103,95103,san jose)
(F,4,0,95103,95103,san jose)
(F,74,37000,95105,95105,san jose)
(M,14,0,95105,95105,san jose)
(F,68,60000,95105,95105,san jose)
(M,23,89000,95105,95105,san jose)
```

Task 8: ILLUSTRATE in Pig

ILLUSTRATE will execute your script with a subset of your data. This can be useful for quickly verifying the operation of your script. Sometimes when the subset of data selected is eliminated as your script runs, a different dataset will be synthesized.

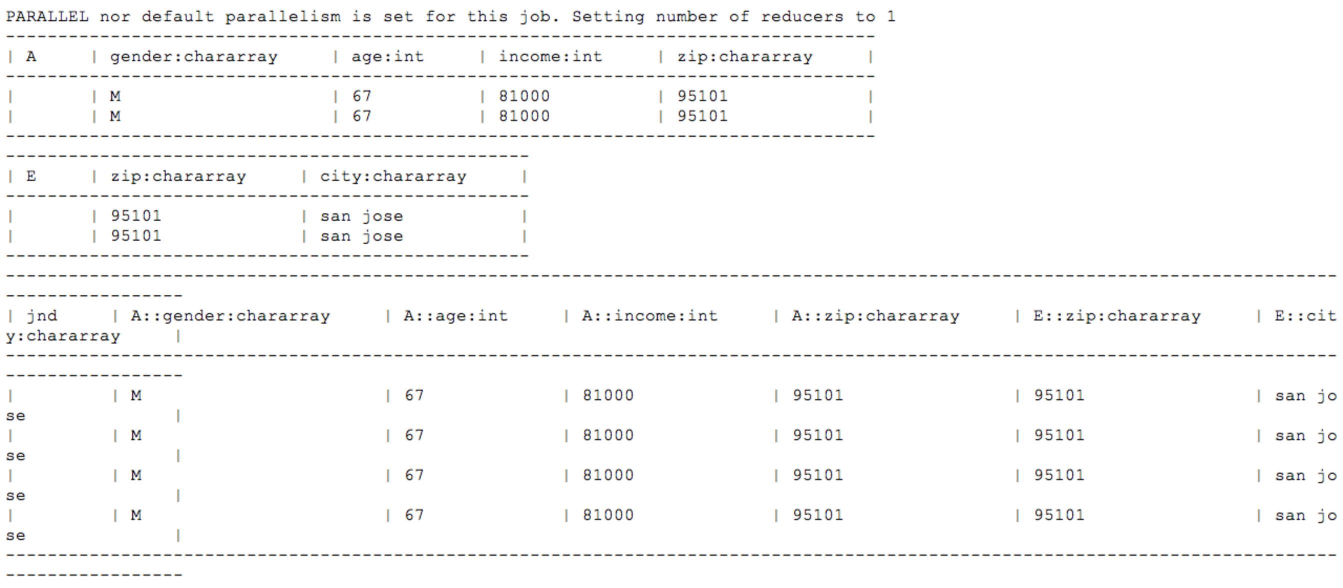
Activity Procedure

Step 1. In the grunt shell type the following commands.

```
grunt> ILLUSTRATE jnd;
```

Activity Verification

- On successful completion, you should see illustration of sample data on the screen as follows. Your results will be similar but not identical to below.



```
PARALLEL nor default parallelism is set for this job. Setting number of reducers to 1
-------------------------------------------------------------------------------------
| A      | gender:chararray      | age:int      | income:int      | zip:chararray      |
-------------------------------------------------------------------------------------
|        | M                     | 67           | 81000           | 95101              |
|        | M                     | 67           | 81000           | 95101              |
-------------------------------------------------------------------------------------
---------------------------------------------------
| E      | zip:chararray      | city:chararray      |
---------------------------------------------------
|        | 95101              | san jose            |
|        | 95101              | san jose            |
---------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
-----------------
| jnd      | A::gender:chararray      | A::age:int      | A::income:int      | A::zip:chararray      | E::zip:chararray      | E::cit
y:chararray      |
------------------------------------------------------------------------------------------------------------------------------------
-----------------
|          | M                        | 67              | 81000              | 95101                 | 95101                 | san jo
se               |
|          | M                        | 67              | 81000              | 95101                 | 95101                 | san jo
se               |
|          | M                        | 67              | 81000              | 95101                 | 95101                 | san jo
se               |
|          | M                        | 67              | 81000              | 95101                 | 95101                 | san jo
se               |
------------------------------------------------------------------------------------------------------------------------------------
-----------------
```

Task 7: EXPLAIN in Pig

An execution plan is used to process your Pig script. The EXPLAIN keyword is used to display the execution plan. This can be useful considering alternative scripts to do the same work.

Activity Procedure

Step 1. Enter the following command:

```
grunt> EXPLAIN jnd;
```

Step 2. The output will display the Logical Plan, the Physical Plan, and the Map Reduce Plan for computing the jnd relation.

Lab 6.3: Pig Latin

Description:

In this lab, we will be going over various data operations as follows:

- Use of GROUP and COGROUP
- FOREACH with average
- Flattening of Data Structure using FLATTEN

Task 1: Using GROUP in PIG

This operation groups data with same key. Let's calculate population per zip code.

Activity Procedure

Complete these steps:

Step 1. Start the Pig shell by typing the following at the command prompt.

```
$ pig
grunt> A = LOAD 'pigintro/input/sample.data' USING PigStorage(',') AS
    (gender:chararray, age:int, income:int, zip:chararray);
grunt> B = GROUP A BY zip;
grunt> DUMP B;
```

Activity Verification

- On successful completion, you should see the data grouped by zip:

```
(94040,{(F,39,3000,94040),(M,79,15000,94040),(M,52,2000,94040),(M,67,99000,9
4040),(M,44,96000,94040),(F,37,65000,94040),(M,34,61000,94040),(M,55,32000,9
4040),(F,83,0,94040)})
(94041,{(F,92,56000,94041),(M,71,0,94041),(M,23,64000,94041),(F,43,11000,940
41),(M,31,95000,94041),(M,72,83000,94041),(M,45,48000,94041)})
(95050,{(F,15,0,95050),(F,17,0,95050),(F,3,0,95050),(M,19,0,95050),(M,1,0,95
050),(F,33,29000,95050)})
(95051,{(M,6,0,95051),(F,84,14000,95051)})
(95101,{(M,67,81000,95101),(F,30,10000,95101)})
(95102,{(M,17,0,95102),(M,40,76000,95102),(M,85,14000,95102),(F,65,70000,951
02),(F,96,9000,95102),(M,50,18000,95102),(F,73,12000,95102),(F,82,10000,9510
2),(F,22,90000,95102),(F,16,0,95102),(M,48,91000,95102),(F,1,0,95102)})
(95103,{(M,68,15000,95103),(F,66,41000,95103),(M,66,84000,95103),(M,15,0,951
03),(F,97,69000,95103),(M,17,0,95103),(F,58,95000,95103),(F,4,0,95103)})
```

```
(95105,{(F,74,37000,95105),(M,14,0,95105),(F,68,60000,95105),(M,23,89000,951
05)})
```

Task 2: Using COUNT along with GROUP in PIG

Step 1. Let's count the records per zip by doing following:

```
grunt> C = FOREACH B GENERATE group, COUNT(A);
grunt> DESCRIBE C;

C: {group: chararray,long}
```

NOTE: Notice the output does not have column-names, only data types.

Step 2. Recreate this dataset with proper column names:

```
grunt> C = FOREACH B GENERATE group AS zip:chararray, COUNT(A) AS
   total:int;
grunt> DESCRIBE C;

C: {zip: chararray,total: int}

grunt> DUMP C;
```

Activity Verification

- On successful completion, you should see the following result:

```
(94040,9)
(94041,7)
(95050,6)
(95051,2)
(95101,2)
(95102,12)
(95103,8)
(95105,4)
```

Step 3. Counting all the records:

```
grunt> B = GROUP A  ALL;
grunt> DESCRIBE B;
B: {group: chararray,A: {(gender: chararray,age: int,income: int,zip:
   chararray)}}

grunt> DUMP B;
```

```
(all,{(F,66,41000,95103),(M,40,76000,95102),(F,58,95000,95103),(F,68,60000,9
5105),(M,85,14000,95102),(M,14,0,95105),(M,52,2000,94040),(M,67,99000,94040)
,(F,43,11000,94041),(F,37,65000,94040),(M,72,83000,94041),(M,68,15000,95103)
,(F,74,37000,95105),(F,15,0,95050),(F,83,0,94040),(F,30,10000,95101),(M,19,0
,95050),(M,23,89000,95105),(M,1,0,95050),(F,4,0,95103),(M,23,64000,94041),(M
,79,15000,94040),(F,65,70000,95102),(F,96,9000,95102),(F,92,56000,94041),(M,
17,0,95102),(M,17,0,95103),(F,17,0,95050),(M,50,18000,95102),(M,15,0,95103),
(M,6,0,95051),(F,3,0,95050),(M,44,96000,94040),(F,73,12000,95102),(M,55,3200
0,94040),(F,82,10000,95102),(F,33,29000,95050),(M,67,81000,95101),(M,31,9500
0,94041),(M,34,61000,94040),(F,22,90000,95102),(M,66,84000,95103),(M,71,0,94
041),(F,16,0,95102),(F,97,69000,95103),(M,48,91000,95102),(F,1,0,95102),(M,4
5,48000,94041),(F,39,3000,94040),(F,84,14000,95051)})
```

```
grunt> C = FOREACH B GENERATE group, COUNT(A);
grunt> DUMP C;

2012-08-29 16:54:47,631 [main] INFO
org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total
input paths to process : 1
(all,50)
```

Task 3: Using AVERAGE along with GROUP

Calculate average income for zipcode

Activity Procedure

Complete these steps:

Step 1. Execute following steps:

```
grunt> B = GROUP A BY zip;
grunt> C = FOREACH B GENERATE group, AVG(A.income);
grunt> DUMP C;
```

Activity Verification

- The output will show the average income by zipcode:

```
(94040,41444.444444444445)
(94041,51000.0)
(95050,4833.333333333333)
(95051,7000.0)
(95101,45500.0)
(95102,32500.0)
(95103,38000.0)
(95105,46500.0)
```

Task 4: COGROUP

COGROUP joins multiple datasets on a common key

Activity Procedure

Step 1. Define a COGROUP relation:

```
grunt> B = LOAD 'pigintro/input/zip-city.data' USING PigStorage(',') AS
    (zip:chararray, city:chararray);
grunt> C = COGROUP A BY zip, B BY zip;
grunt> DESCRIBE C;
grunt> DUMP C;
```

Activity Verification

The output will look like:

```
(94040,{(F,39,3000,94040),(M,79,15000,94040),(M,52,2000,94040),(M,67,99000,9
4040),(M,44,96000,94040),(F,37,65000,94040),(M,34,61000,94040),(M,55,32000,9
4040),(F,83,0,94040)},{(94040,mountain view)})

(94041,{(F,92,56000,94041),(M,71,0,94041),(M,23,64000,94041),(F,43,11000,940
41),(M,31,95000,94041),(M,72,83000,94041),(M,45,48000,94041)},{(94041,mounta
in view)})

(95050,{(F,15,0,95050),(F,17,0,95050),(F,3,0,95050),(M,19,0,95050),(M,1,0,95
050),(F,33,29000,95050)},{(95050,santa clara)})
(95051,{(M,6,0,95051),(F,84,14000,95051)},{})
```

```
(95101,{(M,67,81000,95101),(F,30,10000,95101)},{(95101,san jose)})

(95102,{(M,17,0,95102),(M,40,76000,95102),(M,85,14000,95102),(F,65,70000,951
02),(F,96,9000,95102),(M,50,18000,95102),(F,73,12000,95102),(F,82,10000,9510
2),(F,22,90000,95102),(F,16,0,95102),(M,48,91000,95102),(F,1,0,95102)},{(951
02,san jose)})

(95103,{(M,68,15000,95103),(F,66,41000,95103),(M,66,84000,95103),(M,15,0,951
03),(F,97,69000,95103),(M,17,0,95103),(F,58,95000,95103),(F,4,0,95103)},{(95
103,san jose)})

(95105,{(F,74,37000,95105),(M,14,0,95105),(F,68,60000,95105),(M,23,89000,951
05)},{(95105,san jose)})
```

NOTE: Compare the output above with the output from the JOIN command earlier:

```
(M,15,0,95103,95103,san jose)
(F,97,69000,95103,95103,san jose)
(M,17,0,95103,95103,san jose)
(F,58,95000,95103,95103,san jose)
(F,4,0,95103,95103,san jose)
(F,74,37000,95105,95105,san jose)
(M,14,0,95105,95105,san jose)
(F,68,60000,95105,95105,san jose)
(M,23,89000,95105,95105,san jose)
```

Task 5: FLATTEN

Use FLATTEN keyword to ‘flatten’ bags.

Activity Procedure

Step 1. Execute the following to flatten the output:

```
grunt> D = FOREACH C GENERATE FLATTEN(A), FLATTEN(B);
grunt> DESCRIBE D;
grunt> DUMP D;
```

- Examine the output

Activity Verification

Notice all bags are flattened in D, creating a relation identical to the JOIN relation:

```
(M,15,0,95103,95103,san jose)
(F,97,69000,95103,95103,san jose)
(M,17,0,95103,95103,san jose)
(F,58,95000,95103,95103,san jose)
(F,4,0,95103,95103,san jose)
(F,74,37000,95105,95105,san jose)
(M,14,0,95105,95105,san jose)
(F,68,60000,95105,95105,san jose)
(M,23,89000,95105,95105,san jose)
```

Lab 6.4: Pig Use Case: Baseball Stats

Description:

- We will reuse data from previous Baseball challenge lab
- We will identify players who scored highest runs for each year in ascending order
- We will also determine First and Last name for the each player by joining 2 data sets

Task 1: Review and understand Baseball statistics data files

Activity Procedure

Step 1. Examine following files

a. **`Batting.csv`**

b. **`Master.csv`**

- Here is column definition for both the files

`Batting.csv`

playerID	Player ID code
yearID	Year
stint	player's stint (order of appearances within a season)
teamID	Team
lgID	League
G	Games
G_batting	Game as batter
AB	At Bats
R	Runs
H	Hits
2B	Doubles
3B	Triples
HR	Homeruns
RBI	Runs Batted In
SB	Stolen Bases
CS	Caught Stealing
BB	Base on Balls

SO	Strikeouts
IBB	Intentional walks
HBP	Hit by pitch
SH	Sacrifice hits
SF	Sacrifice flies
GIDP	Grounded into double plays
G_Old	Old version of games (deprecated)

Master.csv

lahmanID	Unique number assigned to each player
playerID links	A unique code asssigned to each player. The playerID the data in this file with records in the other files.
managerID	An ID for individuals who served as managers
hofID	An ID for individuals who are in teh baseball Hall of Fame
birthYear	Year player was born
birthMonth	Month player was born
birthDay	Day player was born
birthCountry	Country where player was born
birthState	State where player was born
birthCity	City where player was born
deathYear	Year player died
deathMonth	Month player died
deathDay	Day player died
deathCountry	Country where player died
deathState	State where player died
deathCity	City where player died
nameFirst	Player's first name
nameLast	Player's last name
nameNote changed	Note about player's name (usually signifying that they their name or played under two differnt names)
nameGiven	Player's given name (typically first and middle)
nameNick	Player's nickname
weight	Player's weight in pounds
height	Player's height in inches

bats	Player's batting hand (left, right, or both)
throws	Player's throwing hand (left or right)
debut	Date that player made first major league appearance
finalGame	Date that player made first major league appearance (blank if still active)
college	College attended
lahman40ID	ID used in Lahman Database version 4.0
lahman45ID	ID used in Lahman database version 4.5
retroID	ID used by retrosheet
holtzID	ID used by Sean Holtz's Baseball Almanac
bbrefID	ID used by Baseball Reference website

- We need to consider following field/columns

c. Batting.csv

i. Column # 1 (Player ID)

ii. Column # 2 (Year)

iii. Column # 9 (Runs)

d.Master.csv

i. Column # 2 (Player ID)

ii. Column # 17 (First Name)

iii. Column # 18 (Last Name)

Task 2: Identify players who scored highest runs for each year

Activity Procedure

Step 1. Create a new directory baseball/input in HDFS and copy batting-data file Batting.csv & master-data files Master.csv to this directory from the local machine

Step 2. Start the Pig shell

Step 3. Load batting data to Pig using Pig Storage ‘,’:

NOTE: A CSV file had data comma-separated in each line. So, we need to inform 'PIG' explicitly about its field delimiter. The default delimiter in PIG is a TAB (\t).

Step 4. Read relevant fields from the loaded data. In this case we are interested in 1st , 2nd and 9th fields for each record. Use foreach and generate command to accomplish this task:

`NOTE: $0 represents 1st field, $1 represents 2nd field and so on..`

Step 5. Group runs from step-4 by year

Step 6. Use foreach, generate, group and MAX function on Step 5 data to get max run for each year

Step 7. Join Step 4 and Step 6 data based on the 'year' and 'runs' fields

Step 8. To identify the playerID who scored the highest one for each year, create a new dataset having Year, PlayerID and Max Run data using foreach, generate on Step7 data:

Step 9. Check the output of the above exercise using dump command on Step 8:

Task 3: Determine First and Last name for the each player

Activity Procedure

Step 1. Load master data to Pig using pig storage ',':

Step 2. Read relevant fields from the file. In this case we are interested in 2nd, 17th & 18th fields for each record using foreach and generate command:

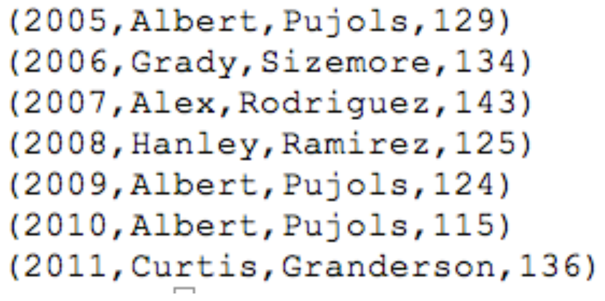
Step 3. Join PLAYERS dataset with the result dataset from previous task (Step 9) based the common field 'playerID':

Step 4. Create a new dataset having Year, Player's First and Last Name and the Max from Step 3 using foreach and generate commands.

Step 5. Make sure that data in Step 4 is sorted on Year in ascending order

Step 6. Get the output Step 5 using dump command:

Step 7. Following is the expected output:



```
(2005,Albert,Pujols,129)
(2006,Grady,Sizemore,134)
(2007,Alex,Rodriguez,143)
(2008,Hanley,Ramirez,125)
(2009,Albert,Pujols,124)
(2010,Albert,Pujols,115)
(2011,Curtis,Granderson,136)
```

Lab 7.1: Intro to Hive

Description:

- Create a Table
- Load a Small Data File
- Do a query on the Table
- Create a second Table
- Populate second Table with Small Data File
- Perform a join of these two Tables

Task 1: Necessary File Location for Hive Lab

Activity Procedure

Step 1. Examine data files in ~/labs/hiveintro/

Task 2: Connecting to Hive Shell

Activity Procedure

Step 1. Start the Hive shell by typing the following at the command prompt.

```
$ hive
hive>
```

Task 3: Create a table in Hive

Activity Procedure

Step 1. In the Hive shell type the following create statement.

```
hive>  create table orders(orderid bigint, customerid bigint,  productid
    int, qty int, rate int, estdlvdate string, status string) row format
    delimited fields terminated by ",";
    OK
    Time taken:  0.744 seconds
```

- Let's view this table using describe command.

```
hive>  desc orders;
OK
orderid bigint
customerid      bigint
```

```
productid       int
qty     int
rate    int
estdlvdate      string
status  string
Time taken: 0.199 seconds
```

Task 4: Load data into the newly created table

Activity Procedure

Step 1. In the Hive shell type the following command.

```
hive>  load data local inpath '/home/train/labs/hiveintro/orders.txt'
   into table orders;
```

Activity Verification

- On successful completion you should see the following output,

```
Copying data from file: /home/train/labs/hiveintro/orders.txt
Copying file: file: /home/train/labs/hiveintro/orders.txt
Loading data to table default.orders
OK
Time taken: 0.747 seconds
```

Task 5: Query newly created table

Activity Procedure

Step 1. In the Hive shell type the following command.

```
hive>  select * from orders;
```

Activity Verification

- On successful completion you should see the following output,

```
…
9999995 605472     4     1     797   2011-10-10 D
9999996 987212     1     3     321   2011-10-10 N
9999997 855217     7     3     576   2011-10-10 N
9999998 466491     6     4     415   2011-10-10 N
9999999 864735     1     5     126   2011-10-10 D
```

```
10000000     134363     7     3     794   2011-10-10 D
Time taken: 0.645 seconds
```

Task 7: Create second Table

Activity Procedure

Step 1. In the Hive shell type the following command.

```
hive>  create table products (productid int, description string) row
   format delimited fields terminated by ",";
```

Activity Verification

- On successful completion you should see the following output,

```
OK
Time taken:  0.081 seconds
```

Task 8: Load data to second Table

Activity Procedure

Step 1. In the Hive shell type the following to load data into the products table.

```
hive>  load data local inpath '/home/train/labs/hiveintro/products.txt'
   into table products;
```

Activity Verification

- On successful completion you should see messages like this,

```
Copying data from file:/home/train/labs/hiveintro/products.txt
Copying file: file:/home/train/labs/hiveintro/products.txt
Loading data to table default.products
OK
Time taken: 0.33 seconds
```

Task 9: Table Join

Activity Procedure

Step 1. In the Hive shell type the following to join the orders and products table.

```
hive>  select orderid, customerid, a.productid, qty, rate,  status,
   description from orders a join products b where
   a.productid=b.productid;
```

Activity Verification

- On successful completion you should see messages like this,

```
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201201071539_0133, Tracking URL = http://ip-10-83-126-
100.ec2.internal:50030/jobdetails.jsp?jobid=job_201201071539_0133
Kill Command = /usr/libexec/../bin/hadoop job  -Dmapred.job.tracker=ip-
10-83-126-100.ec2.internal:9000 -kill job_201201071539_0133
Hadoop job information for Stage-1: number of mappers: 2; number of
reducers: 1
2012-01-11 15:43:21,121 Stage-1 map = 0%,  reduce = 0%
2012-01-11 15:43:27,514 Stage-1 map = 50%,  reduce = 0%
2012-01-11 15:43:29,542 Stage-1 map = 100%,  reduce = 0%
2012-01-11 15:43:38,783 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201201071539_0133
OK
9999001 671548    2    1    833  D    templates
9999003 451625    3    3    407  N    stencils
9999004 387416    2    4    191  N    templates
9999005 224773    7    3    24   D    pencils
9999006 151558    6    1    591  N    baloons
9999007 863456    5    3    603  N    streamers
9999008 906602    1    1    319  D    widgets
9999009 24454 7   5    540  N    pencils
9999010 964182    9    3    229  N    notebooks
9999012 834127    1    1    415  N    widgets
9999013 559904    8    4    37   N    erasers
9999014 9002  1   5    177  D    widgets
9999015 663425    4    3    42   N    graphs
9999016 784338    1    1    825  N    widgets

Time taken: 39.073 seconds
```

Step 2. Exit from hive shell.

```
hive>  quit;
```

Lab 7.2: Hive External Tables & Partitions

Task 1: Load data file into hdfs and create an external table

Hive External Tables is a mechanism by which you can impose a Hive table structure on data that already exists in stores like HDFS or HBase. In this example we will create a Hive External table on a file that we have uploaded into HDFS.

Description:

- Create an external table by pointing it to the orders data file that we will upload into hdfs into a separate directory.

Activity Procedure

Step 1. In command prompt, type the following command to load the file into hdfs.

```
$ hadoop fs -mkdir hiveext/input
$ hadoop fs -put /home/train/labs/hiveext/orders.txt hiveext/input
```

- Launch hive shell

```
$hive
```

- Create the Hive external table.

```
hive> create external table ext_orders (orderid bigint, customerid
   bigint, productid int, qty int, rate int, estdlvdate string, status
   string) row format delimited fields terminated by "," stored as
   textfile location '/user/train/hiveext/input';
```

- You can view the content of the table as follows:

```
hive> desc ext_orders;
OK
orderid bigint
customerid      bigint
productid       int
qty     int
rate    int
estdlvdate      string
status  string
Time taken: 0.418 seconds
```

```
hive> select * from ext_orders;
…
9999983 544238      3     4     912   2011-10-10 D
9999984 280447      6     3     430   2011-10-10 N
9999985 763457      9     4     767   2011-10-10 D
9999986 158398      10    3     272   2011-10-10 N
9999987 1404  10    2     843   2011-10-10 N
9999988 996481      2     2     282   2011-10-10 N
9999989 310726      5     1     846   2011-10-10 N
9999990 617938      9     1     456   2011-10-10 N
9999991 227182      7     1     385   2011-10-10 N
9999992 994834      5     5     975   2011-10-10 N
9999993 894586      3     1     61    2011-10-10 N
9999994 730603      8     3     100   2011-10-10 N
9999995 605472      4     1     797   2011-10-10 D
9999996 987212      1     3     321   2011-10-10 N
9999997 855217      7     3     576   2011-10-10 N
9999998 466491      6     4     415   2011-10-10 N
9999999 864735      1     5     126   2011-10-10 D
10000000     134363      7     3     794   2011-10-10 D
```

Note: In the HDFS system, if there are multiple files in hiveext/input directory, then all of those will be part of ext_orders table.

Task 2: Create partitioned in Hive

This section builds upon the work done in the previous lab to showcase one of the most powerful features of Hive which is **Dynamic Partitioning**.

Partitioning allows you to categorize and store data based on selected fields. One of the advantages of this approach is that selected partitions can be manipulated without affecting other partitions. For instance you can overwrite the contents of a selected partition that affects only that partition without changing the contents of the whole table.

Description:

- Create a new table called that is partitioned on a field.
- Populate this table with the contents on the existing orders table.

Activity Procedure

Step 1. The default configuration parameters must be changed.

With these properties setup, you don't need to specify the partition, i.e. (status = 'D') in the query.

```
hive>  set hive.exec.dynamic.partition=true;
hive>  set hive.exec.dynamic.partition.mode=nonstrict;
```

- In the Hive shell type the following create statement.

```
hive>  create table p_orders (orderid bigint, customerid bigint,
   productid int, qty int, rate int, estdlvdate string) partitioned by
   (status string);
OK
Time taken:  0.065 sec
```

- Import data into the p_orders table from the orders table.

```
hive>   insert overwrite table p_orders partition (status) select
   orderid, customerid, productid, qty, rate, estdlvdate, status from
   ext_orders;
```

Activity Verification

```
Total MapReduce jobs = 2
Launching Job 1 out of 2
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201111191423_0004, Tracking URL =
   http://xxx:50030/jobdetails.jsp?jobid=job_201111191423_0004
Kill Command = /Users/hadoop/hadoop-0.20.2-cdh3u1/bin/hadoop job  -
   Dmapred.job.tracker=localhost:9001 -kill job_201111191423_0004
……..
…….
Ended Job = job_201111191423_0004
Ended Job = -1121683093, job is filtered out (removed at runtime).
Moving data to: hdfs://xxx:9000/tmp/hive-hadoop/hive_2011-11-23_14-07-
   34_426_3958285142272253581/-ext-10000
Loading data to table default.p_orders partition (status=null)
Deleted hdfs://xxx:9000/user/hive/warehouse/p_orders/status=D
Deleted hdfs://localhost:9000/user/hive/warehouse/p_orders/status=N
   Loading partition {status=D}
   Loading partition {status=N}
[Warning] could not update stats.
3000 Rows loaded to p_orders
OK
Time taken: 28.271 seconds
```

Let's see the partitions.

```
hive> show partitions p_orders;
OK
status=D
status=N
Time taken: 0.075 seconds
```

Step 2. Exit from hive shell.

```
hive>  quit;
```

Lab 7.3: Hive Features - User Defined Function (UDF)

This section shows how to create a User Defined Function in Hive. The function we will create will take the value of the estdlvdate column of the Orders table as input and return only the year portion of the date back.

Description:

- Create a UDF called GetYear()
- Perform a query on the `orders` table using this function

Task 1: Create the Java UDF called GetYear

Activity Procedure

Step 1. Compile UDF java file GetYear.java

```
$ cd ~/labs/hiveudf/
$ javac GetYear.java
```

- Create a jar file

```
$ jar cvf GetYear.jar *.class
```

Task 2: Use UDF function in hive

Activity Procedure

Step 1. Load this function into the Hive distributed cache and create an alias for the Java function.

```
$ hive
hive> add jar /home/train/labs/hiveudf/GetYear.jar;
hive> create temporary function GetYear as 'GetYear';
OK
Time taken: 0.373 seconds
```

- Let's use the UDF in a select statement as follows.

```
hive> select orderid, customerid, productid, qty, rate,
   GetYear(estdlvdate), status from orders where status='D';
```

Activity Verified

On successful completion, you should see the following output. Notice that only the year portion of the date is returned for the estdlvdate column.

...

```
9999985 763457     9    4    767   2011  D
```

```
9999995 605472     4    1    797   2011  D
9999999 864735     1    5    126   2011  D
10000000     134363     7    3    794   2011  D
Time taken: 26.943 seconds
```

Lab 7.4: Using Hive to Read Pig Data

Description:

In this lab, we will be going over various data operations as follows:

- Load test data to Pig
- Create External Table on the PIG Data
- Read Pig data on Hive terminal

Task 1: Load test data to Pig

Activity Procedure

Complete these steps:

Step 1. Create the input directory structure

```
$ cd /home/train/labs/pig2hive
```

```
$ hadoop fs -mkdir pig2hive/input
```

- Copy the source data into the input directory

```
$ hadoop fs -put pig2hive.data pig2hive/input
```

- Enter Pig

```
$ pig
```

- Load the test file and save its data in a variable in a variable.

```
grunt> A = LOAD 'pig2hive/input/pig2hive.data' USING PigStorage('\t') AS
    (gender:chararray, age:int, income:int, zip:chararray);
```

- Now store data in A into a pig directory

```
grunt>  STORE A INTO 'pig2hive/output' USING PigStorage('\t');
```

- Check the output from PIG prompt.

```
grunt > fs -ls pig2hive/output
```

```
  •  Found 1 items
-rw-rw-rw-   1 train hdfs        782 2012-08-29 18:50
   /user/train/pig2hive/output/part-m-00000
```

- Quit from PIG prompt.

```
grunt > quit;
```

Task 2: Create an external table on Pig Data

Step 1. Start hive shell:

```
$ hive
```

- Create an External table on Pig data using following command:

```
hive> create external table hive_zip_data (gender String, age int, income
    int, zip int) row format delimited fields terminated by '\t' lines
    terminated by '\n' stored as textFile location
    '/user/train/pig2hive/output';
```

Note: This is the same schema as was used with Pig earlier in the lab.

- Verify data in the new table 'hive_zip_data'

```
hive> select * from hive_zip_data;
```

Activity Verification

- On successful completion, you should see the following result on the screen as follows.

```
...
M       67      81000   95101
M       31      95000   94041
M       34      61000   94040
F       22      90000   95102
M       66      84000   95103
M       71      0       94041
F       16      0       95102
F       97      69000   95103
M       48      91000   95102
F       1       0       95102
M       45      48000   94041
F       39      3000    94040
F       84      14000   95051
```

Lab 7.5: Hive Use Case: Baseball Stats Exercise: Analyze Baseball Statistics data using Hive

Description:

- Let's achieve the same result using Hive this time
- We will identify players who scored highest runs for each year in ascending order
- We will also determine First and Last name for the each player by joining 2 data sets

Task 1: Identify players who scored highest runs for each year

Activity Procedure

Step 1. Change to the baseball directory:

Step 2. Start the Hive shell by typing following

Step 3. Create a temporary table 'temp_batting' having only one column 'col_value' (STRING type) to store batting data into hive:

Step 4. Insert data into the table from the local CSV files using LOAD DATA LOCAL INPATH command.

Step 5. Create another table 'batting' to store only relevant columns (player_id, year, runs) for batting into hive:

Step 6. Read relevant fields from temp_batting table. In this case we are interested in 1st , 2nd and 9th fields for each record :

```
hive> INSERT OVERWRITE TABLE batting
 SELECT
        regexp_extract(col_value, '^(?:([^,]*)\,?){1}', 1) player_id,
        regexp_extract(col_value, '^(?:([^,]*)\,?){2}', 1) year,
        regexp_extract(col_value, '^(?:([^,]*)\,?){9}', 1) run
FROM temp_batting;
```

NOTE: We are using built-in String function '**regexp_extract(str, regular expression, index)**' to identify the right column. You can get more information on this function at following location: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF

Step 7. Write a select query to group data by year and run so that we could identify the highest score for each year

Step 8. Write a select query to join above dataset with batting table to identify the player_id who scored the highest one for each year and create a new query having year, player_id and Max Run data:

Task 2: Determine First and Last name for the each player

Activity Procedure

Step 1. Create a temporary tables ‘temp_master’ to store master data into hive:

Step 2. Insert data into this table from the local CSV files:

Step 3. Create another table ‘master’ to store only relevant columns(player_id, fname, lname) for batting into hive:

Step 4. Read relevant fields from temp_master table. In this case we are interested in 2nd, 17th & 18th fields for each record:

```
hive> INSERT OVERWRITE TABLE master
  SELECT
        regexp_extract(col_value, '^(?:([^,]*)\,?){2}', 1),
        regexp_extract(col_value, '^(?:([^,]*)\,?){17}', 1),
        regexp_extract(col_value, '^(?:([^,]*)\,?){18}', 1)
FROM temp_master;
```

Step 5. Join the final query from previous task with master table to get First and Last name of the player and make sure data is sorted based on year in ascending order

Task 3: Save the output from previous task to a new table

Activity Procedure

Step 1. Create a new table final_result using output of final query from previous task.

Step 2. Check the data of the final_result table to verify its contents.

Step 3. The output should look like:

```
1983    Tim     Raines  133
1984    Dwight  Evans   121
1985    Rickey  Henderson       146
1986    Rickey  Henderson       130
1987    Tim     Raines  123
1988    Wade    Boggs   128
1989    Wade    Boggs   113
1990    Rickey  Henderson       119
1991    Paul    Molitor 133
1992    Tony    Phillips        114
1993    Lenny   Dykstra 143
1994    Frank   Thomas  106
1995    Craig   Biggio  123
1996    Ellis   Burks   142
1997    Craig   Biggio  146
1998    Sammy   Sosa    134
1999    Jeff    Bagwell 143
2000    Jeff    Bagwell 152
2001    Sammy   Sosa    146
2002    Alfonso Soriano 128
2003    Albert  Pujols  137
2004    Albert  Pujols  133
2005    Albert  Pujols  129
2006    Grady   Sizemore        134
2007    Alex    Rodriguez       143
2008    Hanley  Ramirez 125
2009    Albert  Pujols  124
2010    Albert  Pujols  115
2011    Curtis  Granderson      136
Time taken: 0.282 seconds
```

Lab 8.1: HCatalog with Pig and Hive

Description:

In this lab, we will be going over various data operations as follows:

- Load test data to Pig
- Create an External Table using Hive using RCFile format
- Load test data into a variable in Pig
- Store this data in hive table using HCatalog
- Read Pig data on Hive terminal

Task 1: Create Hive table to reference external file

Activity Procedure

Complete these steps:

Step 1. Change directories

```
$ cd /home/train/labs/hcatalog
```

- Create a directory to store your files

```
$ hadoop fs –mkdir hcatalog/input
```

- Store the files in the directory

```
$ hadoop fs –put /home/train/labs/hcatalog/hcatalog.data hcatalog/input
```

- View the contents of the file **`my_data.hcatalog`**. It creates a new table named my_data. To execute this script, enter the following command:

```
$ hcat -f my_data.hcatalog
```

- To verify the table was created successfully, enter the following describe command and verify you have a similar output:

```
$ hcat -e 'describe my_data'

OK
gender  string
age     int
income  int
zip     int
Time taken: 0.918 seconds
```

Task 2: Check existence of external table in HDFS

Step 1. Confirm that table exists in HDFS:

```
$ hadoop fs -ls hcatalog/output
```

```
Found 1 items

drwxrwxrwx   - hcat hdfs          0 2012-09-14 17:57
     /user/train/hcatalog/output/my_data
```

Task 3: Load data into Hive using Pig

Step 1. Load data from a file into Pig

```
$ pig
$ A = LOAD 'hcatalog/input/hcatalog.data' USING PigStorage('\t') AS
   (gender:chararray, age:int, income:int, zip:int);
```

- Check that data loaded successfully:

```
grunt> dump A;
You will see data in a format similar to:
```

...

```
(M,48,91000,95102)
(F,1,0,95102)
(M,45,48000,94041)
(F,39,3000,94040)
(F,84,14000,95051)
```

- Now store A into the hive table 'my_data' using HCatalog.

```
grunt> STORE A into 'my_data' using org.apache.hcatalog.pig.HCatStorer();
```

You will see a map/reduce job run successfully.

```
Counters:
Total records written : 50
Total bytes written : 811
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_201201112105_0011
```

```
2012-01-12 12:30:31,947 [main] INFO
   org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduc
   eLauncher - Success!
```

- Now data is loaded in the Hive table. So it is time to check/verify data from Hive, which is loaded from Pig.

```
grunt> quit;
```

- Start Hive:

```
$ hive
```

- Verify data in the table 'my_data'

```
hive> select * from my_data;
```

Activity Verification

- On successful completion, you should see output similar to:

```
...
M       34      61000   94040
F       22      90000   95102
M       66      84000   95103
M       71      0       94041
F       16      0       95102
F       97      69000   95103
M       48      91000   95102
F       1       0       95102
M       45      48000   94041
F       39      3000    94040
F       84      14000   95051
```

Lab 9.1: HBase Basics

Description:

- Using HBase Shell to get status
- Create table
- Add some test data into table

Task 1: Exploring HBase shell

Activity Procedure

Complete these steps:

Step 1. Execute the following to enter the hbase shell

```
$ hbase shell
```

- Verify you are in hbase shell, with no errors. You will see something like the following:

```
[train@/home/train]$ hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.92.0, r1232557, Fri Mar  2 19:30:44 UTC 2012

hbase(main):001:0>
```

NOTE: To **exit** from hbase shell, use the exit command (do not enter it now):

```
hbase(main):001:0> exit
```

- In the hbase shell type

```
hbase(main) > status
```

You will see output like:

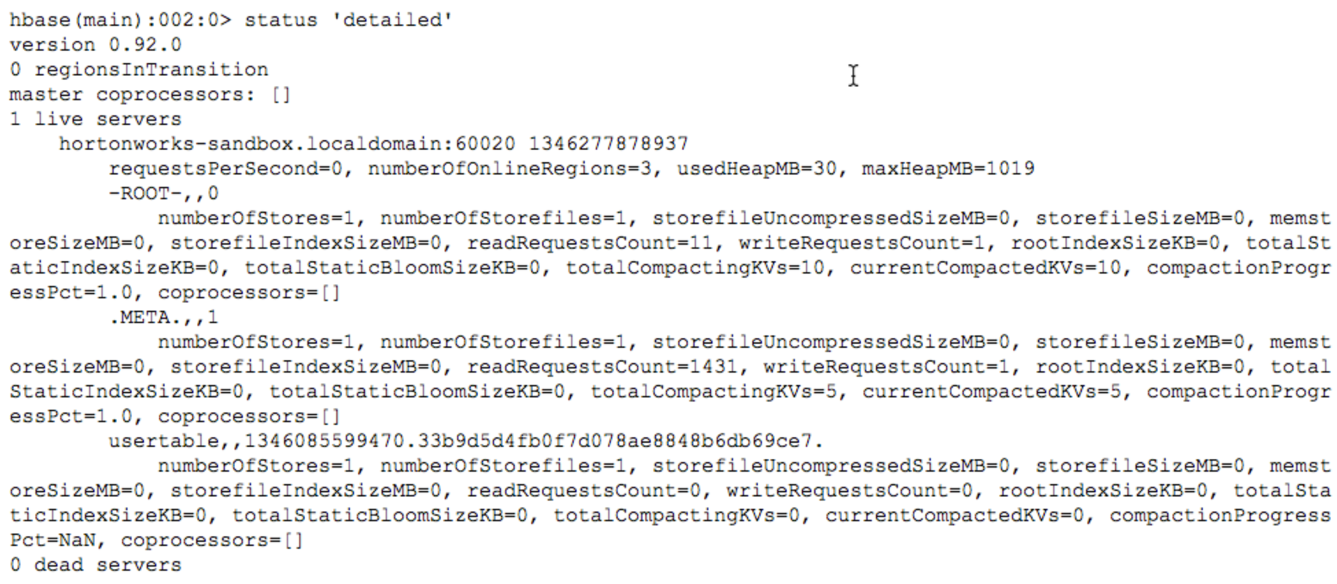
```
1 servers, 0 dead, 3.0000 average load
```

- To get more details on hbase status, use this statement as follows.

```
hbase(main):002:0> status 'detailed'
```

Activity Verification

You should see output similar to:



```
hbase(main):002:0> status 'detailed'
version 0.92.0
0 regionsInTransition
master coprocessors: []
1 live servers
    hortonworks-sandbox.localdomain:60020 1346277878937
        requestsPerSecond=0, numberOfOnlineRegions=3, usedHeapMB=30, maxHeapMB=1019
        -ROOT-,,0
            numberOfStores=1, numberOfStorefiles=1, storefileUncompressedSizeMB=0, storefileSizeMB=0, memst
oreSizeMB=0, storefileIndexSizeMB=0, readRequestsCount=11, writeRequestsCount=1, rootIndexSizeKB=0, totalSt
aticIndexSizeKB=0, totalStaticBloomSizeKB=0, totalCompactingKVs=10, currentCompactedKVs=10, compactionProgr
essPct=1.0, coprocessors=[]
        .META.,,1
            numberOfStores=1, numberOfStorefiles=1, storefileUncompressedSizeMB=0, storefileSizeMB=0, memst
oreSizeMB=0, storefileIndexSizeMB=0, readRequestsCount=1431, writeRequestsCount=1, rootIndexSizeKB=0, total
StaticIndexSizeKB=0, totalStaticBloomSizeKB=0, totalCompactingKVs=5, currentCompactedKVs=5, compactionProgr
essPct=1.0, coprocessors=[]
        usertable,,1346085599470.33b9d5d4fb0f7d078ae8848b6db69ce7.
            numberOfStores=1, numberOfStorefiles=1, storefileUncompressedSizeMB=0, storefileSizeMB=0, memst
oreSizeMB=0, storefileIndexSizeMB=0, readRequestsCount=0, writeRequestsCount=0, rootIndexSizeKB=0, totalSta
ticIndexSizeKB=0, totalStaticBloomSizeKB=0, totalCompactingKVs=0, currentCompactedKVs=0, compactionProgress
Pct=NaN, coprocessors=[]
0 dead servers
```

Task 2: Creating A Table in HBase

Activity Procedure

Step 1. Let's find out if there is any table in HBase. In HBase shell type the following:

```
hbase(main):003:0> list
TABLE
usertable
1 row(s) in 0.1540 seconds
```

- Lets create a table with table_name : users and one column with column_name : info.

```
hbase(main):001:0> create 'users', 'info'
0 row(s) in 1.2060 seconds
```

- To find the list of tables in HBase, you can use the following statement.

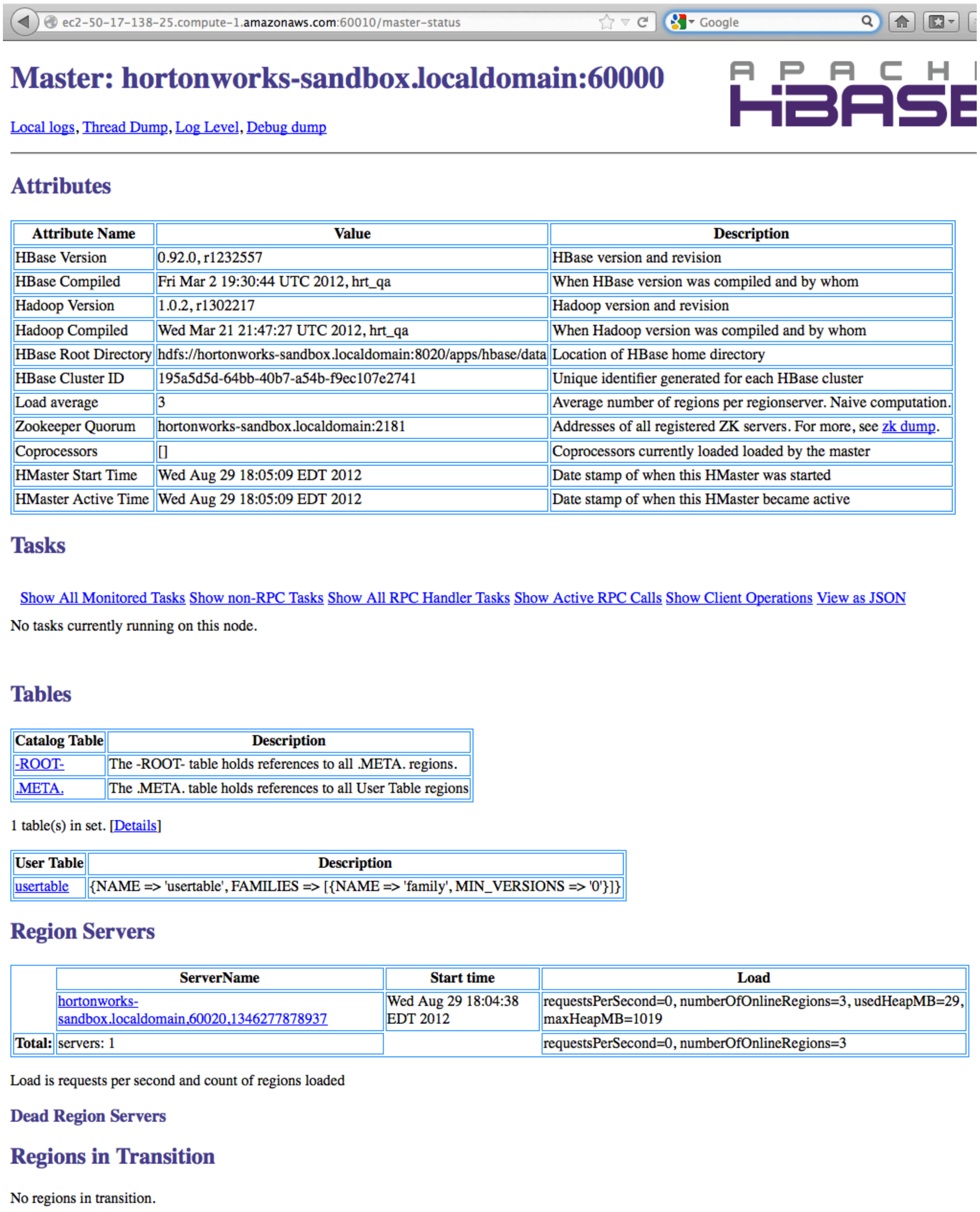
```
Hbase(main):001:0> list
TABLE
users
usertable
2 row(s) in 0.0140 seconds
```

Activity Verification

You have completed this task when you attain these results;

- 'list' command shows 'users' table created in the list
- Also go to HBase UI using the following URL and verify that the table shows in the UI as well:

http://<*EC2_server_name*>:60010/master-status


ec2-50-17-138-25.compute-1.amazonaws.com:60010/master-status

Master: hortonworks-sandbox.localdomain:60000

Local logs, Thread Dump, Log Level, Debug dump

Attributes

Attribute Name	Value	Description
HBase Version	0.92.0, r1232557	HBase version and revision
HBase Compiled	Fri Mar 2 19:30:44 UTC 2012, hrt_qa	When HBase version was compiled and by whom
Hadoop Version	1.0.2, r1302217	Hadoop version and revision
Hadoop Compiled	Wed Mar 21 21:47:27 UTC 2012, hrt_qa	When Hadoop version was compiled and by whom
HBase Root Directory	hdfs://hortonworks-sandbox.localdomain:8020/apps/hbase/data	Location of HBase home directory
HBase Cluster ID	195a5d5d-64bb-40b7-a54b-f9ec107e2741	Unique identifier generated for each HBase cluster
Load average	3	Average number of regions per regionserver. Naive computation.
Zookeeper Quorum	hortonworks-sandbox.localdomain:2181	Addresses of all registered ZK servers. For more, see zk dump.
Coprocessors	[]	Coprocessors currently loaded loaded by the master
HMaster Start Time	Wed Aug 29 18:05:09 EDT 2012	Date stamp of when this HMaster was started
HMaster Active Time	Wed Aug 29 18:05:09 EDT 2012	Date stamp of when this HMaster became active

Tasks

Show All Monitored Tasks Show non-RPC Tasks Show All RPC Handler Tasks Show Active RPC Calls Show Client Operations View as JSON

No tasks currently running on this node.

Tables

Catalog Table	Description
-ROOT-	The -ROOT- table holds references to all .META. regions.
.META.	The .META. table holds references to all User Table regions

1 table(s) in set. [Details]

User Table	Description
usertable	{NAME => 'usertable', FAMILIES => [{NAME => 'family', MIN_VERSIONS => '0'}]}

Region Servers

	ServerName	Start time	Load
	hortonworks-sandbox.localdomain,60020,1346277878937	Wed Aug 29 18:04:38 EDT 2012	requestsPerSecond=0, numberOfOnlineRegions=3, usedHeapMB=29, maxHeapMB=1019
Total:	servers: 1		requestsPerSecond=0, numberOfOnlineRegions=3

Load is requests per second and count of regions loaded

Dead Region Servers

Regions in Transition

No regions in transition.

Task 3: Insert some data into Users table

We are going to enter some data, verify the data. Modify the data and verify again.

Activity Procedure

Complete these steps:

Step 1. Bring up hbase shell

```
$ hbase shell
```

- Enter the following in hbase shell:

```
Hbase(main)> put 'users','user1','info:email', 'user1@foo.com'
Hbase(main)> put 'users','user2','info:email', 'user2@foo.com'
```

- Let's see the contents of users table. Enter this in hbase shell:

```
hbase(main):008:0> scan 'users'
ROW                          COLUMN+CELL
 user1                       column=info:email, timestamp=1326310804390,
   value=user1@foo.com
 user2                       column=info:email, timestamp=1326310833715,
   value=user2@foo.com
2 row(s) in 0.0370 seconds
```

- Lets update 'user1' record. Enter the following in Hbase shell:

```
Hbase(main)> put 'users',  'user1', 'info:email',  'user1@fooooobar.com'
```

- Scan table again:

```
Hbase(main)> scan 'users'
```

The output will be similar to following:

```
ROW             COLUMN+CELL
 user1   column=info:email, timestamp=1322720984603,
   value=user1@fooooobar.com
 user2   column=info:email, timestamp=1322720123249, value=user2@foo.com
```

Activity Verification

You have completed this task when you attain these results;

- Users table has entries
- And the entries can be updated

Task 4: Using Java API to create table and add data

Activity Procedure

Complete these steps:

Step 1. Exit to shell

```
$exit
```

Step 2. Change directory to where the java program we will run resides.

```
$cd ~/labs/hbase
```

- Examine run script: **'run-userput.sh'**
  this is a handy shell script that runs UserPut class.
  The script needs HBASE_HOME to be set, so it can find necessary jar files. Make sure HBASE_HOME is set correctly in the script.

```
$ ./run-userput.sh
```

- If the HBASE_HOME env variable is not set, set environment variables and run the script.

```
$ export HBASE_HOME=/usr/share/hbase
$ cd ~/labs/hbase
$ ./run-userput.sh
```

Activity Verification

You have completed this task when you attain these results;

The script will print out output like following:

```
added user : user1
added user : user2
added user : user3
added user : user4
added user : user5
added user : user6
added user : user7
added user : user8
added user : user9
added user : user10
inserted 10 users  in 25 ms
```

Task 5: Using HBase Java API to query for a record

Activity Procedure

Complete these steps:

Step 1. Change directory to where the java program we will run resides.

```
$cd ~/labs/hbase
```

- Examine run script : '**run-userget.sh**'
  this runs the UserGet java client program.
  The script needs HBASE_HOME to be set, so it can find necessary jar files. Make sure HBASE_HOME is set correctly in the script

```
$ nano run-userget.sh
```

- Run the script:

```
$ ./run-userget.sh
```

Activity Verification

You have completed this task when you attain these results;

You will see output like:

```
querying for : user1 email=user1@foo.com
querying for : user2 email=user2@foo.com
querying for : userXXX userXXX : not found
```

Task 6: Using HBase Java API to scan a table

Activity Procedure

Complete these steps:

Step 1. Change directory to where the java program we will run resides.

```
$cd ~/labs/hbase
```

- Examine run script : '**run-userscan.sh**'
  this runs the UserScan java client program.
  The script needs HBASE_HOME to be set, so it can find necessary jar files. Make sure HBASE_HOME is set correctly in the script

```
$nano run-userscan.sh
```

- Run the script:

```
$ ./run-userscan.sh
```

Activity Verification

You have completed this task when you attain these results;

You will see output like:

```
user1=user1@foo.com
user10=user10@foo.com
user2=user2@foo.com
user3=user3@foo.com
user4=user4@foo.com
user5=user5@foo.com
user6=user6@foo.com
user7=user7@foo.com
user8=user8@foo.com
user9=user9@foo.com
```

Bonus Task: HBase Counters

Activity Procedure

Complete these steps:

Step 1. Examine file Counter.java
This file uses GET API to query for user records

- Examine run script : '**run-counter.sh**'
  this runs the Counter java client program.
  The script needs HBASE_HOME to be set, so it can find necessary jar files. Make sure HBASE_HOME is set correctly in the script

- Run the script:

```
$ ./run-counter.sh
```

Activity Verification

You have completed this task when you attain these results;

You will see output like:

```
google.com : 1
google.com : 2
amazon.com : 1
amazon.com : 2
google.com : 3
amazon.com : 3
amazon.com : 4
google.com : 4
amazon.com : 5
yahoo.com : 1
```

Run the program again. And you will notice counters for each domain picks up from previous values.

Appendix A: Baseball Lab Solution using Java MapReduce

```
//Standard Java imports
import java.io.IOException;
import java.util.Iterator;

import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;

public class Batting {
  public static class MyMap extends MapReduceBase implements
    Mapper<LongWritable, Text, LongWritable, LongWritable> {

     LongWritable outKey = new LongWritable();
     LongWritable outVal = new LongWritable();

     public void map(LongWritable key, Text value,
    OutputCollector<LongWritable, LongWritable> output, Reporter reporter)
         throws IOException {
       String[] values = value.toString().split(",");

       if (StringUtils.isNotEmpty(values[0]) && !values[0].equals("playerID")
    && StringUtils.isNotEmpty(values[1])
           && StringUtils.isNotEmpty(values[8])) {
         outKey.set(Long.valueOf(values[1]));
         outVal.set(Long.valueOf(values[8]));
         output.collect(outKey, outVal);
       }
     }
  }

  public static class MyReduce extends MapReduceBase implements
       Reducer<LongWritable, LongWritable, LongWritable, LongWritable> {

     LongWritable outTotal = new LongWritable();

     public void reduce(LongWritable key, Iterator<LongWritable> values,
         OutputCollector<LongWritable, LongWritable> output, Reporter
    reporter) throws IOException {
       long total = 0;
       while (values.hasNext()) {
        total += values.next().get();
      }
      outTotal.set(total);
      output.collect(key, outTotal);
    }
  }

  public static void main(String[] args) throws IOException {
    // Code to create a new Job specifying the MapReduce class
    final JobConf conf = new JobConf(Batting.class);

    // call different set methods to set the configuration
    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    // add appropriate custom mappers/reducers here

    conf.setMapperClass(MyMap.class);
    conf.setReducerClass(MyReduce.class);
    conf.setMapOutputKeyClass(LongWritable.class);
    conf.setMapOutputValueClass(LongWritable.class);
    conf.setOutputKeyClass(LongWritable.class);
    conf.setOutputValueClass(LongWritable.class);

    // File Input/Output argument passed as a command line argument

    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    // statement to execute the job
    JobClient.runJob(conf);
  }
}
```

Appendix B: Baseball Lab Solution using Pig

Task 1: Review and understand Baseball statistics data files

Activity Procedure

- Create a new directory ‘baseball’ in the ‘labs’ folder:

```
$ mkdir ~/labs/baseball
$ cd ~/labs/baseball
```

- Get the latest baseball stats zip file from Sean Lahman’s web site (version may change):

```
$ wget 'http://seanlahman.com/files/database/lahman591-csv.zip'
```

- Unzip the file. Many statistics files will unpack from the file.

- Examine following files

  a. **`Batting.csv`**

  b. **`Master.csv`**

- Here is column definition for both the files
- We need to consider following field/columns

  c. **Batting.csv**

     i. Column # 1 (Player ID)

     ii. Column # 2 (Year)

     iii. Column # 9 (Runs)

  d.Master.csv

     i. Column # 2 (Player ID)

     ii. Column # 17 (First Name)

     iii. Column # 18 (Last Name)

Task 2: Identify players who scored highest runs for each year

Activity Procedure

Step 1. Create a new directory baseball/input in HDFS and copy batting-data file Batting.csv & master-data files Master.csv to this directory from the local machine

```
$ hadoop fs –mkdir baseball/input

$ hadoop fs –put Batting.csv baseball/input/batting.csv

$ hadoop fs –put Master.csv baseball/input/master.csv

$ hadoop fs –ls baseball/input
```

Step 2. Start the Pig shell by typing following

```
$ pig
grunt>
```

Step 3. Load batting data to Pig using Pig Storage ',':

```
grunt> BATTING = load 'baseball/input/batting.csv' using PigStorage(',');
```

NOTE: A CSV file had data comma-separated in each line. So, we need to inform 'PIG' explicitly about its field delimiter. The default delimiter in PIG is a TAB (\t).

Step 4. Read relevant fields from the file. In this case we are interested in 1st , 2nd and 9th fields for each record:

```
grunt> RUNS = foreach BATTING generate $0 as playerID,  $1 as year, $8 as
   runs;
```

NOTE: $0 represents 1st field, $1 represents 2nd field and so on..

Step 5. Group data for each year together so that we could identify the highest score for each year

```
grunt> GRP_DATA = group RUNS by (year);
grunt> MAX_RUNS = foreach GRP_DATA generate group as grp, MAX(RUNS.runs)
   as max_runs;
```

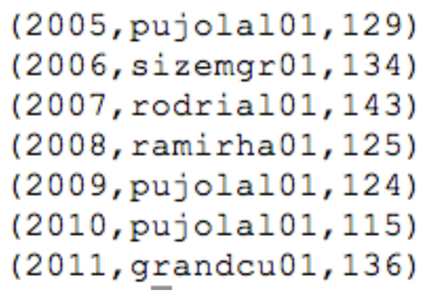
Step 6. Join MAX_RUNS dataset with RUNS dataset to identify the playerID who scored the highest one for each each year and create a new dataset having Year, PlayerID and Max Run data:

```
grunt> JOIN_MAX_RUN = join MAX_RUNS by ($0, max_runs), RUNS by
   (year,runs);
```

```
grunt> JOIN_DATA = foreach JOIN_MAX_RUN generate $0 as year, $2 as
   playerID, $1 as runs;
```

Step 7. Check the output of the above exercise:

```
grunt> dump JOIN_DATA;
```



```
(2005,pujolal01,129)
(2006,sizemgr01,134)
(2007,rodrial01,143)
(2008,ramirha01,125)
(2009,pujolal01,124)
(2010,pujolal01,115)
(2011,grandcu01,136)
```

Task 3: Determine First and Last name for the each player

Activity Procedure

Step 1. Load master data to Pig using following:

```
grunt> MASTER = load 'baseball/input/master.csv' using PigStorage(',');
```

Step 2. Read relevant fields from the file. In this case we are interested in 2nd, 16th & 17th fields for each record:

```
grunt> PLAYERS = foreach MASTER generate $1 as playerID,  $16 as fname,
   $17 as lname;
```

Step 3. Join PLAYERS dataset with the result dataset from previous task based the common field 'playerID':

```
grunt> JOIN_MASTER = join JOIN_DATA by playerID, PLAYERS by playerID;
```

Step 4. Create a new dataset from JOIN_MASTER having Year, Player's First and Last Name and the Max score of the Year fields

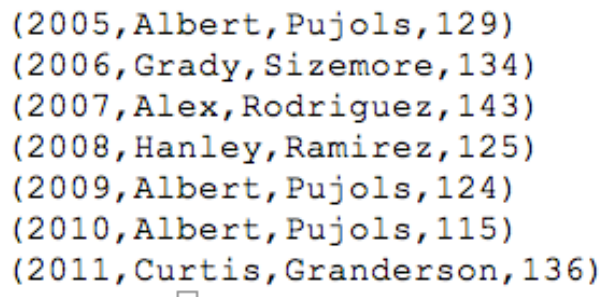
```
grunt> FINAL_DATA = foreach JOIN_MASTER generate $0 as year, $4 as fname,
   $5 as lname, $2 as runs;
```

Step 5. Making sure that data is sorted on Year in ascending order

```
grunt> RESULT = order FINAL_DATA by year asc;
```

Step 6. Get the output of RESULT dataset:

`grunt> dump RESULT;`



```
(2005,Albert,Pujols,129)
(2006,Grady,Sizemore,134)
(2007,Alex,Rodriguez,143)
(2008,Hanley,Ramirez,125)
(2009,Albert,Pujols,124)
(2010,Albert,Pujols,115)
(2011,Curtis,Granderson,136)
```

Appendix C: Baseball Lab Solution using Hive

Task 1: Identify players who scored highest runs for each year

Activity Procedure

Step 1. Change to the baseball directory:

```
$ cd C:\labs\baseball
```

Step 2. Start the Hive shell by typing following

```
$ hive
hive>
```

Step 3. Create a temporary tables to store batting data into hive:

```
hive> create table temp_batting (col_value STRING);
```

Step 4. Insert data into the table from the local CSV files:

```
hive> LOAD DATA LOCAL INPATH 'Batting.csv' OVERWRITE INTO TABLE
    temp_batting;
```

Step 5. Create another table to store only relevant columns for batting into hive:

```
hive> create table batting (player_id STRING, year INT, runs INT) ;
```

Step 6. Read relevant fields from temp_batting table. In this case we are interested in 1st , 2nd and 9th fields for each record :

```
hive> INSERT OVERWRITE TABLE batting
 SELECT
        regexp_extract(col_value, '^(?:([^,]*)\,?){1}', 1) player_id,
        regexp_extract(col_value, '^(?:([^,]*)\,?){2}', 1) year,
        regexp_extract(col_value, '^(?:([^,]*)\,?){9}', 1) run
FROM temp_batting;
```

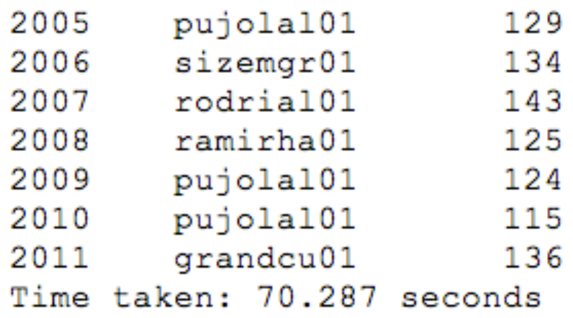
NOTE: We are using built-in String function '**regexp_extract(str, regular expression, index)**' to identify the right column. You can get more information on this function at following location: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF

Step 7. Group data for each year together so that we could identify the highest score for each year

```
hive> SELECT year, max(runs) FROM batting GROUP BY year;
```

Step 8. Join above dataset with batting table to identify the player_id who scored the highest one for each year and create a new query having year, player_id and Max Run data:

```
hive> SELECT a.year, a.player_id, a.runs
 FROM batting a
 JOIN (SELECT year, max(runs) runs FROM batting GROUP BY year) b
 ON (a.year = b.year AND a.runs = b.runs);
```



```
2005    pujolal01       129
2006    sizemgr01       134
2007    rodrial01       143
2008    ramirha01       125
2009    pujolal01       124
2010    pujolal01       115
2011    grandcu01       136
Time taken: 70.287 seconds
```

Task 2: Determine First and Last name for the each player

Activity Procedure

Step 1. Create a temporary tables to store master data into hive:

```
hive> create table temp_master (col_value STRING);
```

Step 2. Insert data into the table from the local CSV files:

```
hive> LOAD DATA LOCAL INPATH 'Master.csv' OVERWRITE INTO TABLE
   temp_master;
```

Step 3. Create another table to store only relevant columns for batting into hive:

```
hive> create table master (player_id STRING, fname STRING, lname STRING);
```

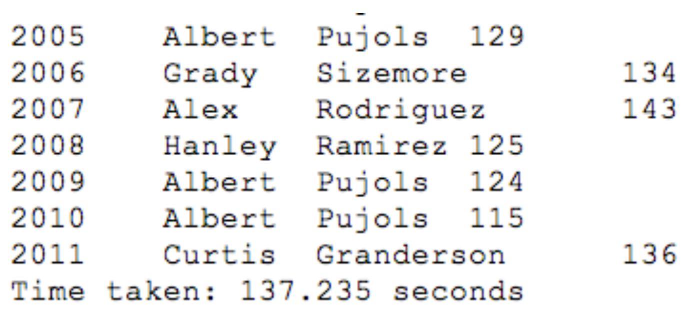
Step 4. Read relevant fields from temp_master table. In this case we are interested in 2nd, 16th & 17th fields for each record:

```
hive> INSERT OVERWRITE TABLE master
 SELECT
        regexp_extract(col_value, '^(?:([^,]*)\,?){2}', 1),
        regexp_extract(col_value, '^(?:([^,]*)\,?){17}', 1),
        regexp_extract(col_value, '^(?:([^,]*)\,?){18}', 1)
```

```
FROM temp_master;
```

Step 5. Join the final query from previous task with master table to get First and Last name of the player and make sure data is sorted based on year in ascending order:

```
hive> SELECT
   a.year, c.fname, c.lname, a.runs
 FROM batting a
 JOIN (SELECT year, max(runs) runs FROM batting GROUP BY year) b
 ON (a.year = b.year AND a.runs = b.runs)
 JOIN master c
 ON (c.player_id = a.player_id)
 ORDER BY a.year asc;
```



```
2005    Albert  Pujols  129
2006    Grady   Sizemore        134
2007    Alex    Rodriguez       143
2008    Hanley  Ramirez 125
2009    Albert  Pujols  124
2010    Albert  Pujols  115
2011    Curtis  Granderson      136
Time taken: 137.235 seconds
```

Task 3: Save the output from previous task to a new table

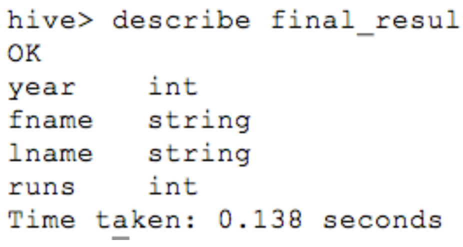
Activity Procedure

Step 1. Create a new table final_result using output of final query from previous task:

```
hive> CREATE TABLE final_result AS
 SELECT
   a.year, c.fname, c.lname, a.runs
 FROM batting a
 JOIN (SELECT year, max(runs) runs FROM batting GROUP BY year) b
 ON (a.year = b.year AND a.runs = b.runs)
 JOIN master c
 ON (c.player_id = a.player_id)
 ORDER BY a.year asc;
```

Step 2. Get the column definition of newly created table:

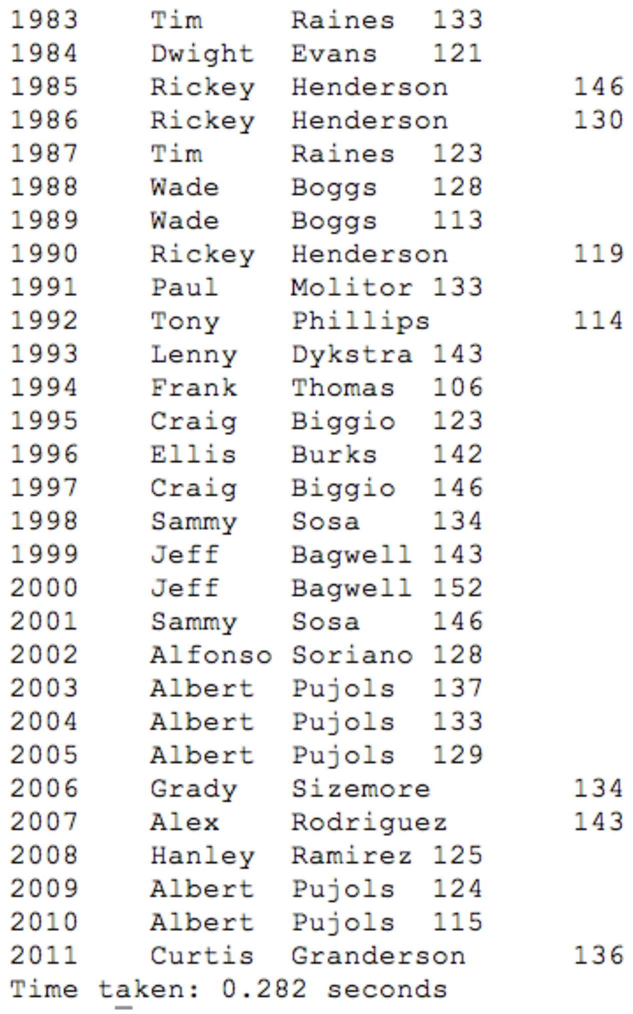
```
hive> describe final_result;
```



```
hive> describe final_result;
OK
year    int
fname   string
lname   string
runs    int
Time taken: 0.138 seconds
```

Step 3. Check the data in this table:

```
hive> select * from final_result;
```



```
1983    Tim     Raines  133
1984    Dwight  Evans   121
1985    Rickey  Henderson       146
1986    Rickey  Henderson       130
1987    Tim     Raines  123
1988    Wade    Boggs   128
1989    Wade    Boggs   113
1990    Rickey  Henderson       119
1991    Paul    Molitor 133
1992    Tony    Phillips        114
1993    Lenny   Dykstra 143
1994    Frank   Thomas  106
1995    Craig   Biggio  123
1996    Ellis   Burks   142
1997    Craig   Biggio  146
1998    Sammy   Sosa    134
1999    Jeff    Bagwell 143
2000    Jeff    Bagwell 152
2001    Sammy   Sosa    146
2002    Alfonso Soriano 128
2003    Albert  Pujols  137
2004    Albert  Pujols  133
2005    Albert  Pujols  129
2006    Grady   Sizemore        134
2007    Alex    Rodriguez       143
2008    Hanley  Ramirez 125
2009    Albert  Pujols  124
2010    Albert  Pujols  115
2011    Curtis  Granderson      136
Time taken: 0.282 seconds
```