Apache Hadoop – A Developer's Course

LABS

Lab 1: Login to remote machine through a terminal

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 the far upper right hand corner of your screen and click on the magnifying glass icon.
Step 2: Type in terminal and select Terminal application.

For a PC:

Step 1: You will need an SSH client like *putty* or *SecureCRT*. Your instructor will direct you on the appropriate application on your machine.
Step 2: Invoke putty and specify an ssh connection to the remote machine.

Activity Verification

When the job completes, you will see a terminal window on your screen.

Task 2: Logging into the lab

Activity Procedure

For a Mac:

Step 1. In your terminal window type

```
$ ssh shrek@<machinename>
```

Your instructor will tell you the <machinename> allocated to you.
Step 2: To the question "Are you sure you want to continue connecting?", type **yes**
Step 3: Then type the password provided by your instructor.

For a PC:

Step 1: Bring up a terminal window in putty. To avoid typing in the machine name every time, you should save the configuration and load it next time.
Step 2: Enter the username **`shrek`**
Step 3: Enter the password provided by your instructor.

Activity Verification

When the job completes you will be logged in to the remote machine and see the prompt

```
shrek@<machinename>:[~]
```

<*machinename*> will be replaced with the actual name of your machine.

Task 3: Find out the contents of your home directory

Activity Procedure

Step 1: Check your current directory

```
$ pwd
```

Your directory should be **`/home/shrek`**
Step 2: Check the contents of your home directory

```
$ ls
```

Activity Verification

When the job completes, you will see the contents of your home directory:

```
Desktop  Developer  Documents  Downloads  Music  Pictures  Public  Templates  Videos
workspace
```

Your source code is in the folder **`Developer/heffalump`**. Data files are in the folder **`Developer/heffalump/data`**.

Lab 2: Get familiar with HDFS commands

Description:

- Practice basic 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/shrek`**.

```
$ hadoop fs –ls
```

Step 2: Check the root folder of the HDFS.

```
$ hadoop fs –ls /
```

Step 3: To check the directory structure recursively:

```
$ hadoop fs –ls -R /user/shrek
```

Task 2: File/Directory creation/deletion commands

Activity Procedure and Verification

Step 1: Make a new directory in HDFS

```
$ hadoop fs –mkdir test
```

Step 2: Make a new sub-directory in HDFS

```
$ hadoop fs –mkdir test/test1
$ Hadoop fs –mkdir test/test2/test3
```

Step 3: Make sure the directory has been created. Do a recursive directory listing:

```
$ hadoop fs –ls -R /user/shrek/
```

You should see:

```
$ hadoop fs -ls -R /user/shrek
drwxr-xr-x   - shrek supergroup          0 2013-06-03 21:44 /user/shrek/test
```

```
drwxr-xr-x   - shrek supergroup          0 2013-06-03 21:44 /user/shrek/test/test1
drwxr-xr-x   - shrek supergroup          0 2013-06-03 21:44 /user/shrek/test/test2
drwxr-xr-x   - shrek supergroup          0 2013-06-03 21:44 /user/shrek/test/test2/test3
```

Step 4: Copy input data file to the HDFS home directory

```
$ cd ~/Developer/heffalump/data
$ hadoop fs –mkdir input
$ hadoop fs –put sample1.txt input
```

Step 5: Check to see if the file was copied correctly to HDFS.

```
$ hadoop fs -ls -R
drwxr-xr-x   - shrek supergroup          0 2013-06-03 21:52 input
-rw-r--r--   1 shrek supergroup       2777 2013-06-03 21:52 input/sample1.txt
drwxr-xr-x   - shrek supergroup          0 2013-06-03 21:44 test
drwxr-xr-x   - shrek supergroup          0 2013-06-03 21:44 test/test1
drwxr-xr-x   - shrek supergroup          0 2013-06-03 21:44 test/test2
drwxr-xr-x   - shrek supergroup          0 2013-06-03 21:44 test/test2/test3
```

Step 6: Make a copy of the file in HDFS to another file within HDFS.

```
$ hadoop fs –cp input/sample1.txt input/copy_of_sample1.txt
```

Step 7: Check to see that both files exist:

```
$ hadoop fs -ls -R input
Found 2 items
-rw-r--r--   1 shrek supergroup       2777 2013-06-03 21:58 input/copy_of_sample1.txt
-rw-r--r--   1 shrek supergroup       2777 2013-06-03 21:52 input/sample1.txt
```

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

```
$ hadoop fs –rm input/copy_of_sample1.txt
```

Step 9: Check that the file was removed.

```
$ hadoop fs –ls -R input
```

Step 10: Remove the entire directory **test** and **input**

```
$ hadoop fs –rm -r input
$ hadoop fs –rm –r test
```

Task 3: File/Directory permissions related commands

Activity Procedure and Verification

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

```
$ cd ~/Developer/heffalump
$ hadoop fs -mkdir input
$ hadoop fs -copyFromLocal data/sample1.txt input
$ hadoop fs -ls input
```

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

```
$ hadoop fs -chmod 777 input/sample1.txt
```

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

```
$ hadoop fs -ls input
```

Task 4: Commands to read a file

Activity Procedure and Verification

Step 1: Check the contents of the file in HDFS

```
$ hadoop fs -cat input/sample1.txt
```

Step 2: To check the contents of an HDFS file, but page wise:

```
$ hadoop fs -cat input/sample1.txt | more
```

Step 3: To check the contents of an HDFS file, but last 20 lines:

```
$ hadoop fs -tail input/sample1.txt
```

Task 5: Other useful commands

Activity Procedure and Verification

Step 1: To find out the disk usage in you HDFS directory, use the following. 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, use the following.

```
$ hadoop fs -count /user/shrek
           2            1                2777 /user/shrek
```

The output columns are DIR_COUNT, FILE_COUNT, CONTENT_SIZE, FILE_NAME.

Step 3: To get help on other **`hadoop fs`** commands, type.

```
$ hadoop fs –help [cmd]
```

Task 6: Exercise – Learning the 'getmerge' command

Activity Procedure and Verification

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

Item, Manufacturer, SKU, Price, Stock

e.g. File1.txt

```
Table Lamp, Plantronics, 4432553434, 19.99, 15
Mouse, Dell, 8343840409, 26.95, 20
Headphone, Sony, 8043904973, 19.99, 10
Keyboard, Logitech, 8034790540, 34.99, 15
USB Cable, Fastdata, 8453597693, 3.99, 13
```

e.g. File2.txt

```
Hard Disk, Maxtor, 4358340984, 119.99,
Stylus, Nexis, 8304803480, 15.49, 10
Adapter, Fry, 7400340943, 14.99, 14
Stapler, Staples, 8944739774, 5.99, 9
Monitor, Samsung, 884084038, 199.99, 5
```

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

Step 3: Get help on getmerge command and use it for these two data files.

```
$ hadoop fs –help getmerge
```

Step 4: Review the output file. Do you notice any merging happening in the results?

Lab 3: Program HDFS

Description:

- Use Java to accomplish HDFS tasks
- Read a file from local Linux file system
- Write a file to the HDFS

Task 1: Set up the build environment

Activity Procedure and Verification

Step 1: Go to the location of the source code for the labs.

```
$ cd ~/Developer/heffalump
```

Step 2: Check that your environment variables are set correctly. We will be using Maven to build our sources. Thus at the beginning make sure that the following are correct.

```
$ set | grep JAVA_HOME
JAVA_HOME=/usr/java/latest
$ set | grep M2_HOME
M2_HOME=/usr/local/apache-maven/apache-maven-3.0.5
```

You only need to ensure that these values are set. They need not necessarily exactly match the above.

Step 3: Try building the source code using Maven.

```
$ mvn package
```

You'll see something similar to the following:

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building heffalump 1.0
[INFO] ------------------------------------------------------------------------
[WARNING] The POM for org.json:JSON:jar:1.0 is missing, no dependency information
available
[WARNING] The POM for com.maxmind:geoip-api:jar:1.2.11 is missing, no dependency
information available
[INFO]
[INFO] --- maven-resources-plugin:2.5:resources (default-resources) @ heffalump ---
[debug] execute contextualize
```

```
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e.
build is platform dependent!
[INFO] skip non existing resourceDirectory
/home/shrek/Developer/heffalump/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.1:compile (default-compile) @ heffalump ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.5:testResources (default-testResources) @
heffalump ---
[debug] execute contextualize
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e.
build is platform dependent!
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:2.3.1:testCompile (default-testCompile) @ heffalump
---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.10:test (default-test) @ heffalump ---
[INFO] Surefire report directory: /home/shrek/Developer/heffalump/target/surefire-
reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.hadooptraining.test.lab7.TestLogProcessorWithCustomWritable
WARNING: org.apache.hadoop.metrics.jvm.EventCounter is deprecated. Please use
org.apache.hadoop.log.metrics.EventCounter in all the log4j.properties files.
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.624 sec

Results :

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.3.1:jar (default-jar) @ heffalump ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.231s
[INFO] Finished at: Thu Jun 06 06:20:54 PDT 2013
[INFO] Final Memory: 13M/309M
[INFO] ------------------------------------------------------------------------
```

The first time, Maven may download some jar files from the internet, which will be indicated in the output. Thus your compilation may take longer.

Task 2: Identify the input file for the program

Activity Procedure and Verification

Step 1: Identify the location of the input file for this lab and see the first few lines.

```
$ pwd
/home/shrek/Developer/heffalump

$ ls -al data/sample1.txt
-rw-r--r--. 1 shrek shrek 2777 May 24 19:59 data/sample1.txt

$ head data/sample1.txt
2. Friendship

A certain father was doubly blessed—he had reached a good old age, and
had ten sons. One day he called them to his side, and after repeated
expressions of affection, told them that he had acquired a fortune by
industry and economy, and would give them one hundred gold pieces each
before his death, so that they might begin business for themselves, and
not be obliged to wait until he had passed away. It happened, however,
that, soon after, he lost a portion of his property, much to his regret,
and had only nine hundred and fifty gold pieces left. So he gave one
```

Task 3: Write a Java class to explore programming HDFS

In this class we are going to write a Java program to copy the input file from the local file system to HDFS. We provide below a step-by-step program to accomplish the above.

The source code is arranged according to the lab number. Since this is lab3, the source code for this lab will be found in the following directory:

```
$ ls -l src/main/java/com/hadooptraining/lab3
total 12
drwxrwxr-x.  2 shrek shrek 4096 May 27 15:08 .
drwxrwxr-x. 14 shrek shrek 4096 May 30 06:44 ..
-rw-rw-r--.  1 shrek shrek 2492 May 27 15:08 FileCopy.java
```

For every lab, you are supposed to create or modify an existing file in the appropriate location (as per the lab number), and run the build using `mvn package`.

Activity Procedure and Verification

Step 1: Edit the file for this lab. You can use your favorite editor. **vi** and **emacs** are both available on the machine.

```
vi src/main/java/com/hadooptraining/lab3/FileCopy.java
```

Step 2: Declare the package for this class.

```
package com.hadooptraining.lab3;
```

Step 3: Include the required libraries for HDFS APIs.

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.LocalFileSystem;
import org.apache.hadoop.fs.Path;
```

Step 4: Import the library for file I/O.

```
import java.io.IOException;
```

Step 5: Create a class to work with the local file system and HDFS.

```
public class FileCopy {
```

Step 6: Declare a method to copy a file. It takes two parameters - an inPath and an outPath. The inPath takes a file path in the local file system. The outPath is the output path in the HDFS.

```
    public void copyFile(Path inPath, Path outPath) throws IOException {
```

Step 7: Instantiate a default Configuration object. This is needed for our work with the **FileSystem** factory methods.

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

Step 8: We want to write 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 9: We want to read a file from the Linux file-system. Acquire a **LocalFileSystem** object which represents the local Linux file-system by invoking the public static **FileSystem** factory method **getLocal(Configuration conf)** such as:

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

Step 10: 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 out 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 11: We want to write data to HDFS using a file-system data output stream. The output stream we want is one wrapped around our output **`Path`** argument to this method, we can just pass that directly to the **`FileSystem create()`** factory method, such as:

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

Step 12: 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 13: Read the input file. This can be achieved by a simple **`read()`** method inherited from the **`java.io.DataInputStream`**. 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 the file (EOF), as value of **`-1`** is returned. Loop over the input stream's **`read()`** method until you reach the end of the file, for example:

```
        int datalength;
        while ((datalength = inStream.read(fromFile)) > 0) {
```

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

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

Step 12: Close the streams:

```
        }
        inStream.close();
        outStream.close();
    }
```

Step 13: Write the **`main()`** method for the class. **`main()`** methods always have the same signature.

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

Step 14: We need an inputPath and an outputPath as arguments to the main(). If the number of arguments is less that two, then you should send an error message to the output. Proceed to copy the file if the number of arguments is exactly two.

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

Step 15: Assign two variables - the **`inputPath`** from the first argument, and the **`outputPath`** from the second argument.

```
            String inputPath = args[0];
            String outputPath = args[1];
```

Step 16: Since our method could throw an exception, we have to put our copy code inside a try-catch block.

```
            try {
```

Step 17: Create a new object for the current class. This object will be used to execute the file copy operation.

```
                FileCopy copier = new FileCopy();
```

Step 18: Use the object's **`copyFile`** method to copy the file. We are calling the method **`copyFile(Path, Path)`** that we just wrote above.

```
                copier.copyFile(new Path(inputPath), new Path(outputPath));
```

Step 19: Catch the exception and print a stack trace. This is written out if a problem happened with the copy operation.

```
            } catch (IOException e) {
                e.printStackTrace();
            }
```

Step 20: If the number of arguments provided does not match two, write a message on the screen and exit.

```
        } else {
            System.out.println("You need to provide two arguments when calling
FileCopy");
        }
    }
}
```

Step 21: Exit out of your editor (**`:x`** for **`vi`** users, **`Ctrl-X Ctrl-C`** for **`emacs`** users). Compile the file and make it part of the package.

```
$ mvn package
```

This results in a jar file inside your target directory.

```
$ ls -l target
total 128
drwxrwxr-x. 3 shrek shrek   4096 Jun  2 23:25 classes
drwxrwxr-x. 4 shrek shrek   4096 Jun  2 23:25 generated-sources
-rw-rw-r--. 1 shrek shrek 104158 Jun  6 06:20 heffalump-1.0.jar
drwxrwxr-x. 2 shrek shrek   4096 Jun  2 23:39 maven-archiver
drwxrwxr-x. 2 shrek shrek   4096 Jun  6 06:20 surefire
drwxrwxr-x. 2 shrek shrek   4096 Jun  2 23:25 surefire-reports
drwxrwxr-x. 3 shrek shrek   4096 Jun  2 23:25 test-classes
```

Make sure that you see the jar file **heffalump-1.0.jar** in your output.

Step 22: Prepare the input file to copy. We have several files in the data directory. This time we are going to copy a file called **sample1.txt**. In Task 2 above we have already validated that the file exists and seen the first few lines of this file.

Step 23: Invoke the script **hadoop** with the generated jar file and the appropriate arguments to copy the file. **hadoop** is a script that sets up the appropriate classpaths containing the Hadoop jar files.

```
$ hadoop jar /home/shrek/Developer/heffalump/target/heffalump-1.0.jar
com.hadooptraining.lab3.FileCopy data/sample1.txt output/sample1_copy.txt
```

Step 24: Check if the file got copied.

```
$ hadoop fs -ls output
Found 1 items
-rw-r--r--   1 shrek supergroup       2777 2013-06-08 10:12 output/sample1_copy.txt
```

Step 25: See the contents of the copied file:

```
$ hadoop fs -cat output/sample1_copy.txt
2. Friendship

A certain father was doubly blessed—he had reached a good old age, and
had ten sons. One day he called them to his side, and after repeated
expressions of affection, told them that he had acquired a fortune by
...
```

Lab 4: Monitor HDFS

Description:

- Practice using basic monitoring tasks in HDFS and explore various configurations and variables
- Explore NameNode and DataNode configuration
- Use the HDP web-based GUI to browse web user interfaces
- Examine Hadoop primary daemons and environmental variables
- Start and stop Hadoop services

Task 1: Checking Hadoop configuration files

Hadoop has two kinds of configuration files.

a) Default configuration files that reside inside the **`hadoop-common.jar`** file.
b) The other kind are user editable configuration files, which can be found under **`/etc/hadoop/conf`**.

Activity Procedure and Verification

Step 1: Identify the configuration files under **`/etc/hadoop/conf`**.

```
$ cd /etc/hadoop/conf
$ ls -l
total 32
-rwxr-xr-x. 1 root root 1461 May 27 20:00 core-site.xml
-rwxr-xr-x. 1 root root 2890 May 27 20:00 hadoop-metrics.properties
-rwxr-xr-x. 1 root root 1875 May 27 20:00 hdfs-site.xml
-rwxr-xr-x. 1 root root 8735 May 27 20:00 log4j.properties
-rwxr-xr-x. 1 root root  582 May 27 20:00 mapred-site.xml
-rwxr-xr-x. 1 root root 1344 May 27 20:00 README
```

Step 2: Examine the configuration file **`hdfs-site.xml`**.

```
$ less hdfs-site.xml
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <!-- Immediately exit safemode as soon as one DataNode checks in.
       On a multi-node cluster, these configurations must be removed.  -->
  <property>
    <name>dfs.safemode.extension</name>
    <value>0</value>
  </property>
  <property>
     <name>dfs.safemode.min.datanodes</name>
     <value>1</value>
  </property>
...
```

Notice how the data directory is defined.

```
  <property>
     <name>dfs.namenode.name.dir</name>
     <value>file:///var/lib/hadoop-hdfs/cache/${user.name}/dfs/name</value>
  </property>
  <property>
     <name>dfs.namenode.checkpoint.dir</name>
     <value>file:///var/lib/hadoop-
hdfs/cache/${user.name}/dfs/namesecondary</value>
  </property>
  <property>
     <name>dfs.datanode.data.dir</name>
     <value>file:///var/lib/hadoop-hdfs/cache/${user.name}/dfs/data</value>
  </property>
```

Type '**q**' to exit out of **less**.

Step 3: Examine the MapReduce configurations.

```
$ less mapred-site.xml
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:8021</value>
  </property>

  <!-- Enable Hue plugins -->
  <property>
    <name>mapred.jobtracker.plugins</name>
    <value>org.apache.hadoop.thriftfs.ThriftJobTrackerPlugin</value>
    <description>Comma-separated list of jobtracker plug-ins to be activated.
    </description>
  </property>
  <property>
```

```
        <name>jobtracker.thrift.address</name>
        <value>0.0.0.0:9290</value>
      </property>
    </configuration>
```

Task 2: Checking Hadoop core configuration files

Let us now examine the configuration files embedded inside the **hadoop-common.jar** file.

Activity Procedure and Verification

Step 1: Expand the **hadoop-common.jar** file and look at the configuration file embedded inside it.

```
$ mkdir /tmp/hadoop-jars
$ cd /tmp/hadoop-jars
$ jar -xvf /usr/lib/hadoop/hadoop-common.jar
$ less core-default.xml
<property>
  <name>hadoop.common.configuration.version</name>
  <value>0.23.0</value>
  <description>version of this configuration file</description>
</property>

<property>
  <name>hadoop.tmp.dir</name>
  <value>/tmp/hadoop-${user.name}</value>
  <description>A base for other temporary directories.</description>
</property>
...
```

You will see many more properties, all of which are well documented within the file. Some properties in the common file will be same as the editable configuration file you had seen earlier under **/etc/hadoop/conf**.

Exit out of less by pressing '**q**'. Then, clean up the temporary directory.

```
$ rm -rf /tmp/hadoop-jars
```

Task 3: Check Hadoop primary daemons and Environment variables

Activity Procedure and Verification

Step 1: To view a list of Hadoop processes running on your machine, create the following alias.

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

Then run the alias to see a list of Hadoop daemons running on your machine.

```
$ hps
shrek  2525   Dproc
mapred 2621   -Dproc_jobtracker
mapred 2722   -Dproc_tasktracker
hdfs   2808   -Dproc_datanode
hdfs   2902   -Dproc_namenode
hdfs   3038   -Dproc_secondarynamenode
```

Step 2: Find out where the Hadoop binaries are located.

```
$ which hadoop
/usr/bin/hadoop
```

Task 4: Checking Hadoop web-based GUI

Activity Procedure and Verification

Step 1: Open the following URLs in your browser.

a) http://localhost:50070 - for Namenode GUI
b) http://localhost:50030 - for JobTracker GUI

Click the links on these pages to explore what the GUI has to offer. We will get into these pages in more detail later.

Task 5: Starting and stopping Hadoop

Activity Procedure and Verification

Step 1: Hadoop may be started/stopped through a script or manually by choosing each process individually. All of these commands work only as super-user. We will first stop all Hadoop individual processes and then start them again.

```
$ su -
```

Ask instructor for the password.

```
$ sudo service hadoop-httpfs stop
$ sudo service hadoop-mapreduce-historyserver stop
$ sudo service hadoop-0.20-mapreduce-tasktracker stop
$ sudo service hadoop-0.20-mapreduce-jobtracker stop
$ sudo service hadoop-hdfs-secondarynamenode stop
$ sudo service hadoop-hdfs-namenode stop
$ sudo service hadoop-hdfs-datanode stop
```

Step 2: Hadoop may be started by providing similar commands.

```
$ sudo service hadoop-hdfs-datanode start
$ sudo service hadoop-hdfs-namenode start
$ sudo service hadoop-hdfs-secondarynamenode start
$ sudo service hadoop-0.20-mapreduce-jobtracker start
$ sudo service hadoop-0.20-mapreduce-tasktracker start
$ sudo service hadoop-mapreduce-historyserver start
$ sudo service hadoop-httpfs start
```

Check that all Hadoop processes are up.

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

Then run the alias to see a list of Hadoop daemons running on your machine.

```
$ hps
mapred 2621   -Dproc_jobtracker
mapred 2722   -Dproc_tasktracker
hdfs   2808   -Dproc_datanode
hdfs   2902   -Dproc_namenode
hdfs   3038   -Dproc_secondarynamenode
root   3493   Dproc
```

You should see the Hadoop processes with different process numbers than before.

Step 3: Exit out of **root** for your next lab.

```
$ exit
```

Lab 5: Run a MapReduce job

Description:

- Build and run a MapReduce job that is already written
- Inspect Hadoop built-in **`map()`** and **`reduce()`** code
- Review code for **`Job`** setup and **`Configuration`**
- Run the job

Task 1: Loading data into HDFS

Activity Procedure

Most text searching systems rely on inverted indices to look up set of documents that contains a given word or a term. In this lab, we are going to run a simple inverted index program that computes a list of terms in the documents, the set of documents that contains each term, and the term frequency in each of the documents. Retrieval of results from an inverted index can be as simple as returning the set of documents that contain the given terms, or involve much more complex operations such as returning the set of documents ordered based on a particular ranking.

Step 1: Go to the location of the source code for the labs.

```
$ cd ~/Developer/heffalump
```

Step 2: Inspect the code at the appropriate lab directory. Since this is lab5, you need to look at files under

```
$ cd src/main/java/com/hadooptraining/lab5
$ vi TextOutInvertedIndexer.java
```

Notice that the class has three sub-classes, IndexingMapper, IndexingReducer and IndexingCombiner. These are the mapper, reducer and combiner classes. We will talk more about the role of these classes as we go along. This class also has a main function at the end where we set up a configuration object and a job, then assign the mapper, reducer and combiner, and finally call the job execution method. This is part of a boiler-plate code that you will reuse for almost all following labs. Note down the pattern of calls very carefully.

Step 3: The inputs to this program are a set of files with some text in it. This program will analyze that text and report how many times the words appeared in each file. You thus need to copy some text files to the HDFS input folder after cleaning it up. After copying, list the files to see how many files are being analyzed.

```
$ hadoop fs -rm -r output
$ hadoop fs -rm -r input
$ hadoop fs -mkdir input
$ hadoop fs -copyFromLocal -f data/sample*.txt input
```

Activity Verification

After copying the files, you'll find something similar to the following.

```
$ hadoop fs -ls input
Found 5 items
-rw-r--r--   1 shrek supergroup       2777 2013-06-09 11:46 input/sample1.txt
-rw-r--r--   1 shrek supergroup      19374 2013-06-09 11:46 input/sample2.txt
-rw-r--r--   1 shrek supergroup       3701 2013-06-09 11:46 input/sample3.txt
-rw-r--r--   1 shrek supergroup       3677 2013-06-09 11:46 input/sample4.txt
-rw-r--r--   1 shrek supergroup      14921 2013-06-09 11:46 input/sample5.txt
```

Task 2: Building and Running the MapReduce job

Activity Procedure

Step 1: Build the program, as is.

```
$ cd ~/Developer/heffalump
$ mvn package
```

Step 2: Run the MapReduce job.

```
$ hadoop jar $HOME/Developer/heffalump/target/heffalump-1.0.jar
com.hadooptraining.lab5.TextOutInvertedIndexer input output
```

Activity Verification

As the job runs, you'll see output on the screen that looks like the following:

```
13/06/09 16:26:16 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/06/09 16:26:16 INFO input.FileInputFormat: Total input paths to process : 5
13/06/09 16:26:16 INFO mapred.JobClient: Running job: job_201306011545_0011
13/06/09 16:26:17 INFO mapred.JobClient:  map 0% reduce 0%
13/06/09 16:26:24 INFO mapred.JobClient:  map 40% reduce 0%
13/06/09 16:26:28 INFO mapred.JobClient:  map 80% reduce 0%
13/06/09 16:26:30 INFO mapred.JobClient:  map 100% reduce 0%
13/06/09 16:26:33 INFO mapred.JobClient:  map 100% reduce 100%
13/06/09 16:26:33 INFO mapred.JobClient: Job complete: job_201306011545_0011
13/06/09 16:26:33 INFO mapred.JobClient: Counters: 32
13/06/09 16:26:33 INFO mapred.JobClient:   File System Counters
13/06/09 16:26:33 INFO mapred.JobClient:     FILE: Number of bytes read=60815
13/06/09 16:26:33 INFO mapred.JobClient:     FILE: Number of bytes written=1286319
13/06/09 16:26:33 INFO mapred.JobClient:     FILE: Number of read operations=0
```

```
13/06/09 16:26:33 INFO mapred.JobClient:     FILE: Number of large read operations=0
13/06/09 16:26:33 INFO mapred.JobClient:     FILE: Number of write operations=0
13/06/09 16:26:33 INFO mapred.JobClient:     HDFS: Number of bytes read=45025
13/06/09 16:26:33 INFO mapred.JobClient:     HDFS: Number of bytes written=48099
13/06/09 16:26:33 INFO mapred.JobClient:     HDFS: Number of read operations=10
13/06/09 16:26:33 INFO mapred.JobClient:     HDFS: Number of large read operations=0
13/06/09 16:26:33 INFO mapred.JobClient:     HDFS: Number of write operations=1
13/06/09 16:26:33 INFO mapred.JobClient:   Job Counters
13/06/09 16:26:33 INFO mapred.JobClient:     Launched map tasks=5
13/06/09 16:26:33 INFO mapred.JobClient:     Launched reduce tasks=1
13/06/09 16:26:33 INFO mapred.JobClient:     Data-local map tasks=5
13/06/09 16:26:33 INFO mapred.JobClient:     Total time spent by all maps in occupied slots
(ms)=19958
13/06/09 16:26:33 INFO mapred.JobClient:     Total time spent by all reduces in occupied slots
(ms)=8810
13/06/09 16:26:33 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving
slots (ms)=0
13/06/09 16:26:33 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving
slots (ms)=0
13/06/09 16:26:33 INFO mapred.JobClient:   Map-Reduce Framework
13/06/09 16:26:33 INFO mapred.JobClient:     Map input records=838
13/06/09 16:26:33 INFO mapred.JobClient:     Map output records=8610
13/06/09 16:26:33 INFO mapred.JobClient:     Map output bytes=180359
13/06/09 16:26:33 INFO mapred.JobClient:     Input split bytes=575
13/06/09 16:26:33 INFO mapred.JobClient:     Combine input records=8610
13/06/09 16:26:33 INFO mapred.JobClient:     Combine output records=2516
13/06/09 16:26:33 INFO mapred.JobClient:     Reduce input groups=1665
13/06/09 16:26:33 INFO mapred.JobClient:     Reduce shuffle bytes=60839
13/06/09 16:26:33 INFO mapred.JobClient:     Reduce input records=2516
13/06/09 16:26:33 INFO mapred.JobClient:     Reduce output records=1665
13/06/09 16:26:33 INFO mapred.JobClient:     Spilled Records=5032
13/06/09 16:26:33 INFO mapred.JobClient:     CPU time spent (ms)=5940
13/06/09 16:26:33 INFO mapred.JobClient:     Physical memory (bytes) snapshot=1356472320
13/06/09 16:26:33 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=4020080640
13/06/09 16:26:33 INFO mapred.JobClient:     Total committed heap usage (bytes)=1102577664
```

Task 3: Viewing the output

Activity Procedure

Step 1: List the files in the HDFS output directory.

```
$ hadoop fs -ls output
```

Step 2: Copy the output HDFS file to local folder to inspect it.

```
$ hadoop fs -get output/part-r-00000 /tmp/inverted.txt
```

Step 3: View the output of your MapReduce job

```
$ less /tmp/inverted.txt
```

Activity Verification

If the job completes as intended, the first few lines of your output will look as follows:

```
177     sample3.txt:1,
1770    sample3.txt:1,
178     sample3.txt:1,
180     sample4.txt:1,
1826    sample3.txt:1,
1838    sample5.txt:1,
1844    sample3.txt:1,
198     sample5.txt:1,
199     sample5.txt:1,
2       sample1.txt:1,
200     sample5.txt:1,
204     sample2.txt:1,
205     sample2.txt:1,
206     sample2.txt:1,
207     sample2.txt:1,
208     sample2.txt:1,
209fallen        sample2.txt:1,
4       sample3.txt:1,
6       sample3.txt:1,
A       sample1.txt:1,sample3.txt:2,sample2.txt:1,
ANDERSEN        sample5.txt:1,sample4.txt:1,sample2.txt:1,
Ages    sample4.txt:1,
Aha     sample5.txt:1,
Alice   sample4.txt:1,
All     sample5.txt:1,sample2.txt:1,
An      sample3.txt:1,
Ancient sample3.txt:1,
And     sample5.txt:11,sample3.txt:2,sample4.txt:1,sample2.txt:22,
Andersen        sample5.txt:2,sample4.txt:3,
As      sample4.txt:1,
```

Lab 6: Write a simple MapReduce job

Description:

- Write a MapReduce program to count the occurrences of distinct words
- Create an input file and copy it to HDFS
- Create a Java file that adds the Java and Hadoop imports, the MapReduce class and sub-classes, **`map()`** and **`reduce()`** methods, and specify other aspects of the job
- Compile and run the program

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

Activity Procedure

Step 1: Go to the location of the source code for the labs and inspect the file that we are going to perform word-count on.

```
$ cd ~/Developer/heffalump

$ ls -l data/humpty.txt
-rw-rw-r--. 1 shrek shrek 142 May 27 10:02 data/humpty.txt

$ cat data/humpty.txt
Humpty Dumpty sat on a wall.
Humpty Dumpty had a great fall.
All the king's horses and all the king's men
Couldn't put Humpty together again!
```

Step 2: Copy the file to HDFS. Clear up the old input directory, create a new one and then copy the file.

```
$ hadoop fs -rm -r input

$ hadoop fs -mkdir input

$ hadoop fs -copyFromLocal -f data/humpty.txt input

$ hadoop fs -ls input
Found 1 items
-rw-r--r--   1 shrek supergroup        142 2013-06-09 22:34 input/humpty.txt
```

Activity Verification

On successful completion you should be able to see the contents of the input file in HDFS.

```
$ hadoop fs -cat input/humpty.txt
Humpty Dumpty sat on a wall.
Humpty Dumpty had a great fall.
All the king's horses and all the king's men
Couldn't put Humpty together again!
```

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

Activity Procedure

Step 1: Open in an editor (`vi` or `emacs`) the Java file for this exercise.

```
$ vi src/main/java/com/hadooptraining/lab6/WordCount.java
```

Step 2: Declare package.

```
package com.hadooptraining.lab6;
```

Step 3: Import necessary libraries.

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
```

Step 4: Start writing the `WordCount` class.

```
public class WordCount {
```

Step 5: Create the mapper class as a subclass of the `WordCount` class. The mapper extends from the `org.apache.hadoop.mapreduce.Mapper` interface. When Hadoop runs, it receives each new line in the input files as an input to the mapper. The "map" function tokenizes the line, and for each token (word) emits (word,1) as the output.
Key 1 = Object
Value 1 = Text
Key 2 = Text - the word itself
Value 2 = IntWritable - the number 1

This class uses two fields.

```
    public static class TokenizerMapper
             extends Mapper<Object, Text, Text, IntWritable>{

        // Create a static variable to store the number 1, (used many times)
        private final static IntWritable one = new IntWritable(1);

        // Create a variable to store the word in the mapper
        private Text word = new Text();
```

Step 6: Write the **`map()`** method that takes each line and emits a key-value pair.

The map function takes the document and splits up the words using blanks as the separator. For each word found in the document, it emits a K2,V2 pair where K2 is the word, and V2 is the number 1.

```
        public void map(Object key, Text value, Context context)
                throws IOException, InterruptedException {

            // Create a String tokenizer to break up the string into parts
            StringTokenizer itr = new StringTokenizer(value.toString());

            // Loop through the words found in the line
            while (itr.hasMoreTokens()) {
                // Set the key as the word itself
                word.set(itr.nextToken());

                // Set the value as number 1, and push it to the context object
                context.write(word, one);
            }
        }
      }
```

Step 7: Create a Reducer class.

```
     public static class IntSumReducer
             extends Reducer<Text, IntWritable, Text, IntWritable> {
         private IntWritable result = new IntWritable();
```

Step 8: Write the **`reduce()`** method. This function takes the keys (word, count) and aggregates the count for each key. Reduce function receives all the values that has the same key as the input, and it outputs the key and the number of occurrences of the key as the output.

```
        public void reduce(Text key, Iterable<IntWritable> values, Context
context
        ) throws IOException, InterruptedException {
            // Create a local variable to store the sum
            int sum = 0;

            // Loop through each value received, and increment sum
            for (IntWritable val : values) {
                sum += val.get();
            }

            // Set the value of the result based on sum
            result.set(sum);

            // Write the result
            context.write(key, result);
        }
    }
```

Step 9: Write the **`main()`** method for the **`WordCount`** program. As input, this program takes any input path in HDFS. It also needs the output path in HDFS where it would write the results.

```
        public static void main(String[] args) throws Exception {
            // This is the configuration object.
            Configuration conf = new Configuration();
            String[] otherArgs = new GenericOptionsParser(conf,
args).getRemainingArgs();

            // If the number of arguments is not correct, print an error message and
exit
            if (otherArgs.length != 2) {
                System.out.println("Usage: wordcount <input_hdfs_dir>
<output_hdfs_dir>");
                System.out.println("Example: wordcount input output");
                System.exit(2);
            }

            // Your job is handled by the Job object - managed by the JobTracker
            Job job = Job.getInstance(conf, "word count");

            // This locates the jar file that needs to be run by using a class name
            job.setJarByClass(WordCount.class);

            // Set the Mapper  class
            job.setMapperClass(TokenizerMapper.class);
            // Set the Reducer class
            job.setReducerClass(IntSumReducer.class);

            // Set the reducer key-value classes
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);

            // Set the input and output paths based on program arguments
            FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
            FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

            // Fire the job and return job status based on success of job
            System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
    }
```

Task 3: Compile and run your MapReduce program

Activity Procedure

Step 1: Compile the program through Maven and create a package.

```
$ cd ~/Developer/heffalump
$ mvn package
```

Step 2: Run the MapReduce job.

```
$ hadoop jar $HOME/Developer/heffalump/target/heffalump-1.0.jar
com.hadooptraining.lab6.WordCount input output
```

Activity Verification

As the job runs, you'll see output on the screen that looks similar to the following:

```
13/06/09 23:07:17 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/06/09 23:07:17 INFO input.FileInputFormat: Total input paths to process : 1
13/06/09 23:07:18 INFO mapred.JobClient: Running job: job_201306092215_0001
13/06/09 23:07:19 INFO mapred.JobClient:  map 0% reduce 0%
13/06/09 23:07:25 INFO mapred.JobClient:  map 100% reduce 0%
13/06/09 23:07:28 INFO mapred.JobClient:  map 100% reduce 100%
13/06/09 23:07:29 INFO mapred.JobClient: Job complete: job_201306092215_0001
13/06/09 23:07:29 INFO mapred.JobClient: Counters: 32
13/06/09 23:07:29 INFO mapred.JobClient:   File System Counters
13/06/09 23:07:29 INFO mapred.JobClient:     FILE: Number of bytes read=304
```

```
13/06/09 23:07:29 INFO mapred.JobClient:     FILE: Number of bytes written=386303
13/06/09 23:07:29 INFO mapred.JobClient:     FILE: Number of read operations=0
13/06/09 23:07:29 INFO mapred.JobClient:     FILE: Number of large read operations=0
13/06/09 23:07:29 INFO mapred.JobClient:     FILE: Number of write operations=0
13/06/09 23:07:29 INFO mapred.JobClient:     HDFS: Number of bytes read=256
13/06/09 23:07:29 INFO mapred.JobClient:     HDFS: Number of bytes written=148
13/06/09 23:07:29 INFO mapred.JobClient:     HDFS: Number of read operations=2
13/06/09 23:07:29 INFO mapred.JobClient:     HDFS: Number of large read operations=0
13/06/09 23:07:29 INFO mapred.JobClient:     HDFS: Number of write operations=1
13/06/09 23:07:29 INFO mapred.JobClient:   Job Counters
13/06/09 23:07:29 INFO mapred.JobClient:     Launched map tasks=1
13/06/09 23:07:29 INFO mapred.JobClient:     Launched reduce tasks=1
13/06/09 23:07:29 INFO mapred.JobClient:     Data-local map tasks=1
13/06/09 23:07:29 INFO mapred.JobClient:     Total time spent by all maps in occupied slots (ms)=5980
13/06/09 23:07:29 INFO mapred.JobClient:     Total time spent by all reduces in occupied slots
(ms)=3234
13/06/09 23:07:29 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving
slots (ms)=0
13/06/09 23:07:29 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving
slots (ms)=0
13/06/09 23:07:29 INFO mapred.JobClient:   Map-Reduce Framework
13/06/09 23:07:29 INFO mapred.JobClient:     Map input records=4
13/06/09 23:07:29 INFO mapred.JobClient:     Map output records=26
13/06/09 23:07:29 INFO mapred.JobClient:     Map output bytes=246
13/06/09 23:07:29 INFO mapred.JobClient:     Input split bytes=114
13/06/09 23:07:29 INFO mapred.JobClient:     Combine input records=0
13/06/09 23:07:29 INFO mapred.JobClient:     Combine output records=0
13/06/09 23:07:29 INFO mapred.JobClient:     Reduce input groups=20
13/06/09 23:07:29 INFO mapred.JobClient:     Reduce shuffle bytes=304
13/06/09 23:07:29 INFO mapred.JobClient:     Reduce input records=26
13/06/09 23:07:29 INFO mapred.JobClient:     Reduce output records=20
13/06/09 23:07:29 INFO mapred.JobClient:     Spilled Records=52
13/06/09 23:07:29 INFO mapred.JobClient:     CPU time spent (ms)=1510
13/06/09 23:07:29 INFO mapred.JobClient:     Physical memory (bytes) snapshot=387842048
13/06/09 23:07:29 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=1542811648
13/06/09 23:07:29 INFO mapred.JobClient:     Total committed heap usage (bytes)=313458688
```

Task 4: Viewing the output

Activity Procedure

Step 1: List the files in the HDFS output directory. You'll find three files there.

```
$ hadoop fs -ls output
Found 3 items
-rw-r--r--   1 shrek supergroup          0 2013-06-09 23:07 output/_SUCCESS
drwxr-xr-x   - shrek supergroup          0 2013-06-09 23:07 output/_logs
```

```
-rw-r--r--   1 shrek supergroup        148 2013-06-09 23:07 output/part-r-00000
```

Step 2: View the output of your MapReduce job

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

Activity Verification

If the job completes as intended, your output will look as follows:

```
All     1
Couldn't        1
Dumpty  2
Humpty  3
a       2
again!  1
all     1
and     1
fall.   1
great   1
had     1
horses  1
king's  2
men     1
on      1
put     1
sat     1
the     2
together        1
wall.   1
```

Task 5: Rewrite WordCount using Tools interface

Activity Procedure

We are going to rewrite the **`WordCount`** main program using the Tools interface. Often Hadoop jobs are executed through a command line. Therefore, each Hadoop job has to support reading, parsing, and processing command-line arguments. To avoid each developer having to rewrite this code, Hadoop provides a **`org.apache.hadoop.util.Tool`** interface. When a job extends from the Tool interface, Hadoop will intercept the command-line arguments, parse the options, and configure the **`JobConf`** object accordingly. Therefore, the job will support standard generic options.

Step 1: Open in an editor (**`vi`** or **`emacs`**) the Java file for this exercise.

```
$ vi src/main/java/com/hadooptraining/lab6/WordCountWithTools.java
```

Step 2: Declare the package.

```
package com.hadooptraining.lab6;
```

Step 3: Import necessary libraries.

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.reduce.IntSumReducer;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

Step 4: We will be using the same Mapper and Reducer classes we wrote earlier. So it is not necessary to write them again. We will only write the class that contains the **main()** function and the **run()** method. First let's write the **run()** method that contains the meat.

```
public class WordCountWithTools extends Configured implements Tool {
    public int run(String[] args) throws Exception {

        // If the number of arguments is insufficient, print error message & exit
        if (args.length < 2) {
            System.out.println("Usage: WordCountWithTools <inDir> <outDir>");
            System.out.println("Example: WordCountWithTools input output");
            ToolRunner.printGenericCommandUsage(System.out);
            System.out.println("");
            return -1;
        }

        // Your job is handled by the Job object - managed by the JobTracker
        Job job = Job.getInstance(getConf(), "Word count with tools");

        // This locates the jar file that needs to be run by using a class name
        job.setJarByClass(WordCount.class);

        // Set the mapper class
```

```
            job.setMapperClass(WordCount.TokenizerMapper.class);

            // Set the reducer class
            job.setReducerClass(IntSumReducer.class);

            // Set the output key class
            job.setOutputKeyClass(Text.class);

            // Set the output value class
            job.setOutputValueClass(IntWritable.class);

            // Add the input and output paths from program arguments
            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));

            // Fire the job and return job status based on success of job
            return job.waitForCompletion(true) ? 0 : 1;
        }
```

Step 5: Now let's write the **`main()`** method that calls the **`run()`** method.

```
    public static void main(String[] args) throws Exception {
            // Invoke the ToolRunner's run method with required arguments
            int res = ToolRunner.run(new Configuration(), new WordCountWithTools(),
    args);

            // Return the same exit code that was returned by ToolRunner.run()
            System.exit(res);
        }
    }
```

Task 6: Compile and run your MapReduce program

Activity Procedure

Step 1: Compile the program through Maven and create a package.

```
$ cd ~/Developer/heffalump
$ mvn package
```

Step 2: Run the MapReduce job.

```
$ hadoop jar $HOME/Developer/heffalump/target/heffalump-1.0.jar
com.hadooptraining.lab6.WordCountWithTools input output
```

Activity Verification

As the job runs, you'll see output on the screen that looks similar to the following:

```
13/06/10 07:48:50 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/06/10 07:48:51 INFO input.FileInputFormat: Total input paths to process : 1
13/06/10 07:48:51 INFO mapred.JobClient: Running job: job_201306092215_0002
13/06/10 07:48:52 INFO mapred.JobClient:  map 0% reduce 0%
13/06/10 07:48:57 INFO mapred.JobClient:  map 100% reduce 0%
13/06/10 07:49:00 INFO mapred.JobClient:  map 100% reduce 100%
13/06/10 07:49:01 INFO mapred.JobClient: Job complete: job_201306092215_0002
13/06/10 07:49:01 INFO mapred.JobClient: Counters: 32
13/06/10 07:49:01 INFO mapred.JobClient:   File System Counters
13/06/10 07:49:01 INFO mapred.JobClient:     FILE: Number of bytes read=304
…
13/06/10 07:49:01 INFO mapred.JobClient:   Job Counters
13/06/10 07:49:01 INFO mapred.JobClient:     Launched map tasks=1
…
13/06/10 07:49:01 INFO mapred.JobClient:     Total time spent by all maps in occupied slots (ms)=5732
13/06/10 07:49:01 INFO mapred.JobClient:     Total time spent by all reduces in occupied slots
(ms)=3155
13/06/10 07:49:01 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving
slots (ms)=0
13/06/10 07:49:01 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving
slots (ms)=0
13/06/10 07:49:01 INFO mapred.JobClient:   Map-Reduce Framework
…
13/06/10 07:49:01 INFO mapred.JobClient:     Spilled Records=52
13/06/10 07:49:01 INFO mapred.JobClient:     CPU time spent (ms)=1510
13/06/10 07:49:01 INFO mapred.JobClient:     Physical memory (bytes) snapshot=406200320
13/06/10 07:49:01 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=1563443200
13/06/10 07:49:01 INFO mapred.JobClient:     Total committed heap usage (bytes)=316866560
```

If the job completes successfully, the output of your MapReduce job will look the same as before.

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

Compare your output with your previous run.

Lab 7: Create a custom Writable Type

Description:

- Create a new Hadoop datatype that can be passed as a value between your **`map()`** and **`reduce()`** code. It can be serialized to disk or transmitted across the network.
- We will analyze log data in this exercise. Each line of the log file will be parsed and selected values from there will be stored in a custom writable object. The key will be set as the IP address found in the line, while the value will be the custom type we build.
- On the reducer side, we'll gather all log lines received for a certain key, and sum up the response size (in bytes) from those values.
- The output should have a list of IP addresses, along with the total number of bytes sent to that IP address.

Task 1: Open the sample input data and copy it to HDFS

Activity Procedure

Step 1: Go to the location of the source code for the labs. Inspect the data file.

```
$ cd ~/Developer/heffalump
$ less data/NASA_access_log.txt
```

Notice the format of the line, e.g.

```
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/
HTTP/1.0" 200 3985
```

It will have the following fields:
`ip_address,<blank_field>,<blank_field>,date,request,status,response_size.`

Your parser must be able to parse these values and set the custom writable object you are about to create.

Type the letter **`q`** to get out of **`less`**.

Step 2: Clean up the input directory in HDFS and copy the data file to HDFS.

```
$ hadoop fs -rm -r input
$ hadoop fs -mkdir input
$ hadoop fs -copyFromLocal -f data/NASA_access_log.txt input
$ hadoop fs -ls input
Found 1 items
-rw-r--r--   1 shrek supergroup    1290399 2013-06-11 07:29 input/NASA_access_log.txt
```

Task 2: Write the custom writable object LogWritable

Most of the code is written for you. You just need to edit the file and add a few key functions.

Activity Procedure

Step 1: Edit the file containing the custom writable object **`LogWritable`**.

```
$ vi src/main/java/com/hadooptraining/lab7/LogWritable.java
```

Fill in the following functions where they are missing:

First write the set function:

```
    public void set (String userIP, String timestamp, String request, int bytes,
int status) {
        this.userIP.set(userIP);
        this.timestamp.set(timestamp);
        this.request.set(request);
        this.responseSize.set(bytes);
        this.status.set(status);
    }
```

Next write the method to read value from a DataInput object.

```
 public void readFields(DataInput in) throws IOException {
        userIP.readFields(in);
        timestamp.readFields(in);
        request.readFields(in);
        responseSize.readFields(in);
        status.readFields(in);
    }
```

Then write the method for writing the record to a DataOutput object.

```
    public void write(DataOutput out) throws IOException {
        userIP.write(out);
        timestamp.write(out);
        request.write(out);
        responseSize.write(out);
        status.write(out);
    }
```

Finally insert a way to calculate hash code for sorting.

```
public int hashCode()
{
    return userIP.hashCode();
}
```

Task 3: Write the Mapper and Reducer class

Activity Procedure

Step 1: Edit the file containing the Mapper and Reducer class. Fill in the missing lines in the **`map()`** function below.

```
$ vi src/main/java/com/hadooptraining/lab7/LogProcessorWithCustomWritable.java
```

Fill in the missing functions by looking up the code below.

```
public static class LogProcessorMap extends
        Mapper<LongWritable, Text, Text, LogWritable> {

    // The following two are the <K,V> pairs for the mapper.
    // We reuse these variables for each call to map() function.
    private Text userHostText = new Text();
    private LogWritable logValue = new LogWritable();

public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {

        // prepare the log pattern string
        String logEntryPattern = "^(\\S+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-
]\\d{4})\\] \"(.+?)\" (\\d{3}) (\\d+)";

        // Compile the pattern and keep it in a local variable
        Pattern p = Pattern.compile(logEntryPattern);
        Matcher matcher = p.matcher(value.toString());

        // if line in log file did not match, get out of the mapper method
        if (!matcher.matches()) {
            return;
        }

        // Set the KEY of the mapper by using one of the extracted values from log
        userHostText.set(matcher.group(1));
```

```
            // Set the VALUE of the mapper by using other extracted values from log
            logValue.set(matcher.group(1), matcher.group(4), matcher.group(5),
Integer.parseInt(matcher.group(7)),
                    Integer.parseInt(matcher.group(6)));

            // Write the key and value to the context object
            context.write(userHostText, logValue);
        }
    }
```

Step 2: Fill in the missing lines in the **`reduce()`** function below.

```
    public static class LogProcessorReduce extends
            Reducer<Text, LogWritable, Text, IntWritable> {
        // Create a common IntWritable object to hold your result
        private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<LogWritable> values,  Context context)
                throws IOException, InterruptedException {

            // Create a local variable to store the sum
            int sum = 0;

            // Iterate through each value received
            for (LogWritable logLine : values) {
                // extract the response size and add it up
                sum += logLine.getResponseSize().get();
            }

            // Set the value of the output as calculated sum
            result.set(sum);

            // Write to the context object
            context.write(key, result);
        }
    }
```

Step 3: Fill in the missing lines in the **`run()`** function below.

```
    public int run(String[] args) throws Exception {
        // If the number of arguments is insufficient, print an error message and exit
        if (args.length < 2) {
            System.err.println("Usage: <input_path> <output_path>");
            System.exit(-1);
        }

        // Your job is handled by the Job object - managed by the JobTracker
        Job job = Job.getInstance(getConf(), "log-analysis");

        // This locates the jar file that needs to be run by using a class name
        job.setJarByClass(LogProcessorWithCustomWritable.class);

        // Set the mapper class
        job.setMapperClass(LogProcessorMap.class);

        // Set the reducer class
        job.setReducerClass(LogProcessorReduce.class);

        // Set the reducer output key class
        job.setOutputKeyClass(Text.class);

        // Set the reducer output value class
        job.setOutputValueClass(IntWritable.class);

        // Set the mapper output key class
        job.setMapOutputKeyClass(Text.class);

        // Set the mapper output value class
        job.setMapOutputValueClass(LogWritable.class);

        // Add the input and output paths from program arguments
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // Fire the job and return job status based on success of job
        return job.waitForCompletion(true) ? 0 : 1;
    }
```

Task 4: Compile and run the MapReduce job

Activity Procedure

Step 1: Compile the program through Maven and create a package.

```
$ cd ~/Developer/heffalump
$ mvn package
```

Step 2: Run the MapReduce job.

```
$ hadoop jar $HOME/Developer/heffalump/target/heffalump-1.0.jar
com.hadooptraining.lab7.LogProcessorWithCustomWritable input output
```

Activity Verification

As the job runs, you'll see output on the screen that looks similar to the following:

```
13/06/11 07:29:55 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/06/11 07:29:56 INFO input.FileInputFormat: Total input paths to process : 1
13/06/11 07:29:56 INFO mapred.JobClient: Running job: job_201306092215_0003
13/06/11 07:29:57 INFO mapred.JobClient:  map 0% reduce 0%
13/06/11 07:30:04 INFO mapred.JobClient:  map 100% reduce 0%
13/06/11 07:30:08 INFO mapred.JobClient:  map 100% reduce 100%
13/06/11 07:30:09 INFO mapred.JobClient: Job complete: job_201306092215_0003
13/06/11 07:30:09 INFO mapred.JobClient: Counters: 32
13/06/11 07:30:09 INFO mapred.JobClient:   File System Counters
13/06/11 07:30:09 INFO mapred.JobClient:     FILE: Number of bytes read=1433238
13/06/11 07:30:09 INFO mapred.JobClient:     FILE: Number of bytes written=3254361
13/06/11 07:30:09 INFO mapred.JobClient:     FILE: Number of read operations=0
13/06/11 07:30:09 INFO mapred.JobClient:     FILE: Number of large read operations=0
13/06/11 07:30:09 INFO mapred.JobClient:     FILE: Number of write operations=0
13/06/11 07:30:09 INFO mapred.JobClient:     HDFS: Number of bytes read=1290522
13/06/11 07:30:09 INFO mapred.JobClient:     HDFS: Number of bytes written=29137
13/06/11 07:30:09 INFO mapred.JobClient:     HDFS: Number of read operations=2
13/06/11 07:30:09 INFO mapred.JobClient:     HDFS: Number of large read operations=0
13/06/11 07:30:09 INFO mapred.JobClient:     HDFS: Number of write operations=1
13/06/11 07:30:09 INFO mapred.JobClient:   Job Counters
13/06/11 07:30:09 INFO mapred.JobClient:     Launched map tasks=1
13/06/11 07:30:09 INFO mapred.JobClient:     Launched reduce tasks=1
13/06/11 07:30:09 INFO mapred.JobClient:     Data-local map tasks=1
13/06/11 07:30:09 INFO mapred.JobClient:     Total time spent by all maps in occupied slots (ms)=7359
13/06/11 07:30:09 INFO mapred.JobClient:     Total time spent by all reduces in occupied slots
(ms)=3957
13/06/11 07:30:09 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving
slots (ms)=0
13/06/11 07:30:09 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving
slots (ms)=0
13/06/11 07:30:09 INFO mapred.JobClient:   Map-Reduce Framework
13/06/11 07:30:09 INFO mapred.JobClient:     Map input records=11676
13/06/11 07:30:09 INFO mapred.JobClient:     Map output records=11552
13/06/11 07:30:09 INFO mapred.JobClient:     Map output bytes=1410014
13/06/11 07:30:09 INFO mapred.JobClient:     Input split bytes=123
13/06/11 07:30:09 INFO mapred.JobClient:     Combine input records=0
13/06/11 07:30:09 INFO mapred.JobClient:     Combine output records=0
```

```
13/06/11 07:30:09 INFO mapred.JobClient:     Reduce input groups=1085
13/06/11 07:30:09 INFO mapred.JobClient:     Reduce shuffle bytes=1433238
13/06/11 07:30:09 INFO mapred.JobClient:     Reduce input records=11552
13/06/11 07:30:09 INFO mapred.JobClient:     Reduce output records=1085
13/06/11 07:30:09 INFO mapred.JobClient:     Spilled Records=23104
13/06/11 07:30:09 INFO mapred.JobClient:     CPU time spent (ms)=3790
13/06/11 07:30:09 INFO mapred.JobClient:     Physical memory (bytes) snapshot=424099840
13/06/11 07:30:09 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=1560547328
13/06/11 07:30:09 INFO mapred.JobClient:     Total committed heap usage (bytes)=321781760
```

Task 5: Viewing the output

Activity Procedure

Step 1: List the files in the HDFS output directory. You'll find three files there.

```
$ hadoop fs -ls output
Found 3 items
-rw-r--r--   1 shrek supergroup          0 2013-06-11 07:30 output/_SUCCESS
drwxr-xr-x   - shrek supergroup          0 2013-06-11 07:29 output/_logs
-rw-r--r--   1 shrek supergroup      29137 2013-06-11 07:30 output/part-r-00000
```

Step 2: View the output of your MapReduce job

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

Activity Verification

If the job completes as intended, your output will look as follows:

```
128.187.140.171    60655
129.188.154.200    1607516
129.193.116.41     46353
129.59.205.2 109598
129.79.164.64      1037740
129.94.144.152     321543
130.161.103.176    143215
130.231.92.40      8677
130.54.33.130      14992
131.112.168.40     142516
...
```

To confirm, type the following and verify the number of lines in the output.

```
$ hadoop fs -cat output/part-r-00000 | wc -l
1085
```

Lab 8: Create a custom InputFormat

Description:

- Write an **`InputFormat`** that can read input files that contain log data
- The real work is in the implementation of interface **`RecordReader`**. We will implement the **`nextKeyValue()`** method of this class.
- 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 - the real work is in understanding how **`RecordReader`** is called and what it is expected to do.
- The goal of this lab is same as the previous lab - to count the number of bytes from each IP address.
- In this lab, we implement an **`InputFormat`** and a **`RecordReader`** for the HTTP log files. This **`InputFormat`** will generate **`LongWritable`** instances as keys and **`LogWritable`** instances as the values.

Task 1: Open the sample input data and copy it to HDFS

Activity Procedure

Step 1: Go to the location of the source code for the labs. Inspect the data file.

```
$ cd ~/Developer/heffalump
$ less data/access.log
```

Notice the format of the line (which is different from the previous exercise), e.g.

```
208.115.111.68 - - [10/Apr/2013:13:47:26 -0400] "GET /events/purchase-event-
tickets HTTP/1.1" 200 56520 "-" "Mozilla/5.0 (compatible; Ezooms/1.0;
ezooms.bot@gmail.com)"
```

It will have the following fields:
ip_address,<blank_field>,<blank_field>,date,request,status,response_size,<blank_field>, response_size,http_agent

Your parser must be able to parse these values and set the custom writable object you are about to create.

Type the letter **q** to get out of **`less`**.

Step 2: Clean up the input directory in HDFS and copy the data file to HDFS.

```
$ hadoop fs -rm -r input
$ hadoop fs -mkdir input
$ hadoop fs -copyFromLocal -f data/access.log input
$ hadoop fs -ls input
Found 1 items
-rw-r--r--   1 shrek supergroup    2907856 2013-06-11 23:41 input/access.log
```

Task 2: Create a FileInputFormat class

LogFileInputFormat extends the **FileInputFormat**, which provides a generic splitting mechanism for HDFS-file based **InputFormat**. We override the **createRecordReader()** method in the **LogFileInputFormat** to provide an instance of our custom **RecordReader** implementation, **LogFileRecordReader**. Optionally, we can also override the **isSplitable()** method of the **FileInputFormat** to control whether the input files are split-up into logical partitions or used as whole files.

Most of the code is written for you. You just need to edit the file and add a few key functions.

Activity Procedure

Step 1: Edit the file containing the input format **LogFileInputFormat**.

```
$ cd ~/Developer/heffalump
$ vi src/main/java/com/hadooptraining/lab8/LogFileInputFormat.java
```

Step 2: Fill in the missing pieces of the file by looking below:

```
public class LogFileInputFormat extends FileInputFormat<LongWritable,
LogWritable> {

@Override
    public RecordReader<LongWritable, LogWritable> createRecordReader(
            InputSplit arg0, TaskAttemptContext arg1)
             throws IOException, InterruptedException {
        return new LogFileRecordReader();
    }
}
```

Task 3: Create a RecordReader class

LogFileRecordReader is the custom input format that reads the log line and parses its values. The **LogFileRecordReader** class extends the **org.apache.hadoop.mapreduce.RecordReader<K,V>** abstract class and uses **LineRecordReader** internally to perform the basic parsing of the input data.

Activity Procedure

Step 1: Edit the file containing the record reader **LogFileRecordReader**.

```
$ vi src/main/java/com/hadooptraining/lab8/LogFileRecordReader.java
```

Step 2: Fill in the missing lines in the **initialize()** function.

```
    @Override
```

```
        public void initialize(InputSplit inputSplit, TaskAttemptContext attempt)
                throws IOException, InterruptedException {
            lineReader = new LineRecordReader();
            lineReader.initialize(inputSplit, attempt);

        }
```

Step 3: Fill in the missing lines in the **nextKeyValue()** function.

```
        @Override
        public boolean nextKeyValue() throws IOException, InterruptedException {
            if (!lineReader.nextKeyValue()) {
                return false;
            }

            // Start parsing the log line by defining the pattern
            String logEntryPattern = "^(\\S+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-
      ]\\d{4})\\] \"(.+?)\" (\\d{3}) (\\d+) \"(\\S+)\" \"(.+?)\"";

            Pattern p = Pattern.compile(logEntryPattern);
            Matcher matcher = p.matcher(lineReader.getCurrentValue().toString());

            // If no match found, flag an error
            if (!matcher.matches()) {
                System.out.println("Bad Record:"+ lineReader.getCurrentValue());
                return nextKeyValue();
            }

            // Assign local variables to the values parsed out of the log line
            String userIP = matcher.group(1);
            String timestamp = matcher.group(4);
            String request = matcher.group(5);
            int status = Integer.parseInt(matcher.group(6));
            int bytes = Integer.parseInt(matcher.group(7));

            // Create a new output value object
            value = new LogWritable();

            // And assign its values
            value.set(userIP, timestamp, request, bytes, status);
            return true;
        }
```

Task 4: Enhance the LogWritable class

You have already seen this class before in the previous lab. This time we are going to use the same class with some enhancements.

Activity Procedure

Step 1: Edit the file containing the record reader **`LogWritable`**.

```
$ vi src/main/java/com/hadooptraining/lab8/LogWritable.java
```

Step 2: Fill in the new function **`compareTo()`**.

```
@Override
    public int compareTo(LogWritable o) {
        if (userIP.compareTo(o.userIP) == 0) {
            return timestamp.compareTo(o.timestamp);
        } else
            return userIP.compareTo(o.userIP);
    }
```

Task 5: Write the Mapper class

In this exercise we have broken down the Mapper and Reducer into separate classes rather than being a sub-class of another class.

Activity Procedure

Step 1: Edit the file containing the Mapper **`LogProcessorMap`**.

```
$ vi src/main/java/com/hadooptraining/lab8/LogProcessorMap.java
```

Step 2: Define a custom counter to count the number of bad or corrupted records in our log processing application.

```
    public static enum LOG_PROCESSOR_COUNTER {
        BAD_RECORDS
    }
```

Step 3: The only difference in this Mapper is that now its incoming value is the **`LogWritable`** object instead of the log line. Parsing is handled by the **`LogFileRecordReader`** class which you have already defined. Fill in the missing lines in the **`map()`** function. Note that we are incrementing the bad record counter for every record that has a response size less than 1.

```
public void map(Object key, LogWritable value, Context context)
        throws IOException, InterruptedException {

    // If the response size is less than 1, we consider it a bad record
    if (value.getResponseSize().get() < 1) {
        context.getCounter(LOG_PROCESSOR_COUNTER.BAD_RECORDS).increment(1);
    }

    // Send the key and value to the context object
    context.write(value.getUserIP(), value.getResponseSize());
}
```

Task 5: Write the Reducer class

The Reducer is pretty much same as before. It just resides in a file of its own.

Activity Procedure

Step 1: Edit the file containing the Reducer **`LogProcessorReduce`**.

```
$ vi src/main/java/com/hadooptraining/lab8/LogProcessorReduce.java
```

Step 2: Fill in the missing parts of the **`reduce()`** function.

```
public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
    // Create a local variable to store the sum
    int sum = 0;

    // Loop through all the values which are IntWritable objects
    for (IntWritable val : values) {
        sum += val.get();
    }

    // Set the output IntWritable object with the sum
    result.set(sum);

    // Write to the context object
    context.write(key, result);
}
```

Task 6: Write the LogProcessorWithCustomInputFormat class

Finally write the main program that constructs the Hadoop job.

Activity Procedure

Step 1: Edit the file containing the main() and run() methods: **`LogProcessorWithCustomInputFormat`**.

```
$ vi src/main/java/com/hadooptraining/lab8/LogProcessorWithCustomInputFormat.java
```

Step 2: Fill in the missing parts of the **`run()`** method.

```
    public int run(String[] args) throws Exception {

        //If the number of arguments is insufficient, print an error message/exit
        if (args.length < 3) {
            System.err.println("Usage:  <input_path> <output_path>
<num_reduce_tasks>");
            System.exit(-1);
        }

        // Your job is handled by the Job object - managed by the JobTracker
        Job job = Job.getInstance(getConf(), "log-analysis");

        // This locates the jar file that needs to be run by using a class name
        job.setJarByClass(LogProcessorWithCustomInputFormat.class);

        // Set the mapper class
        job.setMapperClass(LogProcessorMap.class);

        // Set the reducer class
        job.setReducerClass(LogProcessorReduce.class);

        // Set the reducer output key class
        job.setOutputKeyClass(Text.class);

        // Set the reducer output value class
        job.setOutputValueClass(IntWritable.class);

        // Set the input format class
        job.setInputFormatClass(LogFileInputFormat.class);


        // Add the input and output paths from program arguments
```

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

            // Get the number of reduce tasks from the third argument
            int numReduce = Integer.parseInt(args[2]);

            // Set the number of reduce tasks
            job.setNumReduceTasks(numReduce);

            // Run the job and store the result
            boolean jobResult = job.waitForCompletion(true);

            // Find the counters from the Job object
            Counters counters = job.getCounters();

            // Retrieve the bad records and print it
            Counter badRecords = 
    counters.findCounter(LogProcessorMap.LOG_PROCESSOR_COUNTER.BAD_RECORDS);
            System.out.println("Number of Bad Records: " + badRecords.getValue());

            // Return the status depending on the success of the job
            return jobResult ? 0 : 1;
        }
```

Step 3: Ensure that the **`main()`** method is in place.

```
        public static void main(String[] args) throws Exception {
            // Invoke the ToolRunner's run method with required arguments
            int res = ToolRunner.run(new Configuration(), new 
    LogProcessorWithCustomInputFormat(), args);

            // Return the same exit code that was returned by ToolRunner.run()
            System.exit(res);
        }
```

Task 4: Compile and run the MapReduce job

Activity Procedure

Step 1: Compile the program through Maven and create a package.

```
$ cd ~/Developer/heffalump
$ mvn package
```

Step 2: Run the MapReduce job.

```
$ hadoop jar $HOME/Developer/heffalump/target/heffalump-1.0.jar
com.hadooptraining.lab8.LogProcessorWithCustomInputFormat input output 2
```

Note that we are passing three arguments to this program. The third parameter is the number of reduce tasks.

Activity Verification

As the job runs, you'll see output on the screen that looks similar to the following:

```
13/06/12 09:12:12 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/06/12 09:12:12 INFO input.FileInputFormat: Total input paths to process : 1
13/06/12 09:12:13 INFO mapred.JobClient: Running job: job_201306092215_0005
13/06/12 09:12:14 INFO mapred.JobClient:  map 0% reduce 0%
13/06/12 09:12:21 INFO mapred.JobClient:  map 100% reduce 0%
13/06/12 09:12:25 INFO mapred.JobClient:  map 100% reduce 50%
13/06/12 09:12:26 INFO mapred.JobClient:  map 100% reduce 100%
13/06/12 09:12:26 INFO mapred.JobClient: Job complete: job_201306092215_0005
13/06/12 09:12:26 INFO mapred.JobClient: Counters: 33
13/06/12 09:12:26 INFO mapred.JobClient:   File System Counters
13/06/12 09:12:26 INFO mapred.JobClient:     FILE: Number of bytes read=229721
13/06/12 09:12:26 INFO mapred.JobClient:     FILE: Number of bytes written=1039358
13/06/12 09:12:26 INFO mapred.JobClient:     FILE: Number of read operations=0
13/06/12 09:12:26 INFO mapred.JobClient:     FILE: Number of large read operations=0
13/06/12 09:12:26 INFO mapred.JobClient:     FILE: Number of write operations=0
13/06/12 09:12:26 INFO mapred.JobClient:     HDFS: Number of bytes read=2907970
13/06/12 09:12:26 INFO mapred.JobClient:     HDFS: Number of bytes written=26921
13/06/12 09:12:26 INFO mapred.JobClient:     HDFS: Number of read operations=2
13/06/12 09:12:26 INFO mapred.JobClient:     HDFS: Number of large read operations=0
13/06/12 09:12:26 INFO mapred.JobClient:     HDFS: Number of write operations=2
13/06/12 09:12:26 INFO mapred.JobClient:   Job Counters
13/06/12 09:12:26 INFO mapred.JobClient:     Launched map tasks=1
13/06/12 09:12:26 INFO mapred.JobClient:     Launched reduce tasks=2
13/06/12 09:12:26 INFO mapred.JobClient:     Data-local map tasks=1
13/06/12 09:12:26 INFO mapred.JobClient:     Total time spent by all maps in occupied slots (ms)=7120
13/06/12 09:12:26 INFO mapred.JobClient:     Total time spent by all reduces in occupied slots
(ms)=9096
13/06/12 09:12:26 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving
slots (ms)=0
13/06/12 09:12:26 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving
slots (ms)=0
13/06/12 09:12:26 INFO mapred.JobClient:   Map-Reduce Framework
13/06/12 09:12:26 INFO mapred.JobClient:     Map input records=11516
13/06/12 09:12:26 INFO mapred.JobClient:     Map output records=11516
13/06/12 09:12:26 INFO mapred.JobClient:     Map output bytes=206677
13/06/12 09:12:26 INFO mapred.JobClient:     Input split bytes=114
```

```
13/06/12 09:12:26 INFO mapred.JobClient:     Combine input records=0
13/06/12 09:12:26 INFO mapred.JobClient:     Combine output records=0
13/06/12 09:12:26 INFO mapred.JobClient:     Reduce input groups=1352
13/06/12 09:12:26 INFO mapred.JobClient:     Reduce shuffle bytes=229721
13/06/12 09:12:26 INFO mapred.JobClient:     Reduce input records=11516
13/06/12 09:12:26 INFO mapred.JobClient:     Reduce output records=1352
13/06/12 09:12:26 INFO mapred.JobClient:     Spilled Records=23032
13/06/12 09:12:26 INFO mapred.JobClient:     CPU time spent (ms)=5450
13/06/12 09:12:26 INFO mapred.JobClient:     Physical memory (bytes) snapshot=566046720
13/06/12 09:12:26 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=2329321472
13/06/12 09:12:26 INFO mapred.JobClient:     Total committed heap usage (bytes)=431882240
13/06/12 09:12:26 INFO mapred.JobClient:
com.hadooptraining.lab8.LogProcessorMap$LOG_PROCESSOR_COUNTER
13/06/12 09:12:26 INFO mapred.JobClient:     BAD_RECORDS=30
Number of Bad Records: 30
```

Task 5: Viewing the output

Activity Procedure

Step 1: List the files in the HDFS output directory. You'll find four files there.

```
$ hadoop fs -ls output
Found 4 items
-rw-r--r--   1 shrek supergroup          0 2013-06-12 09:12 output/_SUCCESS
drwxr-xr-x   - shrek supergroup          0 2013-06-12 09:12 output/_logs
-rw-r--r--   1 shrek supergroup      13289 2013-06-12 09:12 output/part-r-00000
-rw-r--r--   1 shrek supergroup      13632 2013-06-12 09:12 output/part-r-00001
```

Step 2: View the output of your MapReduce job. Notice that this time we have two files in the output folder. Why? Because we specified two reducers and each creates a file of its own.

```
$ hadoop fs -cat output/part-r-00000
$ hadoop fs -cat output/part-r-00001
```

Activity Verification

If the job completes as intended, your output will look as follows. Both files will have the same format but different data sets.

```
1.187.209.117      32768
1.202.219.2  0
1.202.219.84 304
1.202.219.86 2407
1.22.116.88  15902
```

```
101.226.168.228    26
101.226.168.231    13
101.226.168.233    51604
...
```

To confirm, type the following and verify the number of lines in the output.

```
$ hadoop fs -cat output/part-r-00000 > /tmp/access.count
$ hadoop fs -cat output/part-r-00001 >> /tmp/access.count
$ wc -l /tmp/access.count
1352
```

Lab 9: Use combiner with MapReduce

Description:

- After running the map function, if there are many key-value pairs with the same key, Hadoop has to move all those values to the reduce function. This can incur a significant overhead. To optimize such scenarios, Hadoop supports a special function called Combiner . If provided, Hadoop will call the combiner from the same node as the map node before invoking the reducer and after running the mapper. This can significantly reduce the amount of data transferred to the reduce step.
- This lab explains how to use the combiner in a stock price analysis program. Since the data is large, using a combiner in such problems makes sense.
- This program uses the NYSE_daily data-set, which has a scheme as follows: *exchange,stock_symbol,date,stock_price_open,stock_price_high,stock_price_low,stock_price_close,stock_volume,stock_price_adj_close*
- It finds the maximum high price for each stock over the course of all the data.

Task 1: Open the sample input data and copy it to HDFS

Activity Procedure

Step 1: Go to the location of the source code for the labs. Inspect the location of data files.

```
$ cd ~/Developer/heffalump
$ ls -l data/NYSEStockData.tar.gz
-rw-rw-r--. 1 shrek shrek 129432738 May 28 09:07 data/NYSEStockData.tar.gz
```

Step 2: Clean up the input directory in HDFS and copy the data file to HDFS.

```
$ hadoop fs -rm -r input
$ hadoop fs -mkdir input
$ tar -xzvf data/NYSEStockData.tar.gz
$ hadoop fs -copyFromLocal -f nyse/* input
$ rm -rf nyse
$ hadoop fs -ls input
Found 26 items
-rw-r--r--   1 shrek supergroup   40990992 2013-06-12 22:32 input/NYSE_daily_prices_A.csv
-rw-r--r--   1 shrek supergroup   32034760 2013-06-12 22:32 input/NYSE_daily_prices_B.csv
-rw-r--r--   1 shrek supergroup   45790256 2013-06-12 22:32 input/NYSE_daily_prices_C.csv
-rw-r--r--   1 shrek supergroup   19234471 2013-06-12 22:32 input/NYSE_daily_prices_D.csv
-rw-r--r--   1 shrek supergroup   22104043 2013-06-12 22:32 input/NYSE_daily_prices_E.csv
-rw-r--r--   1 shrek supergroup   17387253 2013-06-12 22:32 input/NYSE_daily_prices_F.csv
-rw-r--r--   1 shrek supergroup   22608522 2013-06-12 22:32 input/NYSE_daily_prices_G.csv
-rw-r--r--   1 shrek supergroup   23127143 2013-06-12 22:32 input/NYSE_daily_prices_H.csv
-rw-r--r--   1 shrek supergroup   20680033 2013-06-12 22:32 input/NYSE_daily_prices_I.csv
-rw-r--r--   1 shrek supergroup    9537527 2013-06-12 22:32 input/NYSE_daily_prices_J.csv
```

```
-rw-r--r--   1 shrek supergroup   14782892 2013-06-12 22:32 input/NYSE_daily_prices_K.csv
-rw-r--r--   1 shrek supergroup   12958785 2013-06-12 22:32 input/NYSE_daily_prices_L.csv
-rw-r--r--   1 shrek supergroup   38124545 2013-06-12 22:32 input/NYSE_daily_prices_M.csv
-rw-r--r--   1 shrek supergroup   31488945 2013-06-12 22:32 input/NYSE_daily_prices_N.csv
-rw-r--r--   1 shrek supergroup    8865718 2013-06-12 22:32 input/NYSE_daily_prices_O.csv
-rw-r--r--   1 shrek supergroup   31943478 2013-06-12 22:32 input/NYSE_daily_prices_P.csv
-rw-r--r--   1 shrek supergroup     190989 2013-06-12 22:32 input/NYSE_daily_prices_Q.csv
-rw-r--r--   1 shrek supergroup   16808595 2013-06-12 22:32 input/NYSE_daily_prices_R.csv
-rw-r--r--   1 shrek supergroup   31852353 2013-06-12 22:32 input/NYSE_daily_prices_S.csv
-rw-r--r--   1 shrek supergroup   28754690 2013-06-12 22:32 input/NYSE_daily_prices_T.csv
-rw-r--r--   1 shrek supergroup    9951590 2013-06-12 22:32 input/NYSE_daily_prices_U.csv
-rw-r--r--   1 shrek supergroup    9503196 2013-06-12 22:32 input/NYSE_daily_prices_V.csv
-rw-r--r--   1 shrek supergroup   15972013 2013-06-12 22:32 input/NYSE_daily_prices_W.csv
-rw-r--r--   1 shrek supergroup    3613198 2013-06-12 22:32 input/NYSE_daily_prices_X.csv
-rw-r--r--   1 shrek supergroup     686216 2013-06-12 22:32 input/NYSE_daily_prices_Y.csv
-rw-r--r--   1 shrek supergroup    2093424 2013-06-12 22:32 input/NYSE_daily_prices_Z.csv
```

Step 3: Inspect the format of the data.

```
$ hadoop fs -cat input/NYSE_daily_prices_A.csv | head -10
exchange,stock_symbol,date,stock_price_open,stock_price_high,stock_price_low,stoc
k_price_close,stock_volume,stock_price_adj_close
NYSE,AEA,2010-02-08,4.42,4.42,4.21,4.24,205500,4.24
NYSE,AEA,2010-02-05,4.42,4.54,4.22,4.41,194300,4.41
NYSE,AEA,2010-02-04,4.55,4.69,4.39,4.42,233800,4.42
NYSE,AEA,2010-02-03,4.65,4.69,4.50,4.55,182100,4.55
...
```

Notice the format of the line, e.g.

```
NYSE,AEA,2010-02-08,4.42,4.42,4.21,4.24,205500,4.24
```

It will have the following fields: *exchange,stock_symbol,date,stock_price_open,stock_price_high,stock_price_low,stock_price_close,stock_volume,stock_price_adj_close*

Your parser must be able to parse these values to extract the stock symbol and the high price of the day.

Task 2: Create a StockAnalyzer class

We will create a much simpler class here just to demonstrate the partitioner. The **`StockAnalyzer`** class is an all-in-one class that has an embedded Mapper and Reducer.

Most of the code is written for you. You just need to edit the file and add a few key functions.

Activity Procedure

Step 1: Edit the file **`StockPriceAnalyzer.java`** containing the Mapper and Reducer.

```
$ cd ~/Developer/heffalump
$ vi src/main/java/com/hadooptraining/lab9/StockPriceAnalyzer.java
```

Step 2: Fill in the missing pieces of the **map()** method by looking below:

```
public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

    // Convert to a String so we can use the StringTokenizer on it.
    String line = value.toString();

    // Split the line into fields, using comma as the delimiter
    StringTokenizer tokenizer = new StringTokenizer(line, ",");

    // We only care about the 2nd and 5th fields
    String stock_symbol = null;
    String stock_price_high = null;

    // Simple loop to find out the values of interest to us
    for (int i = 0; i < 5 && tokenizer.hasMoreTokens(); i++) {
        switch (i) {
            case 1: // The second field in data line
                stock_symbol = tokenizer.nextToken();
                break;
            case 4: // The fifth field in the data line
                stock_price_high = tokenizer.nextToken();
                break;
            default:
                tokenizer.nextToken();
                break;
        }
    }

    // Exit out of reducer when a bad record is found
    if (stock_symbol == null || stock_price_high == null) {
        // This is a bad record, throw it out and return
        System.err.println("Warning, bad record!");
        return;
    }

    // Discard the schema line at the head of each file
    if (stock_symbol.equals("stock_symbol")) {
```

```
                // Do nothing
            } else {
                // Set the key to be the stock symbol
                stock.set(stock_symbol);

                // Set the value to be the high price of the stock
                FloatWritable high= new FloatWritable(Float.valueOf(stock_price_high));

                // Write the key-value to the context object
                context.write(stock, high);
            }
    }
  }
```

Step 3: Fill in the missing pieces of the **`reduce()`** method by looking below:

```
        public void reduce(Text key, Iterable<FloatWritable> values, Context context)
                throws IOException, InterruptedException {
            // assume prices are never negative
            float max = 0.0f;

            // Loop through the values
            for (FloatWritable price : values) {
                float current = price.get();

                // Set the max value if current value found to be > current max
                if (current > max)
                    max = current;
            }

            // Write the key-value to the context object
            context.write(key, new FloatWritable(max));
        }
```

Step 4: Fill in the missing pieces of the **`main()`** method by looking below. Notice that we set the combiner class to be same as Reducer.

```
      public int run(String[] args) throws Exception {

        // If the number of arguments is insufficient, print an error message and exit
        if (args.length < 2) {
            System.err.println("Usage: <input_path> <output_path>");
            System.exit(-1);
        }

        // Your job is handled by the Job object - managed by the JobTracker
        Job job = Job.getInstance(getConf(), "stock-analysis");

        // This locates the jar file that needs to be run by using a class name
        job.setJarByClass(StockPriceAnalyzer.class);

        // Set the mapper and reducer classes
        job.setMapperClass(StockMapper.class);
        job.setReducerClass(StockReducer.class);

        // Set a combiner for the task
        job.setCombinerClass(StockReducer.class);

        // Set reducer output key and value classes
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(FloatWritable.class);

        // Set mapper output key and value classes
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(FloatWritable.class);

        // Add the input and output paths from program arguments
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // Fire the job and return job status based on success of job
        return job.waitForCompletion(true) ? 0 : 1;
    }
```

Task 3: Compile and run the MapReduce job

Activity Procedure

Step 1: Compile the program through Maven and create a package.

```
$ cd ~/Developer/heffalump
$ mvn package
```

Step 2: Run the MapReduce job.

```
$ hadoop jar $HOME/Developer/heffalump/target/heffalump-1.0.jar
com.hadooptraining.lab9.StockPriceAnalyzer input output
```

Activity Verification

As the job runs, you'll see output on the screen that looks similar to the following:

```
13/06/12 22:33:02 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/06/12 22:33:02 INFO input.FileInputFormat: Total input paths to process : 26
13/06/12 22:33:03 INFO mapred.JobClient: Running job: job_201306092215_0006
13/06/12 22:33:04 INFO mapred.JobClient:  map 0% reduce 0%
13/06/12 22:33:12 INFO mapred.JobClient:  map 4% reduce 0%
13/06/12 22:33:13 INFO mapred.JobClient:  map 8% reduce 0%
13/06/12 22:33:19 INFO mapred.JobClient:  map 15% reduce 0%
13/06/12 22:33:22 INFO mapred.JobClient:  map 15% reduce 5%
13/06/12 22:33:24 INFO mapred.JobClient:  map 23% reduce 5%
13/06/12 22:33:25 INFO mapred.JobClient:  map 23% reduce 8%
13/06/12 22:33:29 INFO mapred.JobClient:  map 31% reduce 8%
13/06/12 22:33:34 INFO mapred.JobClient:  map 38% reduce 12%
13/06/12 22:33:37 INFO mapred.JobClient:  map 38% reduce 13%
13/06/12 22:33:38 INFO mapred.JobClient:  map 42% reduce 13%
13/06/12 22:33:39 INFO mapred.JobClient:  map 46% reduce 13%
13/06/12 22:33:40 INFO mapred.JobClient:  map 46% reduce 15%
13/06/12 22:33:43 INFO mapred.JobClient:  map 54% reduce 17%
13/06/12 22:33:47 INFO mapred.JobClient:  map 62% reduce 18%
13/06/12 22:33:50 INFO mapred.JobClient:  map 62% reduce 21%
13/06/12 22:33:51 INFO mapred.JobClient:  map 69% reduce 21%
13/06/12 22:33:53 INFO mapred.JobClient:  map 69% reduce 23%
13/06/12 22:33:55 INFO mapred.JobClient:  map 77% reduce 23%
13/06/12 22:33:56 INFO mapred.JobClient:  map 77% reduce 24%
13/06/12 22:33:59 INFO mapred.JobClient:  map 85% reduce 26%
13/06/12 22:34:02 INFO mapred.JobClient:  map 88% reduce 28%
13/06/12 22:34:03 INFO mapred.JobClient:  map 92% reduce 28%
13/06/12 22:34:05 INFO mapred.JobClient:  map 92% reduce 31%
13/06/12 22:34:06 INFO mapred.JobClient:  map 100% reduce 31%
13/06/12 22:34:08 INFO mapred.JobClient:  map 100% reduce 100%
13/06/12 22:34:09 INFO mapred.JobClient: Job complete: job_201306092215_0006
13/06/12 22:34:09 INFO mapred.JobClient: Counters: 32
13/06/12 22:34:09 INFO mapred.JobClient:   File System Counters
13/06/12 22:34:09 INFO mapred.JobClient:     FILE: Number of bytes read=54137
13/06/12 22:34:09 INFO mapred.JobClient:     FILE: Number of bytes written=5338292
13/06/12 22:34:09 INFO mapred.JobClient:     FILE: Number of read operations=0
13/06/12 22:34:09 INFO mapred.JobClient:     FILE: Number of large read operations=0
13/06/12 22:34:09 INFO mapred.JobClient:     FILE: Number of write operations=0
13/06/12 22:34:09 INFO mapred.JobClient:     HDFS: Number of bytes read=511088929
13/06/12 22:34:09 INFO mapred.JobClient:     HDFS: Number of bytes written=27922
```

```
13/06/12 22:34:09 INFO mapred.JobClient:     HDFS: Number of read operations=53
13/06/12 22:34:09 INFO mapred.JobClient:     HDFS: Number of large read operations=0
13/06/12 22:34:09 INFO mapred.JobClient:     HDFS: Number of write operations=1
13/06/12 22:34:09 INFO mapred.JobClient:   Job Counters
13/06/12 22:34:09 INFO mapred.JobClient:     Launched map tasks=26
13/06/12 22:34:09 INFO mapred.JobClient:     Launched reduce tasks=1
13/06/12 22:34:09 INFO mapred.JobClient:     Data-local map tasks=26
13/06/12 22:34:09 INFO mapred.JobClient:     Total time spent by all maps in occupied slots
(ms)=115797
13/06/12 22:34:09 INFO mapred.JobClient:     Total time spent by all reduces in occupied slots
(ms)=55492
13/06/12 22:34:09 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving
slots (ms)=0
13/06/12 22:34:09 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving
slots (ms)=0
13/06/12 22:34:09 INFO mapred.JobClient:   Map-Reduce Framework
13/06/12 22:34:09 INFO mapred.JobClient:     Map input records=9211057
13/06/12 22:34:09 INFO mapred.JobClient:     Map output records=9211031
13/06/12 22:34:09 INFO mapred.JobClient:     Map output bytes=72907161
13/06/12 22:34:09 INFO mapred.JobClient:     Input split bytes=3302
13/06/12 22:34:09 INFO mapred.JobClient:     Combine input records=9212372
13/06/12 22:34:09 INFO mapred.JobClient:     Combine output records=4204
13/06/12 22:34:09 INFO mapred.JobClient:     Reduce input groups=2853
13/06/12 22:34:09 INFO mapred.JobClient:     Reduce shuffle bytes=28599
13/06/12 22:34:09 INFO mapred.JobClient:     Reduce input records=2863
13/06/12 22:34:09 INFO mapred.JobClient:     Reduce output records=2853
13/06/12 22:34:09 INFO mapred.JobClient:     Spilled Records=8286
13/06/12 22:34:09 INFO mapred.JobClient:     CPU time spent (ms)=69360
13/06/12 22:34:09 INFO mapred.JobClient:     Physical memory (bytes) snapshot=6774849536
13/06/12 22:34:09 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=20755681280
13/06/12 22:34:09 INFO mapred.JobClient:     Total committed heap usage (bytes)=5331943424
```

Task 4: Run the same job without combiner

Activity Procedure

Step 1: Comment out the line that includes the combiner.

```
// job.setCombinerClass(StockReducer.class);
```

Compile and run the job again. The execution would go somewhat like this.

```
13/06/15 17:58:21 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/06/15 17:58:21 INFO input.FileInputFormat: Total input paths to process : 26
13/06/15 17:58:22 INFO mapred.JobClient: Running job: job_201306092215_0009
13/06/15 17:58:23 INFO mapred.JobClient:  map 0% reduce 0%
13/06/15 17:58:34 INFO mapred.JobClient:  map 8% reduce 0%
13/06/15 17:58:48 INFO mapred.JobClient:  map 15% reduce 0%
```

```
13/06/15 17:58:50 INFO mapred.JobClient:  map 15% reduce 5%
13/06/15 17:58:59 INFO mapred.JobClient:  map 23% reduce 5%
13/06/15 17:59:03 INFO mapred.JobClient:  map 23% reduce 8%
13/06/15 17:59:05 INFO mapred.JobClient:  map 31% reduce 8%
13/06/15 17:59:09 INFO mapred.JobClient:  map 31% reduce 10%
13/06/15 17:59:10 INFO mapred.JobClient:  map 38% reduce 10%
13/06/15 17:59:12 INFO mapred.JobClient:  map 38% reduce 13%
13/06/15 17:59:15 INFO mapred.JobClient:  map 46% reduce 13%
13/06/15 17:59:21 INFO mapred.JobClient:  map 54% reduce 15%
13/06/15 17:59:24 INFO mapred.JobClient:  map 54% reduce 18%
13/06/15 17:59:25 INFO mapred.JobClient:  map 62% reduce 18%
13/06/15 17:59:27 INFO mapred.JobClient:  map 62% reduce 21%
13/06/15 17:59:33 INFO mapred.JobClient:  map 65% reduce 21%
13/06/15 17:59:34 INFO mapred.JobClient:  map 69% reduce 21%
13/06/15 17:59:36 INFO mapred.JobClient:  map 69% reduce 23%
13/06/15 17:59:37 INFO mapred.JobClient:  map 77% reduce 23%
13/06/15 17:59:39 INFO mapred.JobClient:  map 77% reduce 26%
13/06/15 17:59:43 INFO mapred.JobClient:  map 85% reduce 26%
13/06/15 17:59:46 INFO mapred.JobClient:  map 85% reduce 28%
13/06/15 17:59:47 INFO mapred.JobClient:  map 88% reduce 28%
13/06/15 17:59:48 INFO mapred.JobClient:  map 92% reduce 28%
13/06/15 17:59:49 INFO mapred.JobClient:  map 92% reduce 31%
13/06/15 17:59:51 INFO mapred.JobClient:  map 100% reduce 31%
13/06/15 17:59:52 INFO mapred.JobClient:  map 100% reduce 33%
13/06/15 17:59:55 INFO mapred.JobClient:  map 100% reduce 100%
13/06/15 17:59:56 INFO mapred.JobClient: Job complete: job_201306092215_0009
13/06/15 17:59:56 INFO mapred.JobClient: Counters: 32
13/06/15 17:59:56 INFO mapred.JobClient:   File System Counters
13/06/15 17:59:56 INFO mapred.JobClient:     FILE: Number of bytes read=172380218
13/06/15 17:59:56 INFO mapred.JobClient:     FILE: Number of bytes written=268950465
13/06/15 17:59:56 INFO mapred.JobClient:     FILE: Number of read operations=0
13/06/15 17:59:56 INFO mapred.JobClient:     FILE: Number of large read operations=0
13/06/15 17:59:56 INFO mapred.JobClient:     FILE: Number of write operations=0
13/06/15 17:59:56 INFO mapred.JobClient:     HDFS: Number of bytes read=511088929
13/06/15 17:59:56 INFO mapred.JobClient:     HDFS: Number of bytes written=27922
13/06/15 17:59:56 INFO mapred.JobClient:     HDFS: Number of read operations=53
13/06/15 17:59:56 INFO mapred.JobClient:     HDFS: Number of large read operations=0
13/06/15 17:59:56 INFO mapred.JobClient:     HDFS: Number of write operations=1
13/06/15 17:59:56 INFO mapred.JobClient:   Job Counters
13/06/15 17:59:56 INFO mapred.JobClient:     Launched map tasks=26
13/06/15 17:59:56 INFO mapred.JobClient:     Launched reduce tasks=1
13/06/15 17:59:56 INFO mapred.JobClient:     Data-local map tasks=26
13/06/15 17:59:56 INFO mapred.JobClient:     Total time spent by all maps in occupied slots
(ms)=168313
13/06/15 17:59:56 INFO mapred.JobClient:     Total time spent by all reduces in occupied slots
(ms)=81158
13/06/15 17:59:56 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving
slots (ms)=0
```

```
13/06/15 17:59:56 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving
slots (ms)=0
13/06/15 17:59:56 INFO mapred.JobClient:   Map-Reduce Framework
13/06/15 17:59:56 INFO mapred.JobClient:     Map input records=9211057
13/06/15 17:59:56 INFO mapred.JobClient:     Map output records=9211031
13/06/15 17:59:56 INFO mapred.JobClient:     Map output bytes=72907161
13/06/15 17:59:56 INFO mapred.JobClient:     Input split bytes=3302
13/06/15 17:59:56 INFO mapred.JobClient:     Combine input records=0
13/06/15 17:59:56 INFO mapred.JobClient:     Combine output records=0
13/06/15 17:59:56 INFO mapred.JobClient:     Reduce input groups=2853
13/06/15 17:59:56 INFO mapred.JobClient:     Reduce shuffle bytes=91329379
13/06/15 17:59:56 INFO mapred.JobClient:     Reduce input records=9211031
13/06/15 17:59:56 INFO mapred.JobClient:     Reduce output records=2853
13/06/15 17:59:56 INFO mapred.JobClient:     Spilled Records=26597820
13/06/15 17:59:56 INFO mapred.JobClient:     CPU time spent (ms)=77900
13/06/15 17:59:56 INFO mapred.JobClient:     Physical memory (bytes) snapshot=6821113856
13/06/15 17:59:56 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=20781477888
13/06/15 17:59:56 INFO mapred.JobClient:     Total committed heap usage (bytes)=5404033024
```

Notice that the job takes much longer to do compared to your previous lab exercises. Also note that the first run with combiner on this machine takes 1min 7 sec while the second run without combiner takes 1min 35 sec. Look at your output and compare the time for both cases.

Task 5: Viewing the output

Activity Procedure

Step 1: List the files in the HDFS output directory. You'll find three files there.

```
$ hadoop fs -ls output
Found 3 items
-rw-r--r--   1 shrek supergroup          0 2013-06-12 22:34 output/_SUCCESS
drwxr-xr-x   - shrek supergroup          0 2013-06-12 22:33 output/_logs
-rw-r--r--   1 shrek supergroup      27922 2013-06-12 22:34 output/part-r-00000
```

Step 2: View the output of your MapReduce job.

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

Activity Verification

If the job completes as intended, your output will look as follows.

```
AA     94.62
AAI    57.88
AAN    35.21
```

```
AAP   83.65
AAR   25.25
AAV   24.78
AB    94.94
ABA   27.94
ABB   33.39
ABC   84.35
...
```

Lab 10: Create a custom Partitioner

Description:

- Hadoop partitions the intermediate data generated from the Map tasks across the reduce tasks of the computations. A proper partitioning function ensuring balanced load for each reduce task is crucial to the performance of MapReduce computations. Partitioning can also be used to group together related set of records to specific reduce tasks, where you want the certain outputs to be processed or grouped together. Hadoop partitions the intermediate data based on the key space of the intermediate data and decides which reduce task will receive which intermediate record. The sorted set of keys and their values of a partition would be the input for a reduce task. In Hadoop, the total number of partitions should be equal to the number of reduce tasks for the MapReduce computation. Hadoop Partitioners should extend the **`org.apache.hadoop.mapreduce.Partitioner<KEY,VALUE>`** abstract class. Hadoop uses **`org.apache.hadoop.mapreduce.lib.partition.HashPartitioner`** as the default Partitioner for the MapReduce computations. **`HashPartitioner`** partitions the keys based on their **`hashcode(),`** using the formula **`key.hashcode()`** *mod* **`r`**, where **`r`** is the number of reduce tasks. There can be scenarios where our computation logic would require or can be better implemented using an application's specific data-partitioning schema.
- In this lab, we implement a custom Partitioner for our HTTP log processing application, which partitions the keys (IP addresses) based on their geographic regions.
- This lab will be an enhancement to the same log-processing application we did in Lab 8.

Task 1: Open the sample input data and copy it to HDFS

Activity Procedure

Step 1: Clean up the input directory in HDFS and copy the data file to HDFS.

```
$ hadoop fs -rm -r input
$ hadoop fs -mkdir input
$ hadoop fs -copyFromLocal -f data/access.log input
$ hadoop fs -ls input
Found 1 items
-rw-r--r--   1 shrek supergroup    2907856 2013-06-11 23:41 input/access.log
```

Task 2: Create an IPBasedPartitioner class

`IPBasedPartitioner` extends the **`Partioner<KEY,VALE>`**, which defines a **`getPartition()`** method to be overridden by its subclasses.

Activity Procedure

Step 1: Edit the file containing the partitioner **`IPBasedPartitioner`**.

```
$ cd ~/Developer/heffalump
$ vi src/main/java/com/hadooptraining/lab10/IPBasedPartitioner.java
```

Step 2: Fill in the missing pieces of the file by looking below:

```
          public int getPartition(Text ipAddress, IntWritable value,
                   int numPartitions) {
              // Tokenize the IP address by breaking it into its 4 components
              StringTokenizer tokenizer = new StringTokenizer(ipAddress.toString(),
      ".");
              if (tokenizer.hasMoreTokens()) {
                  String token = tokenizer.nextToken();
                  // return a partition Id based on first component of IP address
                  return ((token.hashCode() & Integer.MAX_VALUE) % numPartitions);
              }
              return 0;
          }
```

Task 3: Create the LogProcessor class including the partitioner

You have already written this class before in Lab 8. Here you'll inspect the code below and insert a partitioner in the job.

Activity Procedure

Step 1: Edit the file containing the **main()** and **run()** functions **LogProcessorWithPartitioner**.

```
$ vi src/main/java/com/hadooptraining/lab10/LogProcessorWithPartitioner.java
```

Step 2: Fill in the missing parts of the **run()** method.

```
public int run(String[] args) throws Exception {
    // If the number of arguments is insufficient, print an error message and exit
    if (args.length < 3) {
        System.err.println("Usage: <input_path> <output_path> <num_reduce_tasks>");
        System.exit(-1);
    }

    // Your job is handled by the Job object - managed by the JobTracker
    Job job = Job.getInstance(getConf(), "log-analysis");

    // This locates the jar file that needs to be run by using a class name
    job.setJarByClass(LogProcessorWithPartitioner.class);

    // Set the mapper and reducer classes
    job.setMapperClass(LogProcessorMap.class);
    job.setReducerClass(LogProcessorReduce.class);

    // Set reducer output key and value classes
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
```

```
        // Set the InputFormat class
        job.setInputFormatClass(LogFileInputFormat.class);

        // Configure the partitioner
        job.setPartitionerClass(IPBasedPartitioner.class);

        // Add the input and output paths from program arguments
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // Extract the number of reduce tasks from one of the arguments
        int numReduce = Integer.parseInt(args[2]);

        // Set the number of reduce tasks
        job.setNumReduceTasks(numReduce);

        // Fire the job and return job status based on success of job
        return job.waitForCompletion(true) ? 0 : 1;
    }
```

Task 4: Compile and run the MapReduce job

Activity Procedure

Step 1: Compile the program through Maven and create a package.

```
$ cd ~/Developer/heffalump
$ mvn package
```

Step 2: Run the MapReduce job.

```
$ hadoop jar $HOME/Developer/heffalump/target/heffalump-1.0.jar
com.hadooptraining.lab10.LogProcessorWithPartitioner input output 10
```

Note that we are passing three arguments to this program. The third parameter is the number of reduce tasks.

Activity Verification

As the job runs, you'll see output on the screen that looks similar to the following:

```
13/06/13 06:56:22 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/06/13 06:56:23 INFO input.FileInputFormat: Total input paths to process : 1
13/06/13 06:56:23 INFO mapred.JobClient: Running job: job_201306092215_0007
13/06/13 06:56:24 INFO mapred.JobClient:  map 0% reduce 0%
13/06/13 06:56:30 INFO mapred.JobClient:  map 100% reduce 0%
```

```
13/06/13 06:56:34 INFO mapred.JobClient:  map 100% reduce 10%
13/06/13 06:56:35 INFO mapred.JobClient:  map 100% reduce 20%
13/06/13 06:56:38 INFO mapred.JobClient:  map 100% reduce 30%
13/06/13 06:56:39 INFO mapred.JobClient:  map 100% reduce 40%
13/06/13 06:56:43 INFO mapred.JobClient:  map 100% reduce 60%
13/06/13 06:56:47 INFO mapred.JobClient:  map 100% reduce 80%
13/06/13 06:56:51 INFO mapred.JobClient:  map 100% reduce 90%
13/06/13 06:56:52 INFO mapred.JobClient:  map 100% reduce 100%
13/06/13 06:56:53 INFO mapred.JobClient: Job complete: job_201306092215_0007
13/06/13 06:56:53 INFO mapred.JobClient: Counters: 33
13/06/13 06:56:53 INFO mapred.JobClient:   File System Counters
13/06/13 06:56:53 INFO mapred.JobClient:     FILE: Number of bytes read=229769
13/06/13 06:56:53 INFO mapred.JobClient:     FILE: Number of bytes written=2590825
13/06/13 06:56:53 INFO mapred.JobClient:     FILE: Number of read operations=0
13/06/13 06:56:53 INFO mapred.JobClient:     FILE: Number of large read operations=0
13/06/13 06:56:53 INFO mapred.JobClient:     FILE: Number of write operations=0
13/06/13 06:56:53 INFO mapred.JobClient:     HDFS: Number of bytes read=2907970
13/06/13 06:56:53 INFO mapred.JobClient:     HDFS: Number of bytes written=26921
13/06/13 06:56:53 INFO mapred.JobClient:     HDFS: Number of read operations=2
13/06/13 06:56:53 INFO mapred.JobClient:     HDFS: Number of large read operations=0
13/06/13 06:56:53 INFO mapred.JobClient:     HDFS: Number of write operations=10
13/06/13 06:56:53 INFO mapred.JobClient:   Job Counters
13/06/13 06:56:53 INFO mapred.JobClient:     Launched map tasks=1
13/06/13 06:56:53 INFO mapred.JobClient:     Launched reduce tasks=10
13/06/13 06:56:53 INFO mapred.JobClient:     Data-local map tasks=1
13/06/13 06:56:53 INFO mapred.JobClient:     Total time spent by all maps in occupied slots (ms)=6712
13/06/13 06:56:53 INFO mapred.JobClient:     Total time spent by all reduces in occupied slots
(ms)=42630
13/06/13 06:56:53 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving
slots (ms)=0
13/06/13 06:56:53 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving
slots (ms)=0
13/06/13 06:56:53 INFO mapred.JobClient:   Map-Reduce Framework
13/06/13 06:56:53 INFO mapred.JobClient:     Map input records=11516
13/06/13 06:56:53 INFO mapred.JobClient:     Map output records=11516
13/06/13 06:56:53 INFO mapred.JobClient:     Map output bytes=206677
13/06/13 06:56:53 INFO mapred.JobClient:     Input split bytes=114
13/06/13 06:56:53 INFO mapred.JobClient:     Combine input records=0
13/06/13 06:56:53 INFO mapred.JobClient:     Combine output records=0
13/06/13 06:56:53 INFO mapred.JobClient:     Reduce input groups=1352
13/06/13 06:56:53 INFO mapred.JobClient:     Reduce shuffle bytes=229769
13/06/13 06:56:53 INFO mapred.JobClient:     Reduce input records=11516
13/06/13 06:56:53 INFO mapred.JobClient:     Reduce output records=1352
13/06/13 06:56:53 INFO mapred.JobClient:     Spilled Records=23032
13/06/13 06:56:53 INFO mapred.JobClient:     CPU time spent (ms)=16220
13/06/13 06:56:53 INFO mapred.JobClient:     Physical memory (bytes) snapshot=1775591424
13/06/13 06:56:53 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=8640368640
13/06/13 06:56:53 INFO mapred.JobClient:     Total committed heap usage (bytes)=1295384576
```

```
13/06/13 06:56:53 INFO mapred.JobClient:
com.hadooptraining.lab8.LogProcessorMap$LOG_PROCESSOR_COUNTER
13/06/13 06:56:53 INFO mapred.JobClient:     BAD_RECORDS=30
```

Task 5: Viewing the output

Activity Procedure

Step 1: List the files in the HDFS output directory. You'll find twelve files there this time.

```
$ hadoop fs -ls output
Found 12 items
-rw-r--r--   1 shrek supergroup          0 2013-06-13 06:56 output/_SUCCESS
drwxr-xr-x   - shrek supergroup          0 2013-06-13 06:56 output/_logs
-rw-r--r--   1 shrek supergroup       3833 2013-06-13 06:56 output/part-r-00000
-rw-r--r--   1 shrek supergroup       1482 2013-06-13 06:56 output/part-r-00001
-rw-r--r--   1 shrek supergroup       3049 2013-06-13 06:56 output/part-r-00002
-rw-r--r--   1 shrek supergroup       4279 2013-06-13 06:56 output/part-r-00003
-rw-r--r--   1 shrek supergroup        506 2013-06-13 06:56 output/part-r-00004
-rw-r--r--   1 shrek supergroup       2462 2013-06-13 06:56 output/part-r-00005
-rw-r--r--   1 shrek supergroup       2783 2013-06-13 06:56 output/part-r-00006
-rw-r--r--   1 shrek supergroup       3166 2013-06-13 06:56 output/part-r-00007
-rw-r--r--   1 shrek supergroup       3071 2013-06-13 06:56 output/part-r-00008
-rw-r--r--   1 shrek supergroup       2290 2013-06-13 06:56 output/part-r-00009
```

Step 2: View the output of your MapReduce job.

```
$ hadoop fs -cat output/part-r-00000
$ hadoop fs -cat output/part-r-00001
...
```

Activity Verification

If the job completes as intended, your output will contain the same number of lines as in Lab 8.

```
$ hadoop fs -cat output/part-r-000* | wc -l
1352
```

Lab 11: Use Python for MapReduce

Description:

- Hadoop Streaming feature allows us to use any executable or a script as the mapper or the reducer of a Hadoop MapReduce job.
- Hadoop Streaming enables us to perform rapid prototyping of the MapReduce computations using Linux shell utility programs or using scripting languages.
- Hadoop Streaming also allows the users with some or no Java knowledge to utilize Hadoop to process data stored in HDFS.
- In this lab, we implement a mapper for our HTTP log processing application using Python and use a Hadoop aggregate package based reducer.

Task 1: Clean up HDFS input folder and copy the data file

Activity Procedure

Step 1: Go to the location of the source code for the labs. There is no Java programming involved in this lab. Instead we will be working with a short Python script. The data file for this lab is the same one we used in Lab 7.

```
$ cd ~/Developer/heffalump
$ hadoop fs -rm -r input
$ hadoop fs -mkdir input
$ hadoop fs -copyFromLocal -f data/NASA_access_log.txt input
$ hadoop fs -ls input
Found 1 items
-rw-r--r--   1 shrek supergroup    1290399 2013-06-11 07:29 input/NASA_access_log.txt
```

Task 2: Write a Mapper script

Activity Procedure

Step 1: Write a Python script to analyze lines from the webserver log file and produce key-value pairs.

```
$ vi scripts/logProcessor.py
```

Fill in the missing lines by looking at the script below:

```
#!/usr/bin/python

import sys
import re
```

```
def main(argv):
      regex = re.compile(('(\\S+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\]
\"(.+?)\" (\\d{3}) (\\d+)'))
      line = sys.stdin.readline()
      try:
            while line:
                  fields = regex.match(line)
                  if (fields != None):
                      print "LongValueSum:" + fields.group(1) + "\t" +
fields.group(7);
                  line = sys.stdin.readline();
      except "end of file":
            return None

if __name__ == "__main__":
      main(sys.argv)
```

In our example, we use the Hadoop Aggregate package for the reduction part of our computation. Hadoop aggregate package provides reducer and combiner implementations for simple aggregate operations such as sum, max, unique value count, and histogram. When used with Hadoop Streaming, the mapper outputs must specify the type of aggregation operation of the current computation as a prefix to the output key, which is "**`LongValueSum`**" in our example.

More details may be found at http://hadoop.apache.org/docs/stable/api/org/apache/hadoop/mapred/lib/aggregate/ValueAggregator.html

Task 3: Compile and run the MapReduce job

Activity Procedure

Step 1: Run the MapReduce job.

```
$ cd ~/Developer/heffalump/scripts
$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-
2.0.0-mr1-cdh4.3.0.jar -input input -output output -mapper logProcessor.py -
reducer aggregate -file logProcessor.py
```

Activity Verification

As the job runs, you'll see output on the screen that looks similar to the following:

```
packageJobJar: [logProcessor.py, /tmp/hadoop-shrek/hadoop-unjar7511978391587441854/] []
/tmp/streamjob5661718521908057094.jar tmpDir=null
```

```
13/06/16 18:48:52 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/06/16 18:48:52 INFO mapred.FileInputFormat: Total input paths to process : 1
13/06/16 18:48:52 INFO streaming.StreamJob: getLocalDirs(): [/var/lib/hadoop-
hdfs/cache/shrek/mapred/local]
13/06/16 18:48:52 INFO streaming.StreamJob: Running job: job_201306092215_0017
13/06/16 18:48:52 INFO streaming.StreamJob: To kill this job, run:
13/06/16 18:48:52 INFO streaming.StreamJob: UNDEF/bin/hadoop job  -Dmapred.job.tracker=localhost:8021
-kill job_201306092215_0017
13/06/16 18:48:52 INFO streaming.StreamJob: Tracking URL:
http://localhost:50030/jobdetails.jsp?jobid=job_201306092215_0017
13/06/16 18:48:53 INFO streaming.StreamJob:  map 0%  reduce 0%
13/06/16 18:48:59 INFO streaming.StreamJob:  map 100%  reduce 0%
13/06/16 18:49:02 INFO streaming.StreamJob:  map 100%  reduce 100%
13/06/16 18:49:04 INFO streaming.StreamJob: Job complete: job_201306092215_0017
13/06/16 18:49:04 INFO streaming.StreamJob: Output: output
```

Task 4: Viewing the output

Activity Procedure

Step 1: List the files in the HDFS output directory. You'll find three files there.

```
$ hadoop fs -ls output
Found 3 items
-rw-r--r--   1 shrek supergroup          0 2013-06-16 18:49 output/_SUCCESS
drwxr-xr-x   - shrek supergroup          0 2013-06-16 18:48 output/_logs
-rw-r--r--   1 shrek supergroup      29137 2013-06-16 18:49 output/part-00000
```

Notice that the file naming convention in the output folder is slightly different. This used to be the old format of output files.

Step 2: View the output of your MapReduce job

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

Activity Verification

If the job completes as intended, your output will look as follows:

```
128.187.140.171    60655
129.188.154.200    1607516
129.193.116.41     46353
129.59.205.2 109598
129.79.164.64      1037740
```

```
129.94.144.152     321543
130.161.103.176    143215
130.231.92.40      8677
130.54.33.130      14992
131.112.168.40     142516
131.128.2.155      443162
```

To confirm, type the following and verify the number of lines in the output.

```
$ hadoop fs -cat output/part-00000 | wc -l
1085
```

Lab 12: MapReduce with Distributed Cache

Description:

- We can use the Hadoop **DistributedCache** to distribute read-only file based resources to the Map and Reduce tasks. These resources can be simple data files, archives or JAR files that are needed for the computations performed by the mappers or the reducers.
- In this exercise we will solve the same log processing application as in Lab 10, but instead of using the IP address as the key, we'll use the country. This requires us to lookup the country for every IP address. We'll be using the free GeoIP database for this purpose. The GeoIP database may be downloaded from http://dev.maxmind.com/geoip. The database itself is a data file that needs to be referenced by the mapper. We'll be using the distributed cache to copy the data file over to every machine so that it is available locally for reference.
- When the job completes successfully, you should be able to see the number of bytes downloaded listed by country.

Task 1: Open the sample input data and copy it to HDFS

Activity Procedure

Step 1: Clean up the input directory in HDFS and copy the data file to HDFS.

```
$ hadoop fs -rm -r input
$ hadoop fs -mkdir input
$ hadoop fs -copyFromLocal -f data/access.log input
$ hadoop fs -ls input
Found 1 items
-rw-r--r--   1 shrek supergroup    2907856 2013-06-11 23:41 input/access.log
```

Step 2: Copy the IP lookup data file to HDFS.

```
$ hadoop fs -copyFromLocal -f data/GeoIP.dat
$ hadoop fs -ls
Found 4 items
-rw-r--r--   1 shrek supergroup    1301452 2013-06-15 15:58 GeoIP.dat
drwxr-xr-x   - shrek supergroup          0 2013-06-15 15:58 input
drwxr-xr-x   - shrek supergroup          0 2013-06-04 00:09 merge
drwxr-xr-x   - shrek supergroup          0 2013-06-15 15:58 output
```

Make sure that you see the data file at the top of your HDFS home directory.

Task 2: Create a Mapper using Distributed Cache

LogProcessorMapDistributed is an enhancement to the **LogProcessorMap** that you wrote in Lab 8. In this distributed version, we have a **setup()** and **cleanup()** method to set up and close our country-from-IP lookup object. Secondly, in the **map()** method we use the country as the key instead of the IP address.

Activity Procedure

Step 1: Edit the file containing the mapper **LogProcessorMapDistributed**.

```
$ cd ~/Developer/heffalump
$ vi src/main/java/com/hadooptraining/lab12/LogProcessorMapDistributed.java
```

Step 2: Fill in the missing pieces of the file by looking below. Notice that we have a **setup()** method that initializes the IP lookup object **LookupService**. This object is defined in the Java library found at https://github.com/maxmind/. The cleanup() method closes any file connections after the map() task finishes. The **map()** method itself first looks up the **Country** for the IP address, and then uses that as the key when it writes the key-value pair to the context object. The reducer remains the same as before - which adds the bytes for each key (country in this case).

```
import com.maxmind.geoip.Country;
import com.maxmind.geoip.LookupService;

public class LogProcessorMapDistributed
        extends Mapper<Object, LogWritable, Text, IntWritable> {

    // Store the cache file locally in a Path[] variable
    Path[] localCachePath;

    // Create and keep a lookup object for querying country from IP
    private LookupService lookupService;

public void setup(Context context) throws IOException{
        Configuration conf = context.getConfiguration();

        // Get the local cache file path from the distributed cache
        localCachePath = DistributedCache.getLocalCacheFiles(conf);

        // Create a lookup object to use when resolving IP addresses
        File lookupDbDir = new File(localCachePath[0].toString());

        // Create a lookup object
        lookupService = new LookupService(lookupDbDir,
LookupService.GEOIP_MEMORY_CACHE);
    }

public void cleanup(Context context) throws IOException{
```

```
            // Close the database connection for the lookup service
            lookupService.close();
        }

    public void map(Object key, LogWritable value, Context context)
                throws IOException, InterruptedException {

            // Lookup the country from the user's IP address
            Country country = lookupService.getCountry(value.getUserIP().toString());

            // Write the key-value pair to the context object
            context.write(new Text(country.getName()), value.getResponseSize());
        }
    }
```

Task 3: Create the LogProcessor class using distributed cache

You have already written this class before in Lab 10. Here you'll inspect the code below and insert .

Activity Procedure

Step 1: Edit the file containing the **`main()`** and **`run()`** functions **`LogProcessorDistributed`**.

```
$ vi src/main/java/com/hadooptraining/lab12/LogProcessorDostributed.java
```

Step 2: Fill in the missing parts of the **`run()`** method.

```
        public int run(String[] args) throws Exception {
            // If the number of arguments is insufficient, print error message & exit
            if (args.length < 3) {
                System.err.println("Usage: <input_path> <output_path>
    <num_reduce_tasks>");
                System.exit(-1);
            }

            // Your job is handled by the Job object - managed by the JobTracker
            Job job = Job.getInstance(getConf(), "log-analysis");

            // Add the GeoIP database in job's cache. The data file must be
            // available on HDFS.
            DistributedCache.addCacheFile(new URI("/user/shrek/GeoIP.dat"),
    job.getConfiguration());

            // This locates the jar file that needs to be run by using a class name
            job.setJarByClass(LogProcessorDistributed.class);

            // Set the mapper and reducer classes
            job.setMapperClass(LogProcessorMapDistributed.class);
            job.setReducerClass(LogProcessorReduce.class);

            // Set reducer output key and value classes
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);

            // Set the InputFormat class
            job.setInputFormatClass(LogFileInputFormat.class);

            // Configure the partitioner
            job.setPartitionerClass(IPBasedPartitioner.class);

            // Add the input and output paths from program arguments
            FileInputFormat.setInputPaths(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));

            // Extract the number of reduce tasks from one of the arguments
            int numReduce = Integer.parseInt(args[2]);

            // Set the number of reduce tasks
            job.setNumReduceTasks(numReduce);
            // job.setOutputFormatClass(SequenceFileOutputFormat.class);

            // Fire the job and return job status based on success of job
            return job.waitForCompletion(true) ? 0 : 1;
        }
```

Task 4: Compile and run the MapReduce job

Activity Procedure

Step 1: Compile the program through Maven and create a package.

```
$ cd ~/Developer/heffalump
$ mvn package
```

Step 2: Create a fat jar file containing the GeoIP library. Execute the following set of commands carefully to create a jar file with all dependencies.

```
$ export $DEV_HOME=$HOME/Developer/heffalump
$ pushd /tmp
$ mkdir fatjar
$ cd fatjar
$ jar -xf $DEV_HOME/lib/com/maxmind/geoip-api/1.2.11/*.jar
$ jar -xf $DEV_HOME/target/heffalump-1.0.jar
$ rm $DEV_HOME/target/heffalump-1.0.jar
$ jar -cf $DEV_HOME/target/heffalump-1.0.jar *
$ popd
$ rm -rf /tmp/fatjar
```

Step 3: Run the MapReduce job.

```
$ hadoop jar $HOME/Developer/heffalump/target/heffalump-1.0.jar
com.hadooptraining.lab12.LogProcessorDistributed input output 2
```

Note that again we are passing three arguments to this program. The third parameter is the number of reduce tasks.

Activity Verification

As the job runs, you'll see output on the screen that looks similar to the following:

```
13/06/15 15:58:26 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/06/15 15:58:26 INFO input.FileInputFormat: Total input paths to process : 1
13/06/15 15:58:26 INFO mapred.JobClient: Running job: job_201306092215_0008
13/06/15 15:58:28 INFO mapred.JobClient:  map 0% reduce 0%
13/06/15 15:58:34 INFO mapred.JobClient:  map 100% reduce 0%
13/06/15 15:58:38 INFO mapred.JobClient:  map 100% reduce 50%
13/06/15 15:58:39 INFO mapred.JobClient:  map 100% reduce 100%
13/06/15 15:58:40 INFO mapred.JobClient: Job complete: job_201306092215_0008
13/06/15 15:58:40 INFO mapred.JobClient: Counters: 32
13/06/15 15:58:40 INFO mapred.JobClient:   File System Counters
13/06/15 15:58:40 INFO mapred.JobClient:     FILE: Number of bytes read=190479
13/06/15 15:58:40 INFO mapred.JobClient:     FILE: Number of bytes written=970087
13/06/15 15:58:40 INFO mapred.JobClient:     FILE: Number of read operations=0
13/06/15 15:58:40 INFO mapred.JobClient:     FILE: Number of large read operations=0
13/06/15 15:58:40 INFO mapred.JobClient:     FILE: Number of write operations=0
13/06/15 15:58:40 INFO mapred.JobClient:     HDFS: Number of bytes read=2907970
13/06/15 15:58:40 INFO mapred.JobClient:     HDFS: Number of bytes written=987
13/06/15 15:58:40 INFO mapred.JobClient:     HDFS: Number of read operations=2
13/06/15 15:58:40 INFO mapred.JobClient:     HDFS: Number of large read operations=0
13/06/15 15:58:40 INFO mapred.JobClient:     HDFS: Number of write operations=2
13/06/15 15:58:40 INFO mapred.JobClient:   Job Counters
13/06/15 15:58:40 INFO mapred.JobClient:     Launched map tasks=1
13/06/15 15:58:40 INFO mapred.JobClient:     Launched reduce tasks=2
13/06/15 15:58:40 INFO mapred.JobClient:     Data-local map tasks=1
13/06/15 15:58:40 INFO mapred.JobClient:     Total time spent by all maps in occupied slots (ms)=6953
13/06/15 15:58:40 INFO mapred.JobClient:     Total time spent by all reduces in occupied slots
(ms)=7793
13/06/15 15:58:40 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving
slots (ms)=0
13/06/15 15:58:40 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving
slots (ms)=0
13/06/15 15:58:40 INFO mapred.JobClient:   Map-Reduce Framework
13/06/15 15:58:40 INFO mapred.JobClient:     Map input records=11516
13/06/15 15:58:40 INFO mapred.JobClient:     Map output records=11516
```

```
13/06/15 15:58:40 INFO mapred.JobClient:     Map output bytes=167435
13/06/15 15:58:40 INFO mapred.JobClient:     Input split bytes=114
13/06/15 15:58:40 INFO mapred.JobClient:     Combine input records=0
13/06/15 15:58:40 INFO mapred.JobClient:     Combine output records=0
13/06/15 15:58:40 INFO mapred.JobClient:     Reduce input groups=61
13/06/15 15:58:40 INFO mapred.JobClient:     Reduce shuffle bytes=190479
13/06/15 15:58:40 INFO mapred.JobClient:     Reduce input records=11516
13/06/15 15:58:40 INFO mapred.JobClient:     Reduce output records=61
13/06/15 15:58:40 INFO mapred.JobClient:     Spilled Records=23032
13/06/15 15:58:40 INFO mapred.JobClient:     CPU time spent (ms)=5280
13/06/15 15:58:40 INFO mapred.JobClient:     Physical memory (bytes) snapshot=578961408
13/06/15 15:58:40 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=2352939008
13/06/15 15:58:40 INFO mapred.JobClient:     Total committed heap usage (bytes)=435552256.
```

Task 5: Viewing the output

Activity Procedure

Step 1: List the files in the HDFS output directory. You'll find four files there.

```
$ hadoop fs -ls output
Found 4 items
-rw-r--r--   1 shrek supergroup          0 2013-06-15 15:58 output/_SUCCESS
drwxr-xr-x   - shrek supergroup          0 2013-06-15 15:58 output/_logs
-rw-r--r--   1 shrek supergroup        395 2013-06-15 15:58 output/part-r-00000
-rw-r--r--   1 shrek supergroup        592 2013-06-15 15:58 output/part-r-00001
```

Step 2: View the output of your MapReduce job. You'll see that the number of bytes downloaded are listed by country.

```
$ hadoop fs -cat output/part-r-00000
Australia	438893
Brazil	71734
Canada	1133384
Denmark	1390
Europe	598
Hungary	1469880
Indonesia	244708
Israel	133797
Japan	3398749
Kazakhstan	1402
Luxembourg	181256
Mongolia	5253
Nepal	309086
Netherlands	273561
```

```
Norway	30426
Poland	1350348
Russian Federation	3270010
Saudi Arabia	1386635
Serbia	102811
Singapore	8608
Slovakia	30426
Sweden	1733474
Taiwan	30426
Turkey	162007
United Kingdom	1059516

$ hadoop fs -cat output/part-r-00001
Anonymous Proxy	756077
Austria	138989
Bangladesh	5693168
Belgium	85356
Chile	219628
China	38194761
Czech Republic	11120
Egypt	22780
France	3896578
Germany	7663032
Ghana	225127
Greece	1390
Hong Kong	1390
India	18386185
Ireland	342802
Italy	134054
Korea, Republic of	1243
Kuwait	22780
Latvia	1026405
Lithuania	1070993
Malaysia	566034
Mexico	203454
Moldova, Republic of	252577
New Zealand	5253
Oman	29093
Pakistan	230061
Philippines	60852
Qatar	25272
Romania	6950
```

```
Sri Lanka	128801
Sudan	1868569
Switzerland	3056
Thailand	34614
Ukraine	3011672
United Arab Emirates	165837
United States	119941503
```

Lab 13: Monitor a Job with JobTracker

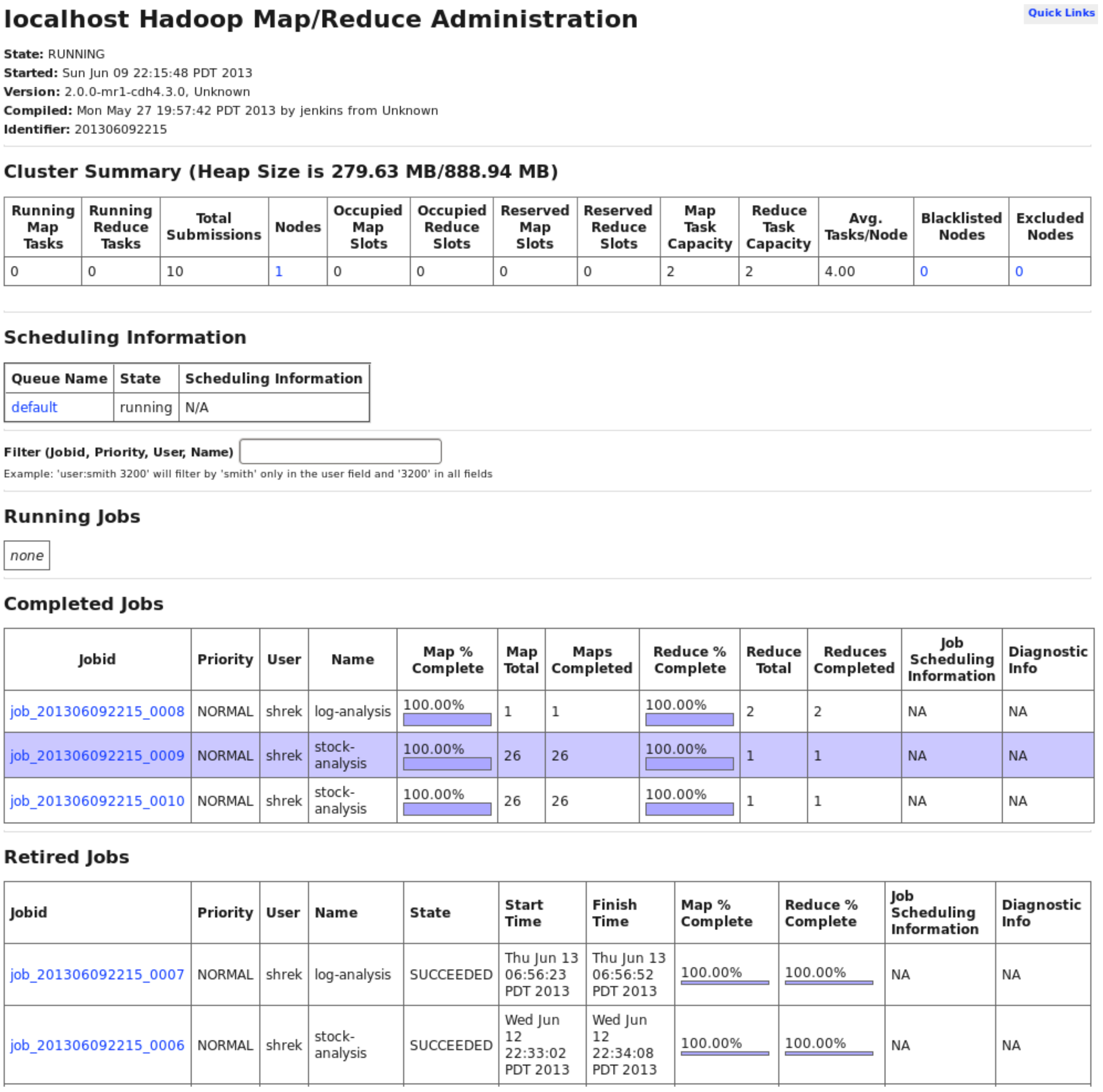
Description:

- Running a MapReduce job and analyze statistics in JobTracker
- Run a Java MapReduce program
- Use JobTracker GUI to analyze the job

Task 1: Run a MapReduce job and monitor it during execution

Activity Procedure

Step 1: Open your browser and go to http://localhost:50030 . If this URL does not work, find out your hostname where the MapReduce jobs are running, and type the full host name instead of *localhost*. You will see something similar to the following:



localhost Hadoop Map/Reduce Administration

Quick Links

State: RUNNING
Started: Sun Jun 09 22:15:48 PDT 2013
Version: 2.0.0-mr1-cdh4.3.0, Unknown
Compiled: Mon May 27 19:57:42 PDT 2013 by jenkins from Unknown
Identifier: 201306092215

Cluster Summary (Heap Size is 279.63 MB/888.94 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	Excluded Nodes
0	0	10	1	0	0	0	0	2	2	4.00	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

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

Running Jobs

none

Completed Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201306092215_0008	NORMAL	shrek	log-analysis	100.00%	1	1	100.00%	2	2	NA	NA
job_201306092215_0009	NORMAL	shrek	stock-analysis	100.00%	26	26	100.00%	1	1	NA	NA
job_201306092215_0010	NORMAL	shrek	stock-analysis	100.00%	26	26	100.00%	1	1	NA	NA

Retired Jobs

Jobid	Priority	User	Name	State	Start Time	Finish Time	Map % Complete	Reduce % Complete	Job Scheduling Information	Diagnostic Info
job_201306092215_0007	NORMAL	shrek	log-analysis	SUCCEEDED	Thu Jun 13 06:56:23 PDT 2013	Thu Jun 13 06:56:52 PDT 2013	100.00%	100.00%	NA	NA
job_201306092215_0006	NORMAL	shrek	stock-analysis	SUCCEEDED	Wed Jun 12 22:33:02 PDT 2013	Wed Jun 12 22:34:08 PDT 2013	100.00%	100.00%	NA	NA

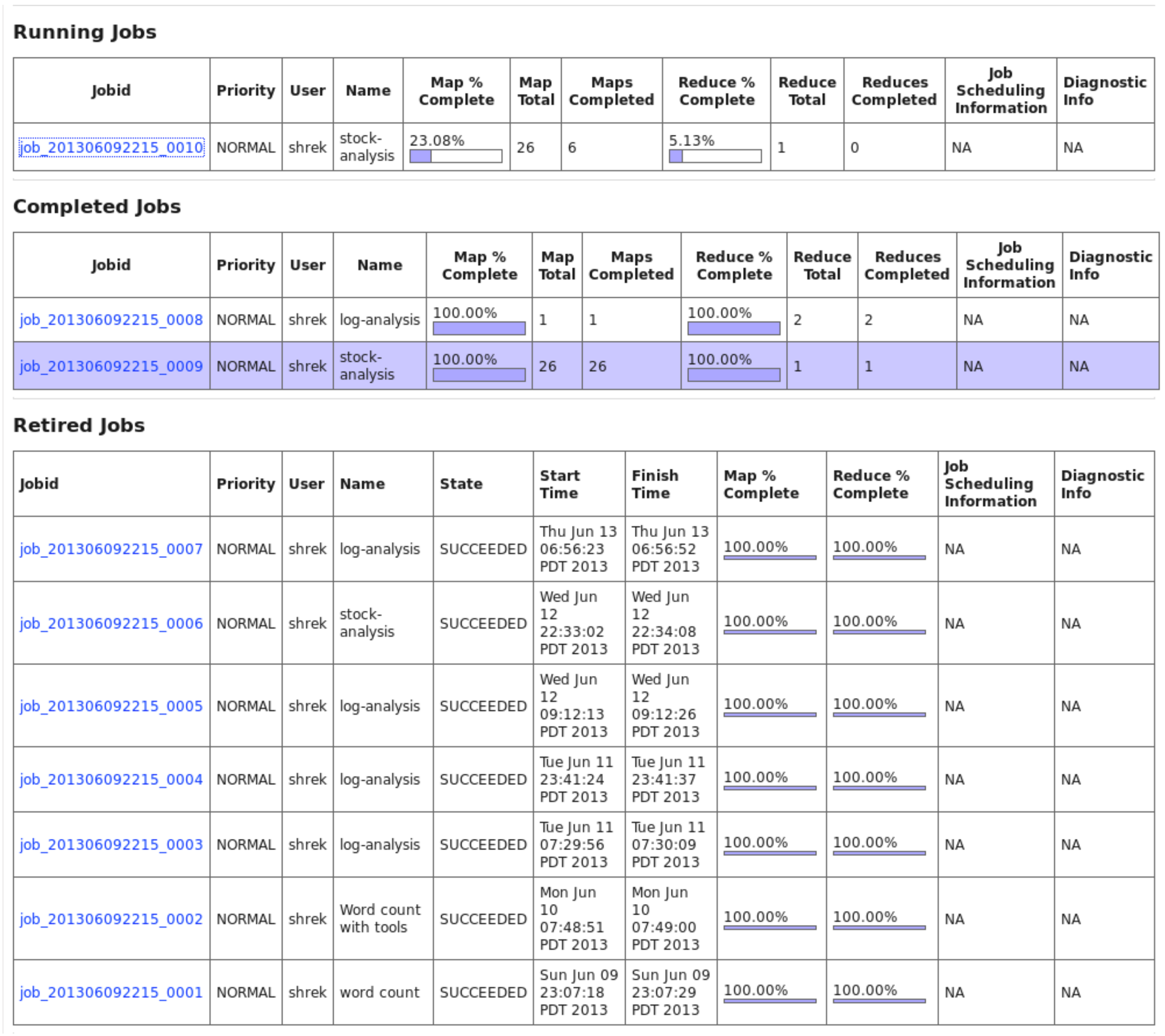
Notice that the jobs are categorized under three sections: Running jobs, Completed jobs and Retired jobs.

Let us now run a MapReduce job and monitor it while it is running.

Step 2: Run a MapReduce job. Type the following and immediately, go and refresh the Job Tracker page on your browser.

```
$ hadoop jar $HOME/Developer/heffalump/target/heffalump-1.0.jar
com.hadooptraining.lab9.StockPriceAnalyzer input output
```

While the job is running, you'll see an entry under running jobs:



Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201306092215_0010	NORMAL	shrek	stock-analysis	23.08%	26	6	5.13%	1	0	NA	NA

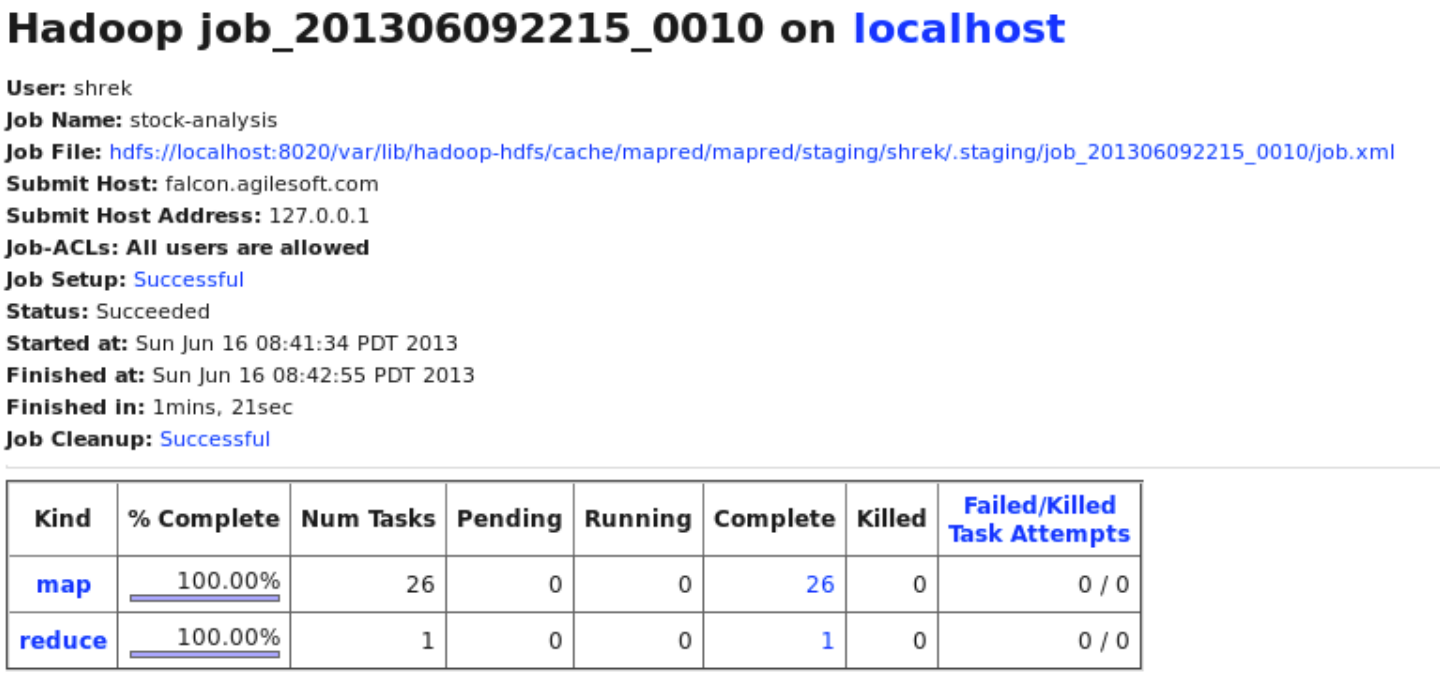
Completed Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201306092215_0008	NORMAL	shrek	log-analysis	100.00%	1	1	100.00%	2	2	NA	NA
job_201306092215_0009	NORMAL	shrek	stock-analysis	100.00%	26	26	100.00%	1	1	NA	NA

Retired Jobs

Jobid	Priority	User	Name	State	Start Time	Finish Time	Map % Complete	Reduce % Complete	Job Scheduling Information	Diagnostic Info
job_201306092215_0007	NORMAL	shrek	log-analysis	SUCCEEDED	Thu Jun 13 06:56:23 PDT 2013	Thu Jun 13 06:56:52 PDT 2013	100.00%	100.00%	NA	NA
job_201306092215_0006	NORMAL	shrek	stock-analysis	SUCCEEDED	Wed Jun 12 22:33:02 PDT 2013	Wed Jun 12 22:34:08 PDT 2013	100.00%	100.00%	NA	NA
job_201306092215_0005	NORMAL	shrek	log-analysis	SUCCEEDED	Wed Jun 12 09:12:13 PDT 2013	Wed Jun 12 09:12:26 PDT 2013	100.00%	100.00%	NA	NA
job_201306092215_0004	NORMAL	shrek	log-analysis	SUCCEEDED	Tue Jun 11 23:41:24 PDT 2013	Tue Jun 11 23:41:37 PDT 2013	100.00%	100.00%	NA	NA
job_201306092215_0003	NORMAL	shrek	log-analysis	SUCCEEDED	Tue Jun 11 07:29:56 PDT 2013	Tue Jun 11 07:30:09 PDT 2013	100.00%	100.00%	NA	NA
job_201306092215_0002	NORMAL	shrek	Word count with tools	SUCCEEDED	Mon Jun 10 07:48:51 PDT 2013	Mon Jun 10 07:49:00 PDT 2013	100.00%	100.00%	NA	NA
job_201306092215_0001	NORMAL	shrek	word count	SUCCEEDED	Sun Jun 09 23:07:18 PDT 2013	Sun Jun 09 23:07:29 PDT 2013	100.00%	100.00%	NA	NA

Step 3: Click the link under Running jobs, to find out more details about the job.

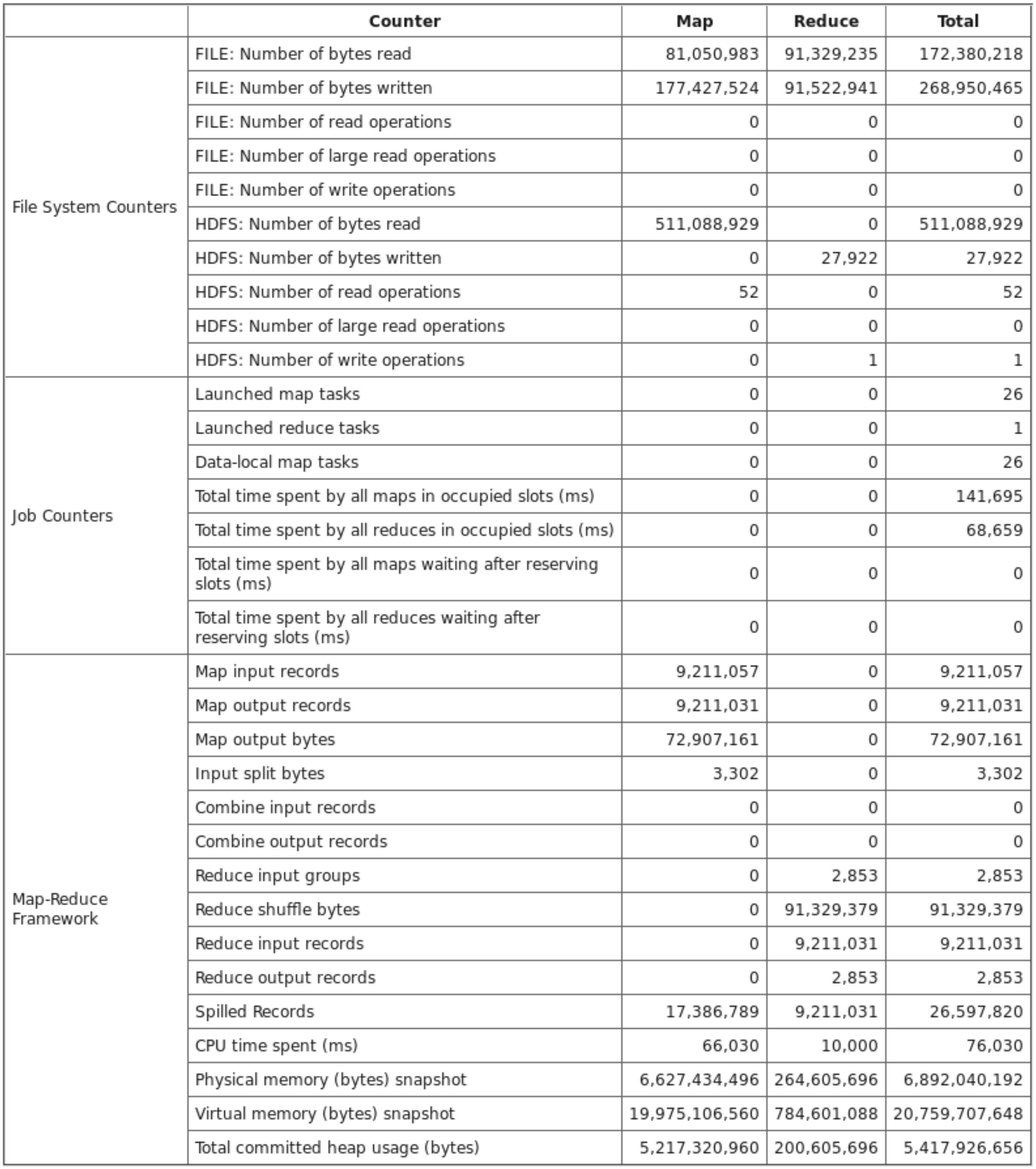


Hadoop job_201306092215_0010 on localhost

User: shrek
Job Name: stock-analysis
Job File: hdfs://localhost:8020/var/lib/hadoop-hdfs/cache/mapred/mapred/staging/shrek/.staging/job_201306092215_0010/job.xml
Submit Host: falcon.agilesoft.com
Submit Host Address: 127.0.0.1
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Succeeded
Started at: Sun Jun 16 08:41:34 PDT 2013
Finished at: Sun Jun 16 08:42:55 PDT 2013
Finished in: 1mins, 21sec
Job Cleanup: Successful

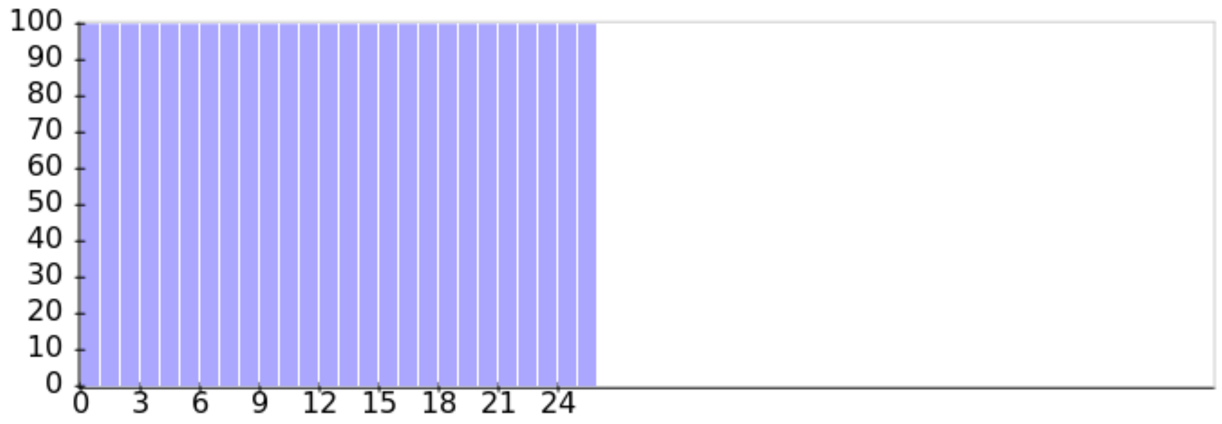
Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	26	0	0	26	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

Click on the map and reduce links while the job is running. These links will not work once the job is over. Also the percentage of the map and reduce tasks changes rapidly, so you need to capture it while the job is in progress.

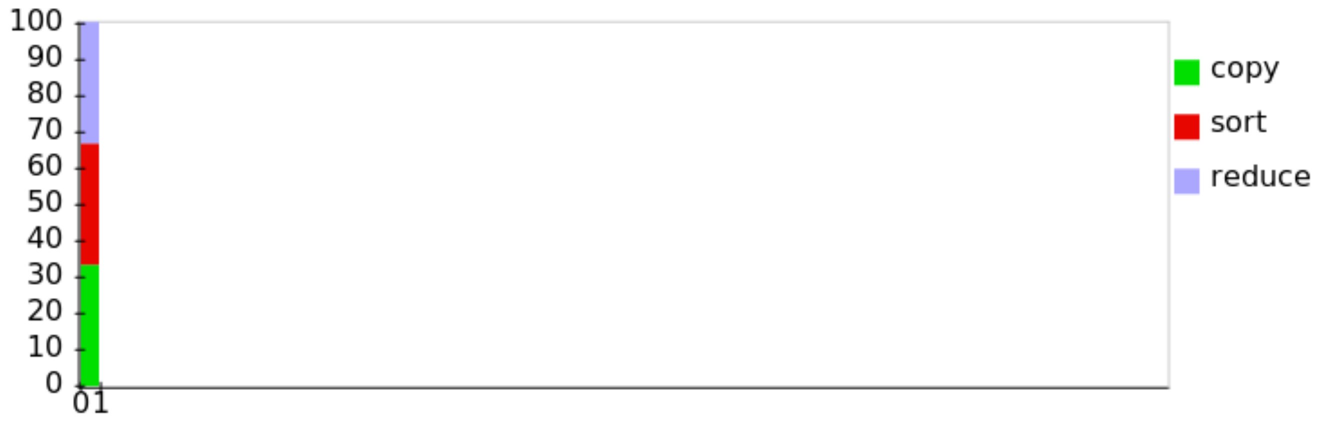


	Counter	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	81,050,983	91,329,235	172,380,218
	FILE: Number of bytes written	177,427,524	91,522,941	268,950,465
	FILE: Number of read operations	0	0	0
	FILE: Number of large read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	511,088,929	0	511,088,929
	HDFS: Number of bytes written	0	27,922	27,922
	HDFS: Number of read operations	52	0	52
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of write operations	0	1	1
Job Counters	Launched map tasks	0	0	26
	Launched reduce tasks	0	0	1
	Data-local map tasks	0	0	26
	Total time spent by all maps in occupied slots (ms)	0	0	141,695
	Total time spent by all reduces in occupied slots (ms)	0	0	68,659
	Total time spent by all maps waiting after reserving slots (ms)	0	0	0
	Total time spent by all reduces waiting after reserving slots (ms)	0	0	0
Map-Reduce Framework	Map input records	9,211,057	0	9,211,057
	Map output records	9,211,031	0	9,211,031
	Map output bytes	72,907,161	0	72,907,161
	Input split bytes	3,302	0	3,302
	Combine input records	0	0	0
	Combine output records	0	0	0
	Reduce input groups	0	2,853	2,853
	Reduce shuffle bytes	0	91,329,379	91,329,379
	Reduce input records	0	9,211,031	9,211,031
	Reduce output records	0	2,853	2,853
	Spilled Records	17,386,789	9,211,031	26,597,820
	CPU time spent (ms)	66,030	10,000	76,030
	Physical memory (bytes) snapshot	6,627,434,496	264,605,696	6,892,040,192
	Virtual memory (bytes) snapshot	19,975,106,560	784,601,088	20,759,707,648
	Total committed heap usage (bytes)	5,217,320,960	200,605,696	5,417,926,656

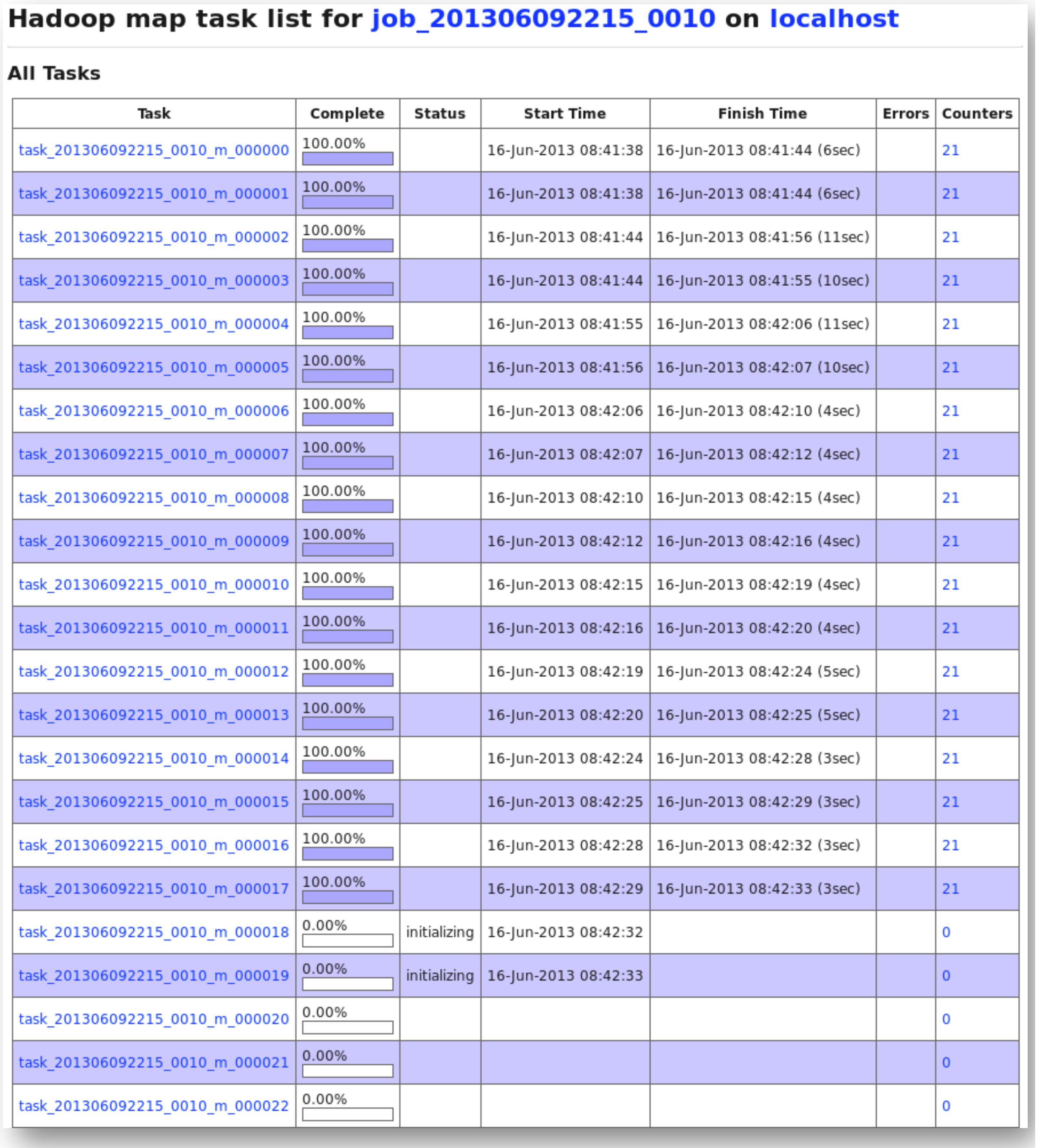
Map Completion Graph - close



Reduce Completion Graph - close



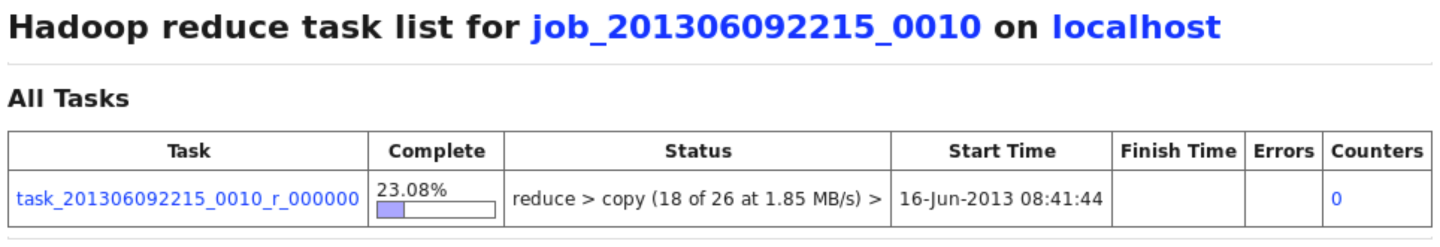
Step 4: Analyze the map task and the reduce task by clicking on the blue link **map** and **reduce** on the job detail page.



Hadoop map task list for job_201306092215_0010 on localhost

All Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
task_201306092215_0010_m_000000	100.00%		16-Jun-2013 08:41:38	16-Jun-2013 08:41:44 (6sec)		21
task_201306092215_0010_m_000001	100.00%		16-Jun-2013 08:41:38	16-Jun-2013 08:41:44 (6sec)		21
task_201306092215_0010_m_000002	100.00%		16-Jun-2013 08:41:44	16-Jun-2013 08:41:56 (11sec)		21
task_201306092215_0010_m_000003	100.00%		16-Jun-2013 08:41:44	16-Jun-2013 08:41:55 (10sec)		21
task_201306092215_0010_m_000004	100.00%		16-Jun-2013 08:41:55	16-Jun-2013 08:42:06 (11sec)		21
task_201306092215_0010_m_000005	100.00%		16-Jun-2013 08:41:56	16-Jun-2013 08:42:07 (10sec)		21
task_201306092215_0010_m_000006	100.00%		16-Jun-2013 08:42:06	16-Jun-2013 08:42:10 (4sec)		21
task_201306092215_0010_m_000007	100.00%		16-Jun-2013 08:42:07	16-Jun-2013 08:42:12 (4sec)		21
task_201306092215_0010_m_000008	100.00%		16-Jun-2013 08:42:10	16-Jun-2013 08:42:15 (4sec)		21
task_201306092215_0010_m_000009	100.00%		16-Jun-2013 08:42:12	16-Jun-2013 08:42:16 (4sec)		21
task_201306092215_0010_m_000010	100.00%		16-Jun-2013 08:42:15	16-Jun-2013 08:42:19 (4sec)		21
task_201306092215_0010_m_000011	100.00%		16-Jun-2013 08:42:16	16-Jun-2013 08:42:20 (4sec)		21
task_201306092215_0010_m_000012	100.00%		16-Jun-2013 08:42:19	16-Jun-2013 08:42:24 (5sec)		21
task_201306092215_0010_m_000013	100.00%		16-Jun-2013 08:42:20	16-Jun-2013 08:42:25 (5sec)		21
task_201306092215_0010_m_000014	100.00%		16-Jun-2013 08:42:24	16-Jun-2013 08:42:28 (3sec)		21
task_201306092215_0010_m_000015	100.00%		16-Jun-2013 08:42:25	16-Jun-2013 08:42:29 (3sec)		21
task_201306092215_0010_m_000016	100.00%		16-Jun-2013 08:42:28	16-Jun-2013 08:42:32 (3sec)		21
task_201306092215_0010_m_000017	100.00%		16-Jun-2013 08:42:29	16-Jun-2013 08:42:33 (3sec)		21
task_201306092215_0010_m_000018	0.00%	initializing	16-Jun-2013 08:42:32			0
task_201306092215_0010_m_000019	0.00%	initializing	16-Jun-2013 08:42:33			0
task_201306092215_0010_m_000020	0.00%					0
task_201306092215_0010_m_000021	0.00%					0
task_201306092215_0010_m_000022	0.00%					0

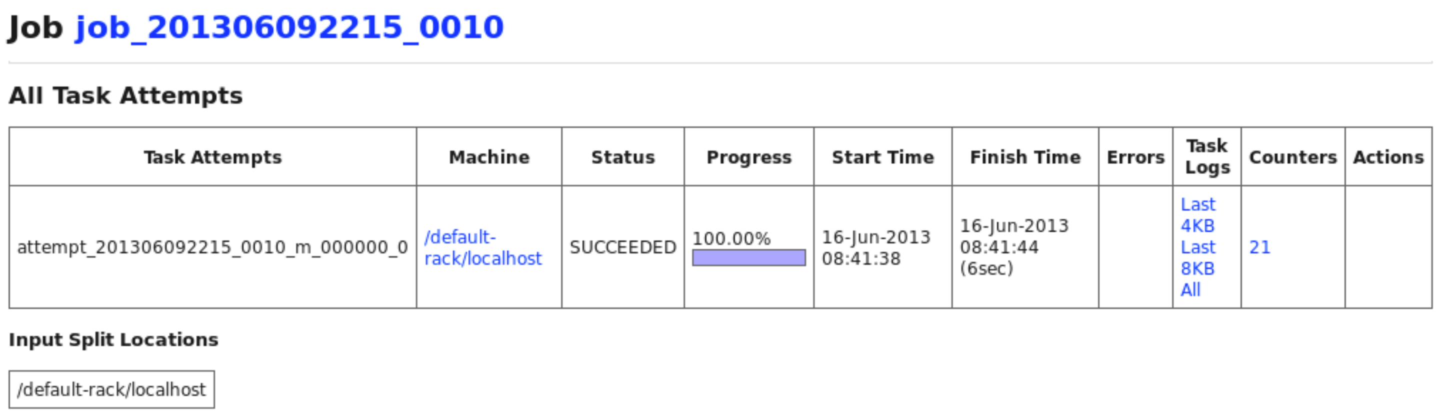


Hadoop reduce task list for job_201306092215_0010 on localhost

All Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
task_201306092215_0010_r_000000	23.08%	reduce > copy (18 of 26 at 1.85 MB/s) >	16-Jun-2013 08:41:44			0

Clicking on the task id, provides more details.



Job job_201306092215_0010

All Task Attempts

Task Attempts	Machine	Status	Progress	Start Time	Finish Time	Errors	Task Logs	Counters	Actions
attempt_201306092215_0010_m_000000_0	/default-rack/localhost	SUCCEEDED	100.00%	16-Jun-2013 08:41:38	16-Jun-2013 08:41:44 (6sec)		Last 4KB Last 8KB All	21	

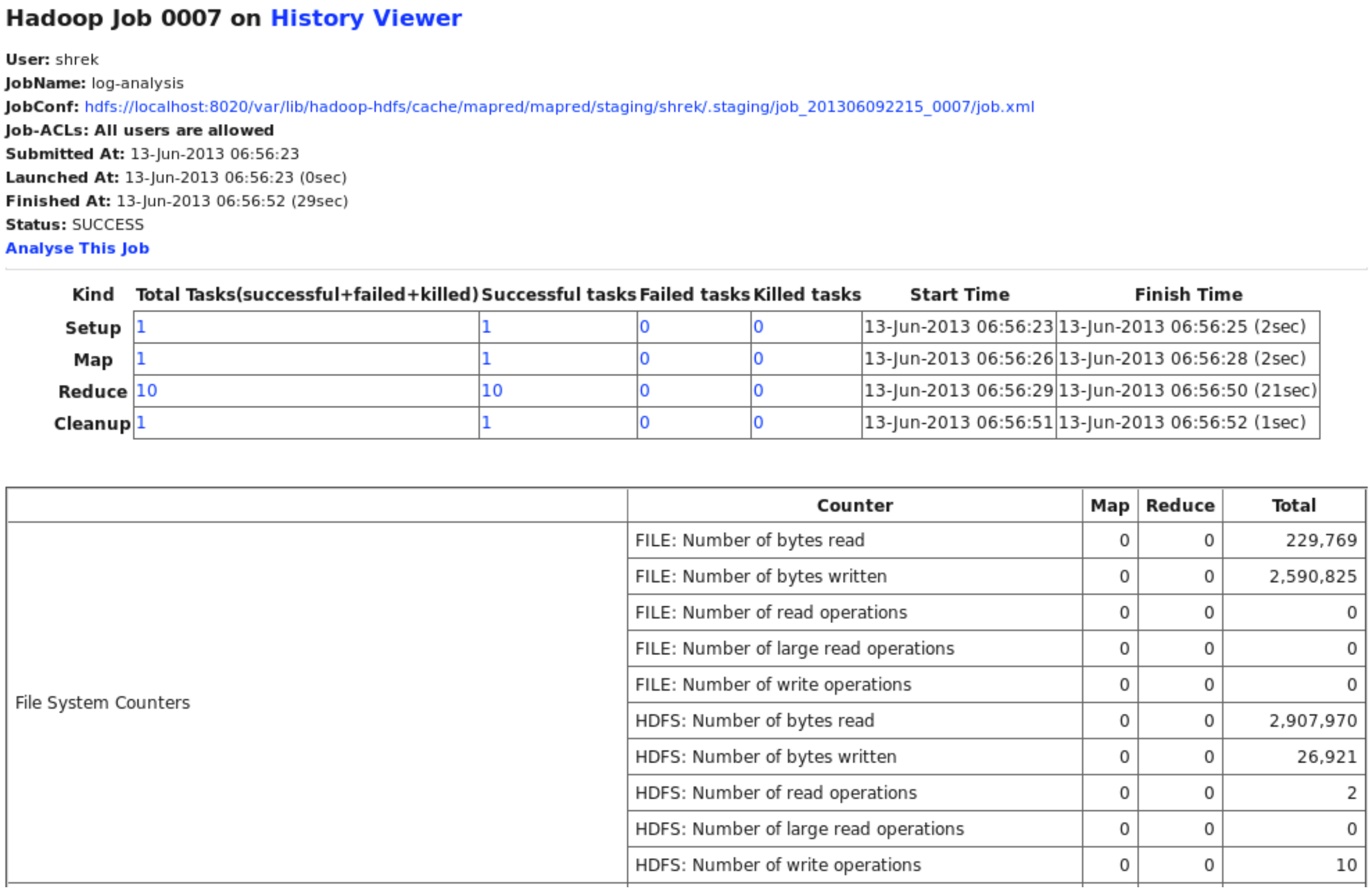
Input Split Locations

/default-rack/localhost

Task 2: Deep analysis of a retired job

Activity Procedure

Step 1: Go to the job tracker page http://localhost:50030 and find a retired job. Click on the link to see more details. You'll find something similar to the following:



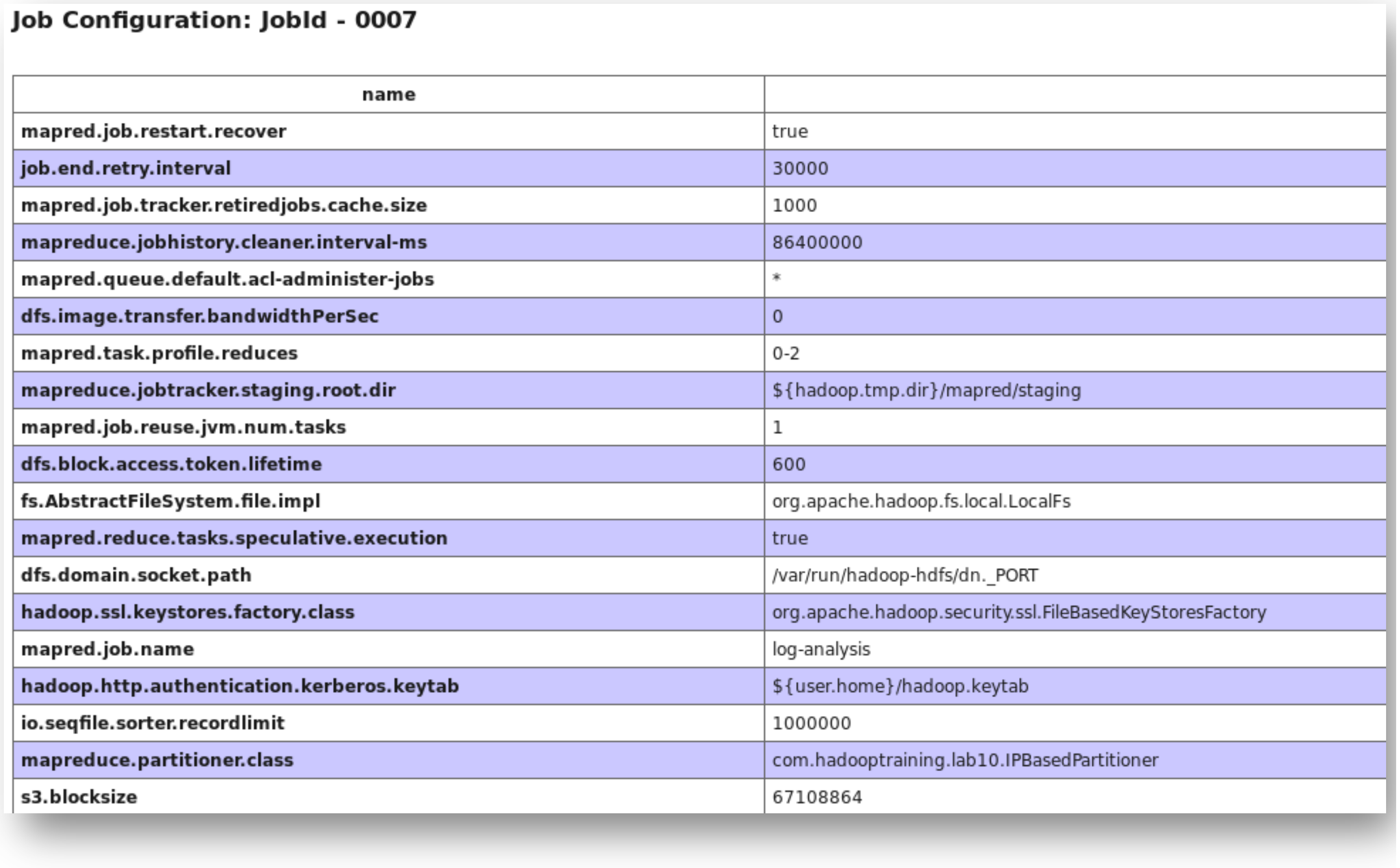
Hadoop Job 0007 on History Viewer

User: shrek
JobName: log-analysis
JobConf: hdfs://localhost:8020/var/lib/hadoop-hdfs/cache/mapred/mapred/staging/shrek/.staging/job_201306092215_0007/job.xml
Job-ACLs: All users are allowed
Submitted At: 13-Jun-2013 06:56:23
Launched At: 13-Jun-2013 06:56:23 (0sec)
Finished At: 13-Jun-2013 06:56:52 (29sec)
Status: SUCCESS
Analyse This Job

Kind	Total Tasks(successful+failed+killed)	Successful tasks	Failed tasks	Killed tasks	Start Time	Finish Time
Setup	1	1	0	0	13-Jun-2013 06:56:23	13-Jun-2013 06:56:25 (2sec)
Map	1	1	0	0	13-Jun-2013 06:56:26	13-Jun-2013 06:56:28 (2sec)
Reduce	10	10	0	0	13-Jun-2013 06:56:29	13-Jun-2013 06:56:50 (21sec)
Cleanup	1	1	0	0	13-Jun-2013 06:56:51	13-Jun-2013 06:56:52 (1sec)

	Counter	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	0	0	229,769
	FILE: Number of bytes written	0	0	2,590,825
	FILE: Number of read operations	0	0	0
	FILE: Number of large read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	0	0	2,907,970
	HDFS: Number of bytes written	0	0	26,921
	HDFS: Number of read operations	0	0	2
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of write operations	0	0	10

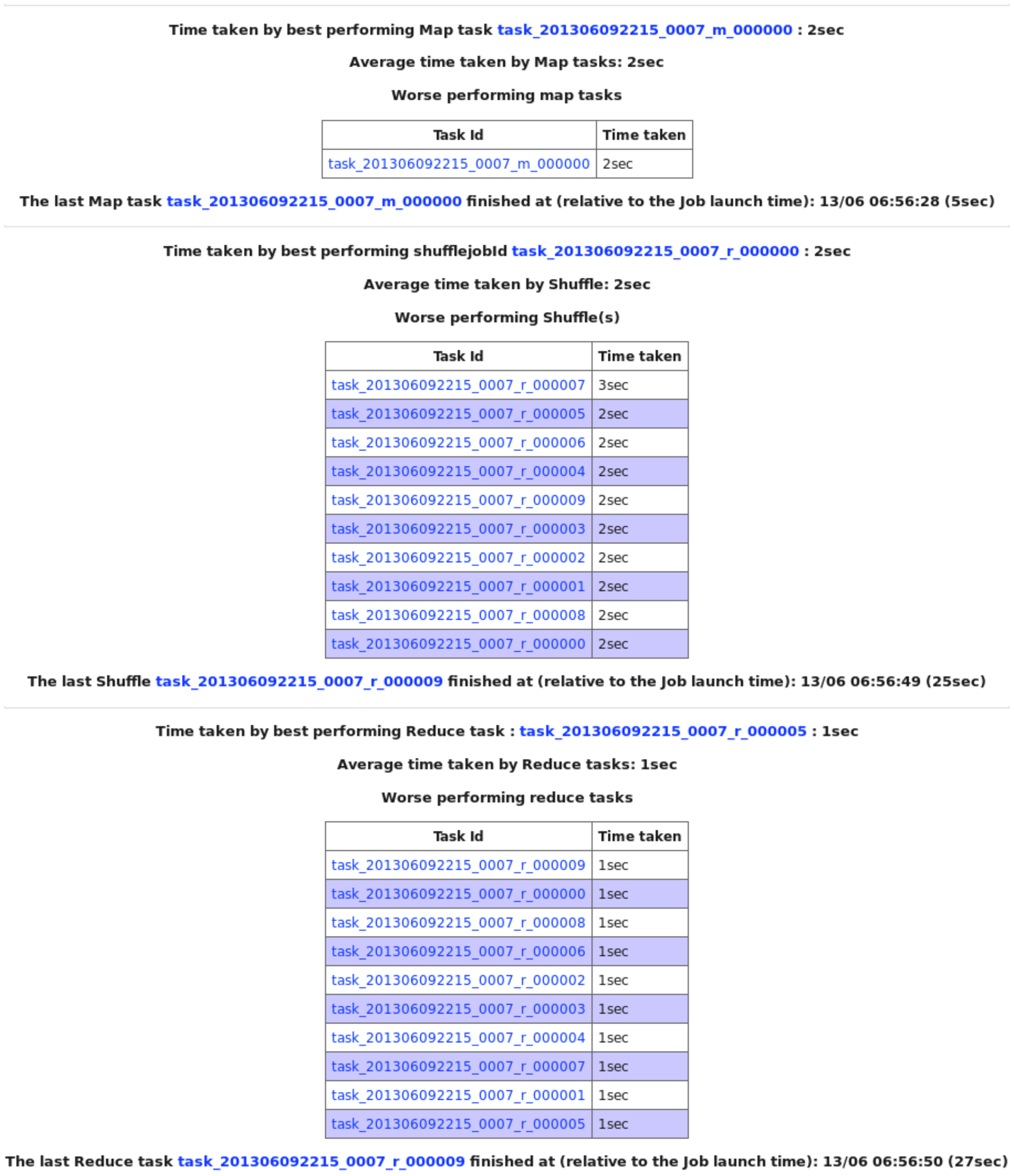
Step 2: Click on the JobConf link (the very first link) to see the job configuration.



Job Configuration: JobId - 0007

name	
mapred.job.restart.recover	true
job.end.retry.interval	30000
mapred.job.tracker.retiredjobs.cache.size	1000
mapreduce.jobhistory.cleaner.interval-ms	86400000
mapred.queue.default.acl-administer-jobs	*
dfs.image.transfer.bandwidthPerSec	0
mapred.task.profile.reduces	0-2
mapreduce.jobtracker.staging.root.dir	${hadoop.tmp.dir}/mapred/staging
mapred.job.reuse.jvm.num.tasks	1
dfs.block.access.token.lifetime	600
fs.AbstractFileSystem.file.impl	org.apache.hadoop.fs.local.LocalFs
mapred.reduce.tasks.speculative.execution	true
dfs.domain.socket.path	/var/run/hadoop-hdfs/dn._PORT
hadoop.ssl.keystores.factory.class	org.apache.hadoop.security.ssl.FileBasedKeyStoresFactory
mapred.job.name	log-analysis
hadoop.http.authentication.kerberos.keytab	${user.home}/hadoop.keytab
io.seqfile.sorter.recordlimit	1000000
mapreduce.partitioner.class	com.hadooptraining.lab10.IPBasedPartitioner
s3.blocksize	67108864

Step 3: Click on the link **Analyse This Job** to see the job execution details.



Time taken by best performing Map task task_201306092215_0007_m_000000 : 2sec

Average time taken by Map tasks: 2sec

Worse performing map tasks

Task Id	Time taken
task_201306092215_0007_m_000000	2sec

The last Map task task_201306092215_0007_m_000000 finished at (relative to the Job launch time): 13/06 06:56:28 (5sec)

Time taken by best performing shufflejobId task_201306092215_0007_r_000000 : 2sec

Average time taken by Shuffle: 2sec

Worse performing Shuffle(s)

Task Id	Time taken
task_201306092215_0007_r_000007	3sec
task_201306092215_0007_r_000005	2sec
task_201306092215_0007_r_000006	2sec
task_201306092215_0007_r_000004	2sec
task_201306092215_0007_r_000009	2sec
task_201306092215_0007_r_000003	2sec
task_201306092215_0007_r_000002	2sec
task_201306092215_0007_r_000001	2sec
task_201306092215_0007_r_000008	2sec
task_201306092215_0007_r_000000	2sec

The last Shuffle task_201306092215_0007_r_000009 finished at (relative to the Job launch time): 13/06 06:56:49 (25sec)

Time taken by best performing Reduce task : task_201306092215_0007_r_000005 : 1sec

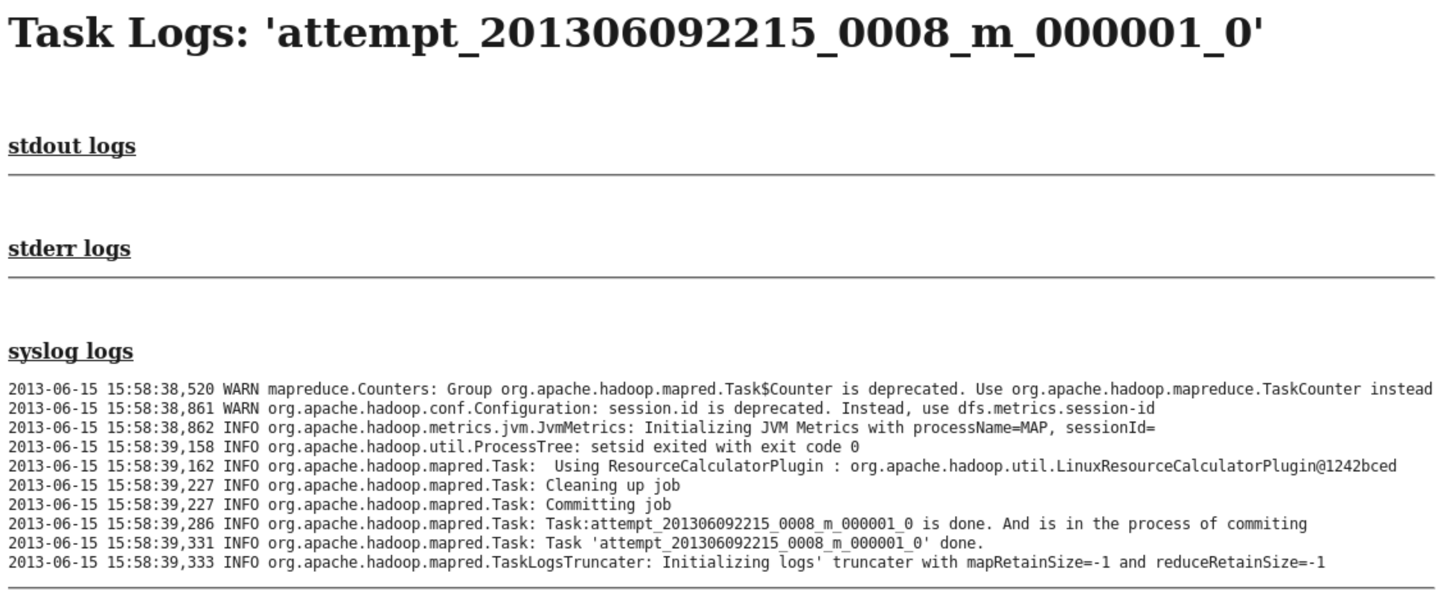
Average time taken by Reduce tasks: 1sec

Worse performing reduce tasks

Task Id	Time taken
task_201306092215_0007_r_000009	1sec
task_201306092215_0007_r_000000	1sec
task_201306092215_0007_r_000008	1sec
task_201306092215_0007_r_000006	1sec
task_201306092215_0007_r_000002	1sec
task_201306092215_0007_r_000003	1sec
task_201306092215_0007_r_000004	1sec
task_201306092215_0007_r_000007	1sec
task_201306092215_0007_r_000001	1sec
task_201306092215_0007_r_000005	1sec

The last Reduce task task_201306092215_0007_r_000009 finished at (relative to the Job launch time): 13/06 06:56:50 (27sec)

Step 3: Click on the link **All** under task details to see the raw log files.



Task Logs: 'attempt_201306092215_0008_m_000001_0'

stdout logs

stderr logs

syslog logs

```
2013-06-15 15:58:38,520 WARN mapreduce.Counters: Group org.apache.hadoop.mapred.Task$Counter is deprecated. Use org.apache.hadoop.mapreduce.TaskCounter instead
2013-06-15 15:58:38,861 WARN org.apache.hadoop.conf.Configuration: session.id is deprecated. Instead, use dfs.metrics.session-id
2013-06-15 15:58:38,862 INFO org.apache.hadoop.metrics.jvm.JvmMetrics: Initializing JVM Metrics with processName=MAP, sessionId=
2013-06-15 15:58:39,158 INFO org.apache.hadoop.util.ProcessTree: setsid exited with exit code 0
2013-06-15 15:58:39,162 INFO org.apache.hadoop.mapred.Task:  Using ResourceCalculatorPlugin : org.apache.hadoop.util.LinuxResourceCalculatorPlugin@1242bced
2013-06-15 15:58:39,227 INFO org.apache.hadoop.mapred.Task: Cleaning up job
2013-06-15 15:58:39,227 INFO org.apache.hadoop.mapred.Task: Committing job
2013-06-15 15:58:39,286 INFO org.apache.hadoop.mapred.Task: Task:attempt_201306092215_0008_m_000001_0 is done. And is in the process of commiting
2013-06-15 15:58:39,331 INFO org.apache.hadoop.mapred.Task: Task 'attempt_201306092215_0008_m_000001_0' done.
2013-06-15 15:58:39,333 INFO org.apache.hadoop.mapred.TaskLogsTruncater: Initializing logs' truncater with mapRetainSize=-1 and reduceRetainSize=-1
```

Lab 14: Debug MapReduce code

Description:

- Write a simple MapReduce program that uses log4j and standard error output (stderr) to debug code.
- We will reuse one of our earlier lab exercises (Lab 7) and add some debugging code in it.
- The output will be written to the file system under **/var/log**. We will then look up the log files and find the log messages.

Task 1: Prepare HDFS with the appropriate data files

Activity Procedure

Step 1: We will be re-doing Lab 7 - that analyzes log files using custom Writable type.

```
$ cd ~/Developer/heffalump
```

Step 2: Clean up the input directory in HDFS and copy the data file to HDFS.

```
$ hadoop fs -rm -r input
$ hadoop fs -mkdir input
$ hadoop fs -copyFromLocal -f data/NASA_access_log.txt input
```

Task 2: Write the custom writable object LogWritable

Most of the code is written for you. You just need to edit the file and add a few key functions.

Activity Procedure

Step 1: Edit the file containing the Mapper and Reducer class. Fill in the missing lines in the **map()** function below.

```
$ vi src/main/java/com/hadooptraining/lab14/LogProcessorWithDebug.java
```

Fill in the missing lines in the function **map()** by looking up the code below.

```
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {

            // prepare the log pattern string
            String logEntryPattern = "^(\\S+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-
]\\d{4})\\] \"(.+?)\" (\\d{3}) (\\d+)";
            // Example: unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400]
"GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310

            // Compile the pattern and keep it in a local variable
            Pattern p = Pattern.compile(logEntryPattern);
            Matcher matcher = p.matcher(value.toString());

            // if line in log file did not match, get out of the mapper method
            if (!matcher.matches()) {
                System.err.println("Bad record found: " + value.toString());
                return;
            }

            // Set the KEY of the mapper by using one of the extracted values
from log line
            userHostText.set(matcher.group(1));

            // Set the VALUE of the mapper by using other extracted values from
log line
            logValue.set(matcher.group(1), matcher.group(4), matcher.group(5),
Integer.parseInt(matcher.group(7)),  Integer.parseInt(matcher.group(6)));

            // Print out some values to the console for debugging purposes
            System.out.println("Host: " + userHostText.toString() + "\t\tbytes
read: " + logValue.getResponseSize());

            // Write the key and value to the context object
            context.write(userHostText, logValue);
        }
    }
```

Next fill in the missing lines in the function **`reduce()`** by looking up the code below.

```
    public void reduce(Text key, Iterable<LogWritable> values,  Context context)
                throws IOException, InterruptedException {

            // Create a local variable to store the sum
            int sum = 0;

            // Iterate through each value received
            for (LogWritable logLine : values) {
                // extract the response size and add it up
                sum += logLine.getResponseSize().get();
            }

            // Set the value of the output as calculated sum
            result.set(sum);
            // Send some diagnostic messages to the log4j logger
            logger.info("[" + key.toString() + "] -> " + result.toString());

            // Write to the context object
            context.write(key, result);
        }
    }
```

Task 3: Compile and run the MapReduce job

Activity Procedure

Step 1: Compile the program through Maven and create a package.

```
$ cd ~/Developer/heffalump
$ mvn package
```

Step 2: Run the MapReduce job.

```
$ hadoop jar $HOME/Developer/heffalump/target/heffalump-1.0.jar
com.hadooptraining.lab14.LogProcessorWithDebug input output
```

Activity Verification

As the job runs, you'll see output on the screen that looks similar to the following:

```
13/06/16 12:28:11 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
13/06/16 12:28:11 INFO input.FileInputFormat: Total input paths to process : 1
13/06/16 12:28:12 INFO mapred.JobClient: Running job: job_201306092215_0012
13/06/16 12:28:13 INFO mapred.JobClient:  map 0% reduce 0%
13/06/16 12:28:19 INFO mapred.JobClient:  map 100% reduce 0%
13/06/16 12:28:22 INFO mapred.JobClient:  map 100% reduce 100%
13/06/16 12:28:23 INFO mapred.JobClient: Job complete: job_201306092215_0012
13/06/16 12:28:23 INFO mapred.JobClient: Counters: 32
13/06/16 12:28:23 INFO mapred.JobClient:   File System Counters
13/06/16 12:28:23 INFO mapred.JobClient:     FILE: Number of bytes read=1433238
13/06/16 12:28:23 INFO mapred.JobClient:     FILE: Number of bytes written=3254351
13/06/16 12:28:23 INFO mapred.JobClient:     FILE: Number of read operations=0
13/06/16 12:28:23 INFO mapred.JobClient:     FILE: Number of large read operations=0
13/06/16 12:28:23 INFO mapred.JobClient:     FILE: Number of write operations=0
13/06/16 12:28:23 INFO mapred.JobClient:     HDFS: Number of bytes read=1290522
13/06/16 12:28:23 INFO mapred.JobClient:     HDFS: Number of bytes written=29137
13/06/16 12:28:23 INFO mapred.JobClient:     HDFS: Number of read operations=2
13/06/16 12:28:23 INFO mapred.JobClient:     HDFS: Number of large read operations=0
```

```
13/06/16 12:28:23 INFO mapred.JobClient:     HDFS: Number of write operations=1
13/06/16 12:28:23 INFO mapred.JobClient:   Job Counters
13/06/16 12:28:23 INFO mapred.JobClient:     Launched map tasks=1
13/06/16 12:28:23 INFO mapred.JobClient:     Launched reduce tasks=1
13/06/16 12:28:23 INFO mapred.JobClient:     Data-local map tasks=1
13/06/16 12:28:23 INFO mapred.JobClient:     Total time spent by all maps in occupied slots (ms)=6546
13/06/16 12:28:23 INFO mapred.JobClient:     Total time spent by all reduces in occupied slots
(ms)=3507
13/06/16 12:28:23 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving
slots (ms)=0
13/06/16 12:28:23 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving
slots (ms)=0
13/06/16 12:28:23 INFO mapred.JobClient:   Map-Reduce Framework
13/06/16 12:28:23 INFO mapred.JobClient:     Map input records=11676
13/06/16 12:28:23 INFO mapred.JobClient:     Map output records=11552
13/06/16 12:28:23 INFO mapred.JobClient:     Map output bytes=1410014
13/06/16 12:28:23 INFO mapred.JobClient:     Input split bytes=123
13/06/16 12:28:23 INFO mapred.JobClient:     Combine input records=0
13/06/16 12:28:23 INFO mapred.JobClient:     Combine output records=0
13/06/16 12:28:23 INFO mapred.JobClient:     Reduce input groups=1085
13/06/16 12:28:23 INFO mapred.JobClient:     Reduce shuffle bytes=1433238
13/06/16 12:28:23 INFO mapred.JobClient:     Reduce input records=11552
13/06/16 12:28:23 INFO mapred.JobClient:     Reduce output records=1085
13/06/16 12:28:23 INFO mapred.JobClient:     Spilled Records=23104
13/06/16 12:28:23 INFO mapred.JobClient:     CPU time spent (ms)=4020
13/06/16 12:28:23 INFO mapred.JobClient:     Physical memory (bytes) snapshot=413298688
13/06/16 12:28:23 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=1525846016
13/06/16 12:28:23 INFO mapred.JobClient:     Total committed heap usage (bytes)=317128704
```

However, the intent of this exercise is to see the debug log output written by the logger and **`println()`** statements.

Task 4: Look at debug logs on the file system

Activity Procedure

Step 1: The debug logs are generally under **`/var/log`** which can only be accessed by **`root`** user. Type the following to become **`root`** user. Your instructor will tell you the password.

```
$ su -
$ cd /var/log/hadoop-0.20-mapreduce/userlogs
$ ls -lrt
```

Activity Verification

You'll see a listing similar to the following.

```
total 20
```

```
drwx--x---. 2 mapred mapred 4096 Jun 15 15:58 job_201306092215_0008
drwx--x---. 2 mapred mapred 4096 Jun 15 17:59 job_201306092215_0009
drwx--x---. 2 mapred mapred 4096 Jun 16 08:42 job_201306092215_0010
drwx--x---. 2 mapred mapred 4096 Jun 16 09:11 job_201306092215_0011
drwx--x---. 2 mapred mapred 4096 Jun 16 12:28 job_201306092215_0012
```

Select the last one in this list and change directory to that job.

```
$ cd job_201306092215_0012
$ ls -lrt
lrwxrwxrwx. 1 mapred mapred 114 Jun 16 12:28 attempt_201306092215_0012_m_000002_0 ->
/var/lib/hadoop-
hdfs/cache/mapred/mapred/local/userlogs/job_201306092215_0012/attempt_201306092215_0012
_m_000002_0
lrwxrwxrwx. 1 mapred mapred 114 Jun 16 12:28 attempt_201306092215_0012_m_000000_0 ->
/var/lib/hadoop-
hdfs/cache/mapred/mapred/local/userlogs/job_201306092215_0012/attempt_201306092215_0012
_m_000000_0
lrwxrwxrwx. 1 mapred mapred 114 Jun 16 12:28 attempt_201306092215_0012_r_000000_0 ->
/var/lib/hadoop-
hdfs/cache/mapred/mapred/local/userlogs/job_201306092215_0012/attempt_201306092215_0012
_r_000000_0
lrwxrwxrwx. 1 mapred mapred 114 Jun 16 12:28 attempt_201306092215_0012_m_000001_0 ->
/var/lib/hadoop-
hdfs/cache/mapred/mapred/local/userlogs/job_201306092215_0012/attempt_201306092215_0012
_m_000001_0
```

You'll notice that these are soft links to the folders for each of the jobs. There are three map jobs and one reduce job in the listing above. Those marked with **`_m_`** are map jobs while those markes with **`_r_`** are reduce jobs. Change directory to one to the map jobs.

```
$ cd attempt_201306092215_0012_m_000001_0
$ ls -lrt
total 528
-rw-r--r--. 1 mapred mapred  15060 Jun 16 12:28 stderr
-rw-r--r--. 1 mapred mapred 512927 Jun 16 12:28 stdout
-rw-r--r--. 1 mapred mapred   1805 Jun 16 12:28 syslog
-rw-r--r--. 1 mapred mapred    143 Jun 16 12:28 log.index
```

Finally you will see the stderr and stdout log files here. Look at the files using **`less`** command.

```
$ less stderr
Bad record found: dd15-062.compuserve.com - - [01/Jul/1995:00:01:12 -0400] "GET
/news/sci.space.shuttle/archive/sci-space-shuttle-22-apr-1995-40.txt HTTP/1.0" 404 -
Bad record found: dynip42.efn.org - - [01/Jul/1995:00:02:14 -0400] "GET /software
HTTP/1.0" 302 -
```

```
Bad record found: ix-or10-06.ix.netcom.com - - [01/Jul/1995:00:02:40 -0400] "GET
/software/winvn HTTP/1.0" 302 -
Bad record found: ix-or10-06.ix.netcom.com - - [01/Jul/1995:00:03:24 -0400] "GET
/software HTTP/1.0" 302 -
Bad record found: link097.txdirect.net - - [01/Jul/1995:00:05:06 -0400] "GET /shuttle
HTTP/1.0" 302 -
Bad record found: ix-war-mi1-20.ix.netcom.com - - [01/Jul/1995:00:05:13 -0400] "GET
/shuttle/missions/sts-78/news HTTP/1.0" 302 -
```

Type '**q**' to get out of **less**.

```
$ less stdout
Host: 199.72.81.55              bytes read: 6245
Host: unicomp6.unicomp.net              bytes read: 3985
Host: 199.120.110.21            bytes read: 4085
Host: burger.letters.com                bytes read: 0
Host: 199.120.110.21            bytes read: 4179
Host: burger.letters.com                bytes read: 0
Host: burger.letters.com                bytes read: 0
Host: 205.212.115.106           bytes read: 3985
Host: d104.aa.net               bytes read: 3985
Host: 129.94.144.152            bytes read: 7074
```

Type '**q**' to get out of **less**. Now change directory to the reduce task.

```
$cd attempt_201306092215_0012_r_000001_0
$ ls -lrt
total 124
-rw-r--r--. 1 mapred mapred      0 Jun 16 12:28 stdout
-rw-r--r--. 1 mapred mapred      0 Jun 16 12:28 stderr
-rw-r--r--. 1 mapred mapred 122156 Jun 16 12:28 syslog
-rw-r--r--. 1 mapred mapred    143 Jun 16 12:28 log.index
```

Your **log4j** output can be found in the **syslog** file.

```
$ less syslog
2013-06-16 12:28:18,398 WARN mapreduce.Counters: Group org.apache.hadoop.mapred.Task$Counter is
deprecated. Use org.apache.hadoop.mapreduce.TaskCounter instead
2013-06-16 12:28:18,745 WARN org.apache.hadoop.conf.Configuration: session.id is deprecated. Instead,
use dfs.metrics.session-id
2013-06-16 12:28:18,746 INFO org.apache.hadoop.metrics.jvm.JvmMetrics: Initializing JVM Metrics with
processName=SHUFFLE, sessionId=
2013-06-16 12:28:19,036 INFO org.apache.hadoop.util.ProcessTree: setsid exited with exit code 0
2013-06-16 12:28:19,040 INFO org.apache.hadoop.mapred.Task:  Using ResourceCalculatorPlugin :
org.apache.hadoop.util.LinuxResourceCalculatorPlugin@2c97c9cf
2013-06-16 12:28:19,108 INFO org.apache.hadoop.mapred.ReduceTask: Using ShuffleConsumerPlugin:
org.apache.hadoop.mapred.ReduceTask$ReduceCopier
```

```
2013-06-16 12:28:19,113 INFO org.apache.hadoop.mapred.ReduceTask: ShuffleRamManager:
MemoryLimit=130514944, MaxSingleShuffleLimit=32628736
2013-06-16 12:28:19,121 INFO org.apache.hadoop.mapred.ReduceTask:
attempt_201306092215_0012_r_000000_0 Thread started: Thread for merging on-disk files
2013-06-16 12:28:19,121 INFO org.apache.hadoop.mapred.ReduceTask:
attempt_201306092215_0012_r_000000_0 Thread started: Thread for merging in memory files
2013-06-16 12:28:19,121 INFO org.apache.hadoop.mapred.ReduceTask:
attempt_201306092215_0012_r_000000_0 Thread waiting: Thread for merging on-disk files
2013-06-16 12:28:19,123 INFO org.apache.hadoop.mapred.ReduceTask:
attempt_201306092215_0012_r_000000_0 Need another 1 map output(s) where 0 is already in progress
2013-06-16 12:28:19,123 INFO org.apache.hadoop.mapred.ReduceTask:
attempt_201306092215_0012_r_000000_0 Thread started: Thread for polling Map Completion Events
2013-06-16 12:28:19,123 INFO org.apache.hadoop.mapred.ReduceTask:
attempt_201306092215_0012_r_000000_0 Scheduled 0 outputs (0 slow hosts and0 dup hosts)
2013-06-16 12:28:19,127 INFO org.apache.hadoop.mapred.ReduceTask:
attempt_201306092215_0012_r_000000_0 Scheduled 1 outputs (0 slow hosts and0 dup hosts)
2013-06-16 12:28:19,302 INFO org.apache.hadoop.mapred.ReduceTask: GetMapEventsThread exiting
2013-06-16 12:28:19,303 INFO org.apache.hadoop.mapred.ReduceTask: getMapsEventsThread joined.
2013-06-16 12:28:19,303 INFO org.apache.hadoop.mapred.ReduceTask: Closed ram manager
2013-06-16 12:28:19,303 INFO org.apache.hadoop.mapred.ReduceTask: Interleaved on-disk merge complete:
0 files left.
2013-06-16 12:28:19,303 INFO org.apache.hadoop.mapred.ReduceTask: In-memory merge complete: 1 files
left.
2013-06-16 12:28:19,333 INFO org.apache.hadoop.mapred.Merger: Merging 1 sorted segments
2013-06-16 12:28:19,334 INFO org.apache.hadoop.mapred.Merger: Down to the last merge-pass, with 1
segments left of total size: 1433234 bytes
2013-06-16 12:28:19,484 INFO org.apache.hadoop.mapred.ReduceTask: Merged 1 segments, 1433234 bytes to
disk to satisfy reduce memory limit
2013-06-16 12:28:19,484 INFO org.apache.hadoop.mapred.ReduceTask: Merging 1 files, 1433238 bytes from
disk
2013-06-16 12:28:19,485 INFO org.apache.hadoop.mapred.ReduceTask: Merging 0 segments, 0 bytes from
memory into reduce
2013-06-16 12:28:19,485 INFO org.apache.hadoop.mapred.Merger: Merging 1 sorted segments
2013-06-16 12:28:19,487 INFO org.apache.hadoop.mapred.Merger: Down to the last merge-pass, with 1
segments left of total size: 1433234 bytes
2013-06-16 12:28:19,538 INFO com.hadooptraining.lab14.LogProcessorWithDebug: [128.187.140.171] ->
60655
2013-06-16 12:28:19,540 INFO com.hadooptraining.lab14.LogProcessorWithDebug: [129.188.154.200] ->
1607516
2013-06-16 12:28:19,540 INFO com.hadooptraining.lab14.LogProcessorWithDebug: [129.193.116.41] ->
46353
2013-06-16 12:28:19,540 INFO com.hadooptraining.lab14.LogProcessorWithDebug: [129.59.205.2] -> 109598
2013-06-16 12:28:19,540 INFO com.hadooptraining.lab14.LogProcessorWithDebug: [129.79.164.64] ->
1037740
2013-06-16 12:28:19,540 INFO com.hadooptraining.lab14.LogProcessorWithDebug: [129.94.144.152] ->
321543
2013-06-16 12:28:19,541 INFO com.hadooptraining.lab14.LogProcessorWithDebug: [130.161.103.176] ->
143215
```

Remember to type '**`q`**' to get out of **`less`**. The lines at the bottom are coming from the code that we introduced in our program.

Task 5: Get out of superuser mode

Activity Procedure

Step 1: Since you are seeing these logs as superuser, you need to get out of it and become a normal user for your next exercise.

```
$ exit
```

Lab 15: Practice MapReduce use-case

Description:

- You will be asked to analyze a fairly large set of statistics - batching and pitching records of baseball players since 1871.
- This lab has two parts:
- In **part A**, you will be writing a MapReduce application that will determine the total number of runs allowed by a pitcher in their lifetime.
- In **part B**, you will be calculating the runs made in each year since baseball records are available.
- Using this data, you will find out the year in recent history when the highest runs were scored.
- Instructions for this lab are intentionally kept at a minimum - for your practice.
- You can download baseball statistics data from http://seanlahman.com

Task 1: Open the input data for Part A, analyze it and copy it to HDFS

Activity Procedure

Step 1: Go to the location of the source code for the labs. Inspect the data file for part A of this exercise. In the first part you will be calculating the lifetime runs allowed by a pitcher.

```
$ cd ~/Developer/heffalump
$ less data/baseballstats/Pitching.csv
```

Notice that the pitching data is in the following format:

```
      PITCHING TABLE
      ==============
1     playerID       Player ID code
2     yearID         Year
3     stint          player's stint (order of appearances within a season)
4     teamID         Team
5     lgID           League
6     W              Wins
7     L              Losses
8     G              Games
9     GS             Games Started
10     CG             Complete Games
11    SHO            Shutouts
12    SV             Saves
13    IPOuts         Outs Pitched (innings pitched x 3)
14    H              Hits
15    ER             Earned Runs
16    HR             Homeruns
```

17	BB	Walks
18	SO	Strikeouts
19	BAOpp	Opponent's Batting Average
20	ERA	Earned Run Average
21	IBB	Intentional Walks
22	WP	Wild Pitches
23	HBP	Batters Hit By Pitch
24	BK	Balks
25	BFP	Batters faced by Pitcher
26	GF	Games Finished
27	R	Runs Allowed
28	SH	Sacrifices by opposing batters
29	SF	Sacrifice flies by opposing batters
30	GIDP	Grounded into double plays by opposing batter

Step 2: Clean up the input directory in HDFS and copy the data file to HDFS.

```
$ hadoop fs -rm -r input
$ hadoop fs -mkdir input
$ hadoop fs -copyFromLocal -f data/baseballstats/Pitching.csv input
$ hadoop fs -ls input
```

Task 2: Write the custom writable object for storing Pitching data

You can use Lab 7 as your base for the LogWritable object. Instructions below are minimal.

Activity Procedure

Step 1: Edit the file containing the custom writable object **PitchingWritable**.

```
$ vi src/main/java/com/hadooptraining/lab15/PitchingWritable.java
```

Fill in the missing lines below.

```
public class PitchingWritable implements Writable {

    private Text playerID, year;
    private IntWritable runsAllowed;

    public PitchingWritable() {
      // TODO
    }

    public void set (String playerID, String year, int runsAllowed) {
      // TODO
```

```
        }

        @Override
        public void readFields(DataInput in) throws IOException {
            // TODO
        }

        @Override
        public void write(DataOutput out) throws IOException {
            // TODO
        }

        public int hashCode() {
            // TODO
        }

        public Text getPlayerID() {
            // TODO
        }

        public Text getYear() {
            // TODO
        }

        public IntWritable getRunsAllowed() {
            // TODO
        }
    }
```

Task 3: Write the Mapper and Reducer class

Activity Procedure

Step 1: Edit the file containing the Mapper and Reducer class. Fill in the missing lines in the **`map()`** and **`reduce()`** functions below.

```
$ vi src/main/java/com/hadooptraining/lab15/PitchingDataProcessor.java
```

Fill in the missing functions below.

```
public class PitchingDataProcessor extends Configured implements Tool {

    public static class PitchingDataProcessorMap extends
            Mapper<LongWritable, Text, Text, PitchingWritable> {
```

```
        private Text playerIDText = new Text();
        private PitchingWritable pitchingValue = new PitchingWritable();

        /**
         * Mapper class using 'playerID' as key and 'PitchingWritable' as VALUE.
         */
        public void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {

            // TODO
        }
    }

    public static class PitchingDataProcessorReduce extends
            Reducer<Text, PitchingWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<PitchingWritable> values, Context
context)
                throws IOException, InterruptedException {

             // TODO
        }
    }

@Override
    public int run(String[] args) throws Exception {
        if (args.length < 2) {
            System.err.println("Usage: <input_path> <output_path>");
            System.exit(-1);
        }

        Job job = Job.getInstance(getConf(), "pitching-analysis");
        job.setJarByClass(PitchingDataProcessor.class);
        job.setMapperClass(PitchingDataProcessorMap.class);
        job.setReducerClass(PitchingDataProcessorReduce.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(PitchingWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.setInputPaths(job, new Path(args[0]));
```

```
            FileOutputFormat.setOutputPath(job, new Path(args[1]));
            return job.waitForCompletion(true) ? 0 : 1;
        }

        public static void main(String[] args) throws Exception {
            int res = ToolRunner.run(new Configuration(), new
    PitchingDataProcessor(), args);
            System.exit(res);
        }
```

Task 4: Compile and run the MapReduce job

Activity Procedure

Step 1: Compile the program through Maven and create a package.

```
$ cd ~/Developer/heffalump
$ mvn package
```

Step 2: Run the MapReduce job.

```
$ hadoop jar $HOME/Developer/heffalump/target/heffalump-1.0.jar
com.hadooptraining.lab15.PitchingDataProcessor input output
```

Activity Verification

As the job runs, you'll see familiar output on the screen.

Your output will look like:

```
$ hadoop fs -cat output/part-r-00000
aardsda01    132
aasedo01     503
abadfe01     51
abbeybe01    442
abbeych01    3
abbotda01    14
abbotgl01    707
```

Confirm the number of lines in your output by checking the following:

```
$ hadoop fs -cat output/part-r-00000 | wc -l
8673
```

Task 5: Open the input data for Part B, analyze it and copy it to HDFS

Activity Procedure

Step 1: Go to the location of the source code for the labs. Inspect the data file for part B of this exercise. In the second part you will be calculating the total runs made each year in baseball since it started.

```
$ cd ~/Developer/heffalump
$ less data/baseballstats/Batting.csv
```

Notice that the batting data is in the following format:

```
          Batting TABLE
          =============
1         playerID       Player ID code
2         yearID         Year
3         stint          player's stint (order of appearances within a season)
4         teamID         Team
5         lgID           League
6         G              Games
7         G_batting      Game as batter
8         AB             At Bats
9         R              Runs
10        H              Hits
11        2B             Doubles
12        3B             Triples
13        HR             Homeruns
14        RBI            Runs Batted In
15        SB             Stolen Bases
16        CS             Caught Stealing
17        BB             Base on Balls
18        SO             Strikeouts
19        IBB            Intentional walks
20        HBP            Hit by pitch
21        SH             Sacrifice hits
22        SF             Sacrifice flies
23        GIDP           Grounded into double plays
24        G_Old          Old version of games (deprecated)
```

Step 2: Clean up the input directory in HDFS and copy the data file to HDFS.

```
$ hadoop fs -rm -r input
$ hadoop fs -mkdir input
$ hadoop fs -copyFromLocal -f data/baseballstats/Batting.csv input
$ hadoop fs -ls input
```

Task 6: Write the custom writable object for storing Batting data

You can use the previous class you wrote **PitchingWritable** as your base for the **BattingWritable** class.

Activity Procedure

Step 1: Edit the file containing the custom writable object **BattingWritable**.

```
$ vi src/main/java/com/hadooptraining/lab15/BattingWritable.java
```

Fill in the missing lines below.

```
public class BattingWritable implements Writable {

    private Text playerID, year;
    private IntWritable runs;

    public BattingWritable() {
        // TODO
    }

    public void set (String playerID, String year, int runsAllowed) {
        // TODO
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        // TODO
    }

    @Override
    public void write(DataOutput out) throws IOException {
        // TODO
    }

    public int hashCode() {
        // TODO
    }

    public Text getPlayerID() {
        // TODO
    }

    public Text getYear() {
        // TODO
    }

    public IntWritable getRuns() {
        // TODO
    }
}
```

Task 7: Write the Mapper and Reducer class

Activity Procedure

Step 1: Edit the file containing the Mapper and Reducer class. Fill in the missing lines in the **`map()`** and **`reduce()`** functions below.

```
$ vi src/main/java/com/hadooptraining/lab15/BattingDataProcessor.java
```

Fill in the missing functions below.

```
public class BattingDataProcessor extends Configured implements Tool {

    /**
     * Mapper class using 'year' as key and 'BattingWritable' as VALUE.
     */
    public static class BattingDataProcessorMap extends
            Mapper<LongWritable, Text, Text, BattingWritable> {
        private Text year = new Text();
        private BattingWritable battingValue = new BattingWritable();

        public void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            // Format:
playerID,yearID,stint,teamID,lgID,G,G_batting,AB,R,H,2B,3B,HR,RBI,SB,CS,BB,SO,IBB
,HBP,SH,SF,GIDP,G_old

            String entryPattern =
"^(\\S+),(\\d{4}),(\\d+),(\\S+),(\\S+),(\\d+),(\\d*),(\\d*),(\\d*),(\\d*),(\\d*),
(\\d*),(\\d*),(\\d*),(\\d*),(\\d*),(\\d*),(\\d*),(\\d*),(\\d*),(\\d*),(\\d*),(\\d
*),(\\d*)";

            // TODO
        }
    }

    /**
     * Reducer class to add up all numbers associated with key.
     */
    public static class BattingDataProcessorReduce extends
            Reducer<Text, BattingWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();
        public void reduce(Text key, Iterable<BattingWritable> values, Context
context)
                throws IOException, InterruptedException {
            // TODO
        }
    }

@Override
    public int run(String[] args) throws Exception {
        if (args.length < 2) {
            System.err.println("Usage: <input_path> <output_path>");
            System.exit(-1);
        }
        Job job = Job.getInstance(getConf(), "batting-analysis");
        job.setJarByClass(BattingDataProcessor.class);
        job.setMapperClass(BattingDataProcessorMap.class);
        job.setReducerClass(BattingDataProcessorReduce.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(BattingWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        return job.waitForCompletion(true) ? 0 : 1;
    }

public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new BattingDataProcessor(),
args);
        System.exit(res);
    }

}
```

Task 8: Compile and run the MapReduce job

Activity Procedure

Step 1: Compile the program through Maven and create a package.

```
$ cd ~/Developer/heffalump
$ mvn package
```

Step 2: Run the MapReduce job.

```
$ hadoop jar $HOME/Developer/heffalump/target/heffalump-1.0.jar
com.hadooptraining.lab15.BattingDataProcessor input output
```

Activity Verification

As the job runs, you'll see familiar output on the screen.

Your output will look like:

```
$ hadoop fs -cat output/part-r-00000
1871   2659
1872   3390
1873   3580
1874   3470
1875   4234
1876   3066
1877   2040
1878   1904
1879   3409
1880   3191
...
```

Records for the last few years look like this:

```
...
1985   18216
1986   18545
1987   19883
1988   17380
1989   17405
1990   17919
1991   18127
1992   17341
```

1993	20864
1994	15752
1995	19554
1996	22831
1997	21604
1998	23297
1999	24691
2000	24971
2001	23199
2002	22408
2003	22978
2004	23376
2005	22325
2006	23599
2007	23322
2008	22585
2009	22419
2010	21308
2011	20808
2012	21017

Can you now answer the question about trends in the last few years? In which year was the highest cumulative score made in recent history?