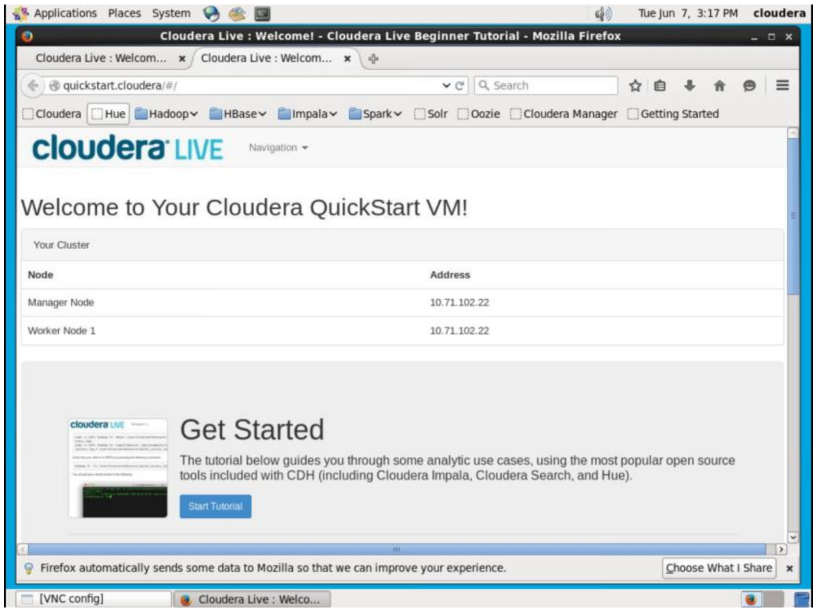
Tutorial MapReduce on Hadoop with Cloudera Quickstart Virtual Machine

Task: Connect to remote Cloudera Quickstart Virtual Machine (VM)

Open a web browser and go to https://hcc-anvil-175-11.unl.edu/guacamole/. Your instructor will hand you out your user name and password. Every student has its own virtual machine. This is the Quickstart VM provided by Clouerda. The Quickstart VM is a single-node pseudo distributed cluster that allows you to get to know Hadoop and related applications without having to create a fully distributed cluster. You should see this screen upon login.


Applications Places System
Tue Jun 7, 3:17 PM cloudera
Cloudera Live : Welcome! - Cloudera Live Beginner Tutorial - Mozilla Firefox
Cloudera Live : Welcom...
Cloudera Live : Welcom...
quickstart.cloudera/#/
Search
Cloudera Hue Hadoop HBase Impala Spark Solr Oozie Cloudera Manager Getting Started
cloudera LIVE
Navigation
Start Tutorial
Choose What I Share
[VNC config]
Cloudera Live : Welco...

Welcome to Your Cloudera QuickStart VM!

Your Cluster

Node	Address
Manager Node	10.71.102.22
Worker Node 1	10.71.102.22

Get Started

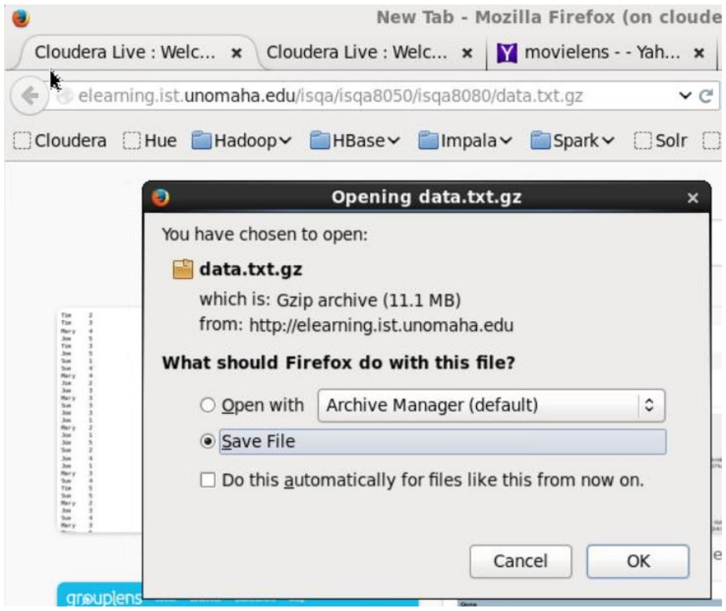
The tutorial below guides you through some analytic use cases, using the most popular open source tools included with CDH (including Cloudera Impala, Cloudera Search, and Hue).

Firefox automatically sends some data to Mozilla so that we can improve your experience.

All tasks below will require you to download tasks with a web browser. If you want to exchange text between your local machine and the VM you have to click on Ctr+Alt+Shift to open the Clipboard. Here you can also logout a session. Alternatively, you can simply close your browser.

Task: Copying a local file to HDFS

1. In the VM, download the file data.txt from http://elearning.ist.unomaha.edu/isqa/isqa8050/isqa8080/data.txt.gz to the Downloads folder in the VM. First, paste the link into a tab of Firefox. The data.txt file contains names similar to the example on the slides.



You have chosen to open:

data.txt.gz

which is: Gzip archive (11.1 MB)

from: http://elearning.ist.unomaha.edu

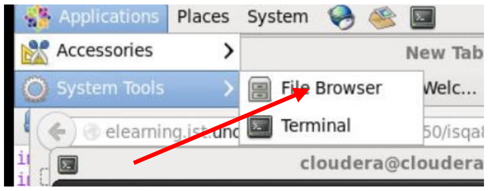
What should Firefox do with this file?

( ) Open with Archive Manager (default)

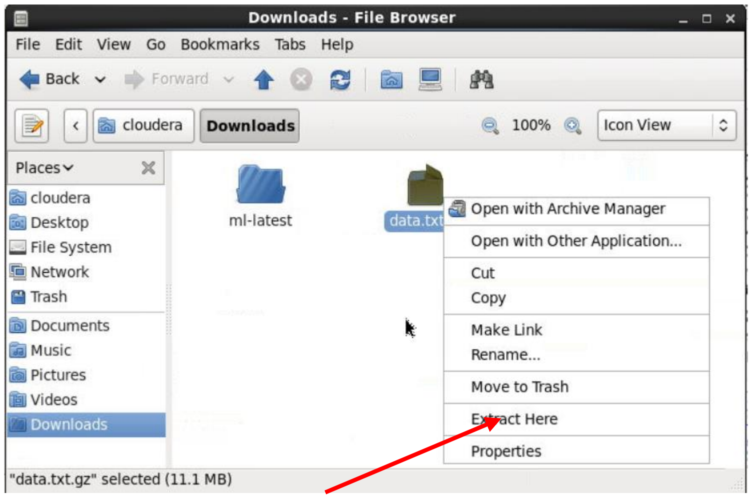
(●) Save File

☐ Do this automatically for files like this from now on.

2. Download the file to the *Downloads* folder (/cloudera/Downloads). This should be the default and you should only need to click on *Save*.
3. Open a File Browser Window and navigate to the Downloads folder.

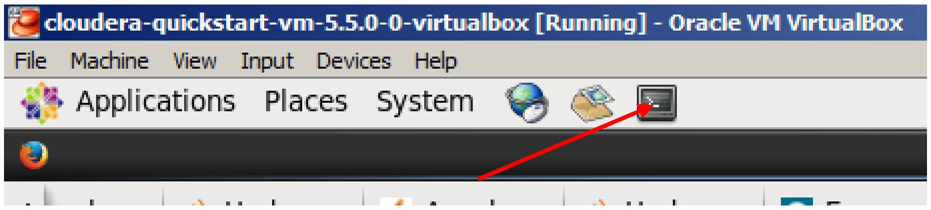


4. Right-click on the data.txt.gz file and extract it.

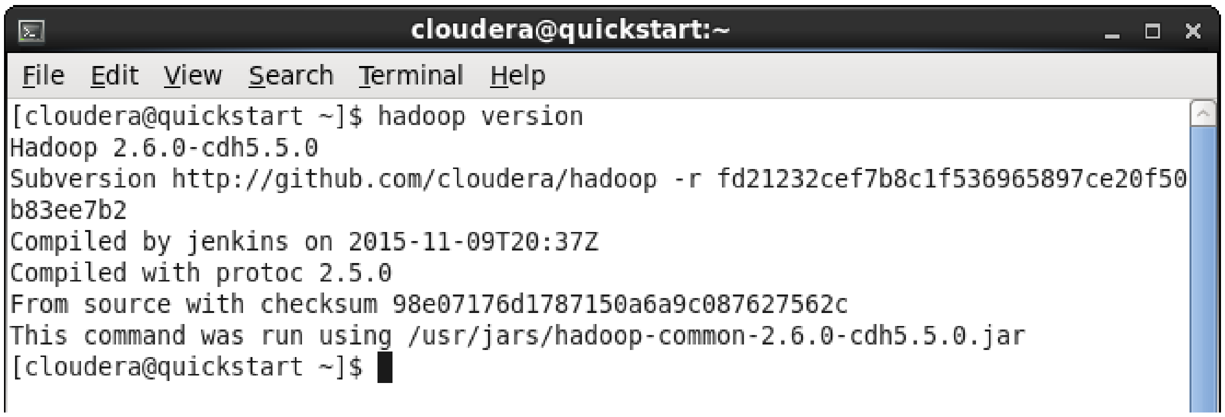


"data.txt.gz" selected (11.1 MB)

5. Open a Terminal Shell in the VM.


cloudera-quickstart-vm-5.5.0-0-virtualbox [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places System

6. First, let's explore some commands to work with files in Hadoop. For more commands, please review the documentation for the respective Hadoop version. You can see the Hadoop version by typing `hadoop version` into the Terminal Shell. The Cloudera version for this tutorial is Hadoop 2.6.0 (e.g. https://hadoop.apache.org/docs/r2.6.0/hadoop-project-dist/hadoop-common/FileSystemShell.html). File systems commands that work with the hdfs file system have the basic syntax `hadoop fs <args> or hdfs dfs <args>`. As far as I understand both are equivalent when working with Hadoop HDFS whereas `hdfs` is specific to hdfs and `hadoop fs` can work with other file systems. That is my current understanding. I will use `hdfs dfs <args>` in the following, but you may find `hadoop fs` online. So, don't be confused. `hadoop dfs`, that you also find in tutorials, seems to be deprecated, though.



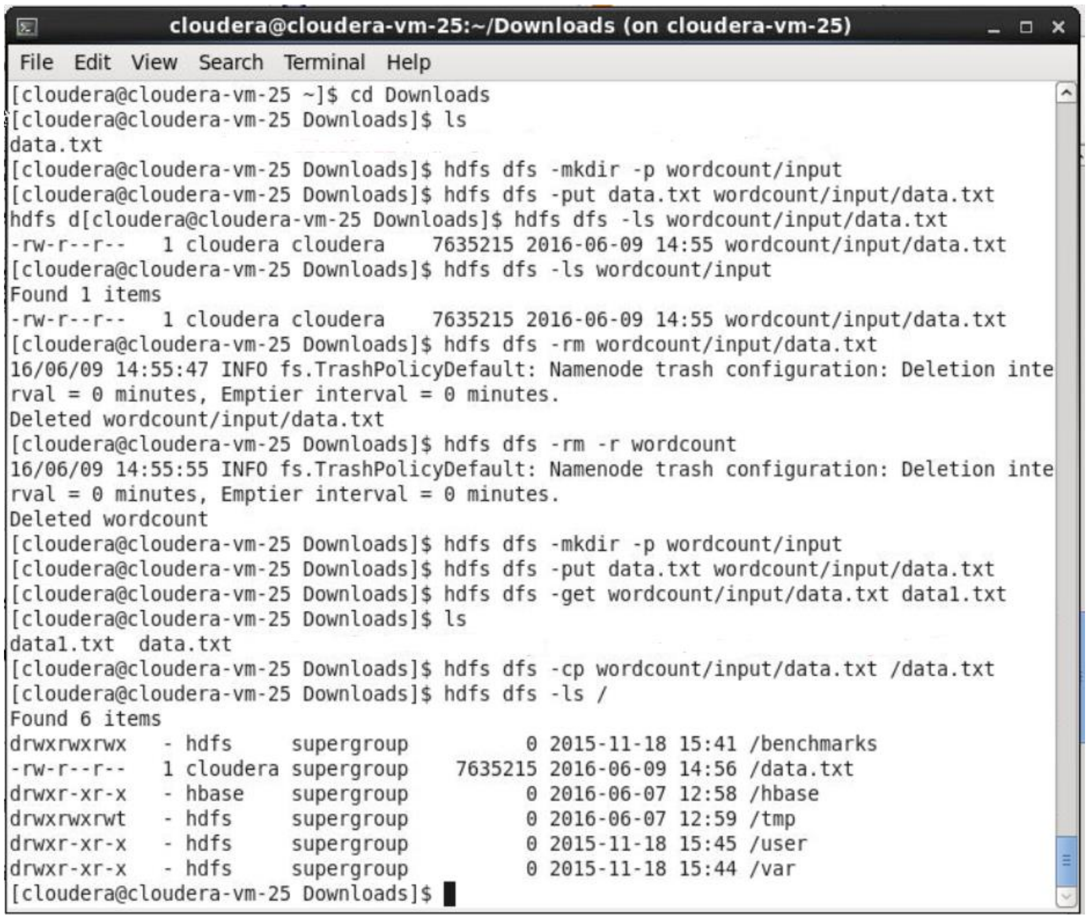
```
[cloudera@quickstart ~]$ hadoop version
Hadoop 2.6.0-cdh5.5.0
Subversion http://github.com/cloudera/hadoop -r fd21232cef7b8c1f536965897ce20f50
b83ee7b2
Compiled by jenkins on 2015-11-09T20:37Z
Compiled with protoc 2.5.0
From source with checksum 98e07176d1787150a6a9c087627562c
This command was run using /usr/jars/hadoop-common-2.6.0-cdh5.5.0.jar
[cloudera@quickstart ~]$ 
```

7. We first need to navigate to the data.txt file by changing the directory to Downloads in the local file system with `cd Downloads` and then verify that the file is present by typing `ls`.
8. Before we copy the data.txt file to HDFS, we want to create a directory called *wordcount* with sub-directory *input* and put the file in this directory. This can be done with *mkdir*. Please type in `hdfs dfs -mkdir -p wordcount/input`[1]. This creates the directory wordcount with sub directory input under your user directory on hdfs (which is /user/cloudera) (The -p option is needed to create the parent directory wordcount but can be left out if only one directory is created). It can take a couple of seconds, so please be patient.
9. We copy the data.txt file from the local file system to hdfs by using *put* (*copyFromLocal* could be used as well). Please type in `hdfs dfs -put data.txt wordcount/input/data.txt`. (Make sure you have the first forward slash (/) before wordcount. If not, you will get error messages that the directory cannot be found).
10. We can use *ls* to verify that it worked. We use `hdfs dfs -ls wordcount/input/data.txt`. You can also only use `hdfs dfs -ls wordcount/input` to have all files in the input directory listed.
11. Some other useful commands are to delete the files and directories. This is done with *rm*. So, `hdfs dfs -rm wordcount/input/data.txt` will remove a single file. If you want to delete a folder with all content, you can use `hdfs dfs -rm -r wordcount`. Try it out and then create the directory again and copy the data.txt file to the input folder.

[1] Be careful when copying into your Terminal shell. There is a difference between - and –. Only the normal dash - will work with the commands. If you use the wrong commands, e.g., –put instead of -put you will get "-put : Unknown command." Error.

12. You can copy a file from hdfs to your local system by using *get* (or use *copyToLocal*). Copy the data.txt file to your local system by using `hdfs dfs -get wordcount/input/data.txt data1.txt`. Did it work? Check with `ls`.
13. Files can also be copied within HDFS. You can use `hdfs dfs -cp wordcount/input/data.txt /data.txt` to copy the data.txt file from the input directory to the root directory. Verify with `hdfs dfs –ls /`.

Now, we have the data.txt file in hdfs in the directory wordcount/input. The picture below shows the complete dialogue.

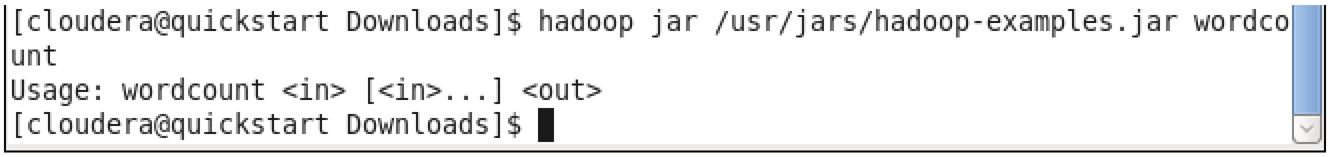

cloudera@cloudera-vm-25:~/Downloads (on cloudera-vm-25)
File Edit View Search Terminal Help

```
[cloudera@cloudera-vm-25 ~]$ cd Downloads
[cloudera@cloudera-vm-25 Downloads]$ ls
data.txt
[cloudera@cloudera-vm-25 Downloads]$ hdfs dfs -mkdir -p wordcount/input
[cloudera@cloudera-vm-25 Downloads]$ hdfs dfs -put data.txt wordcount/input/data.txt
hdfs d[cloudera@cloudera-vm-25 Downloads]$ hdfs dfs -ls wordcount/input/data.txt
-rw-r--r--   1 cloudera cloudera    7635215 2016-06-09 14:55 wordcount/input/data.txt
[cloudera@cloudera-vm-25 Downloads]$ hdfs dfs -ls wordcount/input
Found 1 items
-rw-r--r--   1 cloudera cloudera    7635215 2016-06-09 14:55 wordcount/input/data.txt
[cloudera@cloudera-vm-25 Downloads]$ hdfs dfs -rm wordcount/input/data.txt
16/06/09 14:55:47 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion inte
rval = 0 minutes, Emptier interval = 0 minutes.
Deleted wordcount/input/data.txt
[cloudera@cloudera-vm-25 Downloads]$ hdfs dfs -rm -r wordcount
16/06/09 14:55:55 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion inte
rval = 0 minutes, Emptier interval = 0 minutes.
Deleted wordcount
[cloudera@cloudera-vm-25 Downloads]$ hdfs dfs -mkdir -p wordcount/input
[cloudera@cloudera-vm-25 Downloads]$ hdfs dfs -put data.txt wordcount/input/data.txt
[cloudera@cloudera-vm-25 Downloads]$ hdfs dfs -get wordcount/input/data.txt data1.txt
[cloudera@cloudera-vm-25 Downloads]$ ls
data1.txt  data.txt
[cloudera@cloudera-vm-25 Downloads]$ hdfs dfs -cp wordcount/input/data.txt /data.txt
[cloudera@cloudera-vm-25 Downloads]$ hdfs dfs -ls /
Found 6 items
drwxrwxrwx   - hdfs     supergroup          0 2015-11-18 15:41 /benchmarks
-rw-r--r--   1 cloudera supergroup    7635215 2016-06-09 14:56 /data.txt
drwxr-xr-x   - hbase    supergroup          0 2016-06-07 12:58 /hbase
drwxrwxrwt   - hdfs     supergroup          0 2016-06-07 12:59 /tmp
drwxr-xr-x   - hdfs     supergroup          0 2015-11-18 15:45 /user
drwxr-xr-x   - hdfs     supergroup          0 2015-11-18 15:44 /var
[cloudera@cloudera-vm-25 Downloads]$ 
```

If you want to check the files you uploaded, you can use *hdfs fsck*. Type in `hdfs fsck /user/cloudera/wordcount/input -files -blocks` to see some information about the files and blocks in the directory.

Task: Run the WordCount example

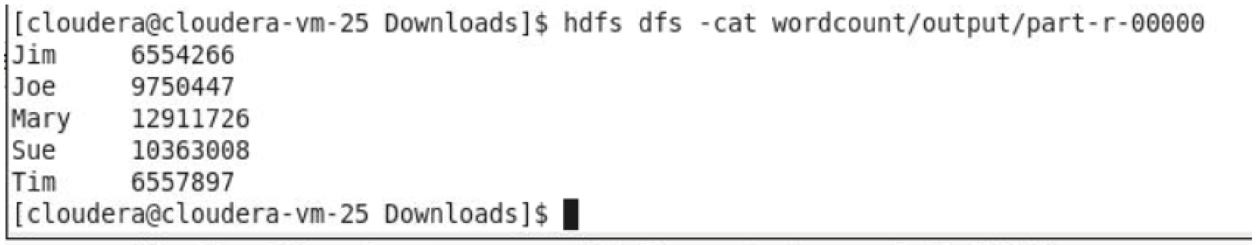
1. Hadoop comes with several example MapReduce jobs. You can see which one there are by typing in `hadoop jar /usr/jars/hadoop-examples.jar` in your Terminal shell. You will get a list of the examples. The wordcount example is the one we are looking for.
2. By entering `hadoop jar /usr/jars/hadoop-examples.jar wordcount` we can see what arguments the example needs. We need to specify an input file(s) (or directories) (<in> [<in>] <- the bracket shows optional multiple input paths possible) and an output directory (<out>).



```
[cloudera@quickstart Downloads]$ hadoop jar /usr/jars/hadoop-examples.jar wordco
unt
Usage: wordcount <in> [<in>...] <out>
[cloudera@quickstart Downloads]$ 
```

3. The following command will run wordcount over all input files in the input folder we copied into wordcount/input in the previous task. You could also specify a specific input file, of course. Type in `hadoop jar /usr/jars/hadoop-examples.jar wordcount wordcount/input/ wordcount/output/`. This will take a while. Hadoop will print out the progress to the terminal shell. Explanation:
    - hadoop – run in hadoop
    - jar – run a jar file
    - /usr/jars/hadoop=examples.jar – specifies which jar to run
    - wordcount – specifies which class in the jar to run
    - wordcount/input – specify the input path (s) – here a directory, but could be specific file or files or multiple directories
    - wordcount/output – specifies the output directory
    - Note: File names are case sensitive, so there is difference between wordcount and WordCount and Wordcount

4. Review the output that hadoop produces. You see that one input path was processed (one directory wordcount/input) and one split (data.txt is small enough to fit in one block). We can see that progress of the map and reduce tasks. The reduce task did not start before the map task. Under job counters we see that three map task was launched and one reduce task.
5. Once Hadoop is done, we can see what output it produced. Type in `hdfs dfs -ls wordcount/output` to see what files were generated. The file part-r-0000 has the result. We can view the file with the command `hdfs dfs -cat wordcount/output/part-r-00000`. You can also copy the file to your local file system and then use the file browser (Applications -> System Tools -> File Browser) to view the file.



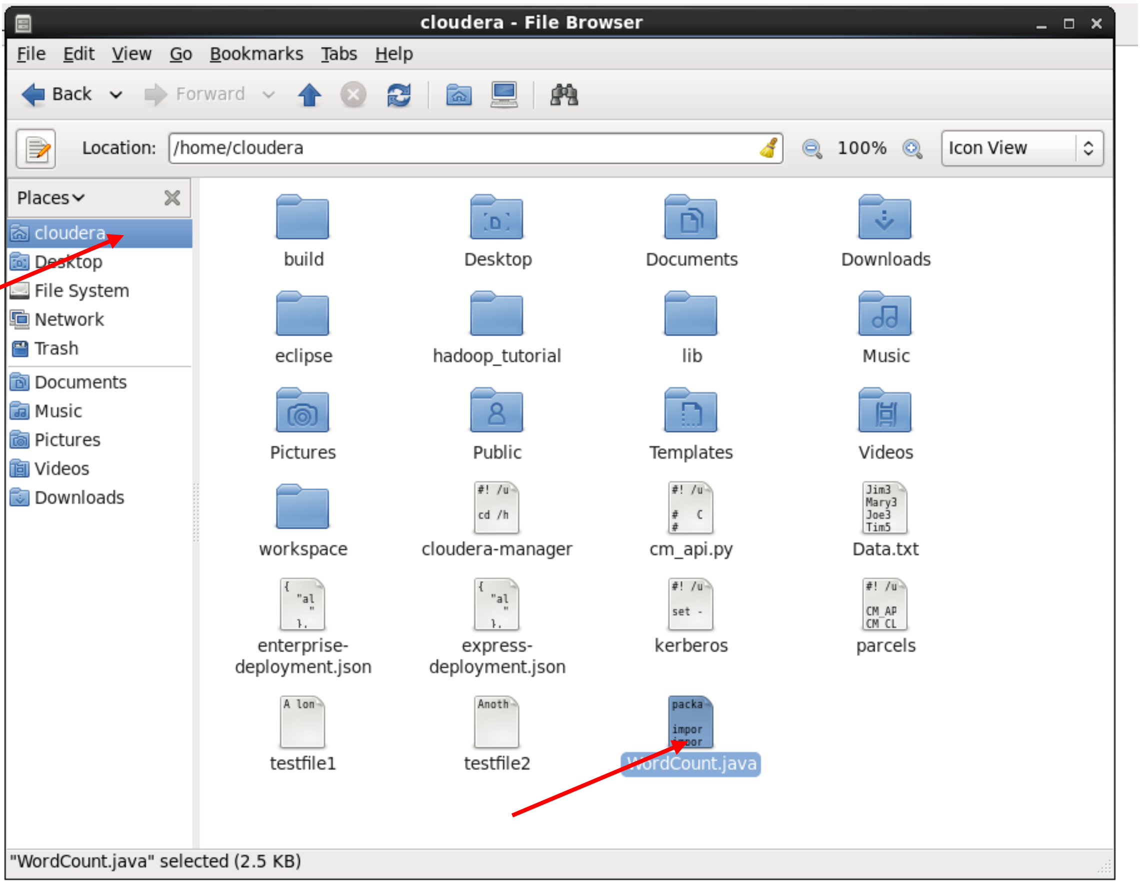
```
[cloudera@cloudera-vm-25 Downloads]$ hdfs dfs -cat wordcount/output/part-r-00000
Jim     6554266
Joe     9750447
Mary    12911726
Sue     10363008
Tim     6557897
[cloudera@cloudera-vm-25 Downloads]$ 
```

You ran your first MapReduce job. Let's create a MapReduce job of our own based on the WordCount example.

Task: Modify the WordCount example – add a combiner

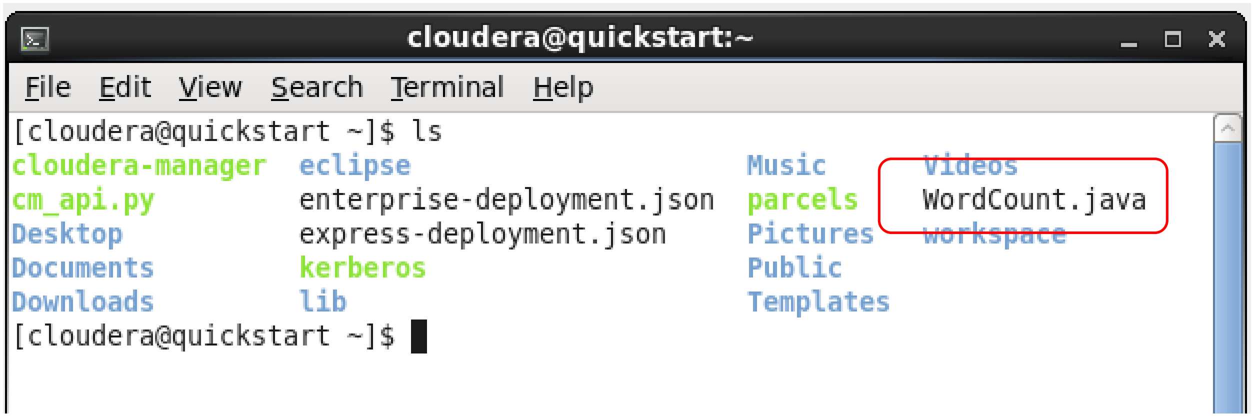
During the lecture, we learned that we can set a combiner that takes the keys and combines them into a pair of key and many values (e.g., Jim 1, Jim 1, Jim 1 -> Jim [1, 1, 1] before passing the key-value to the reducer. Doing so should reduce the time the MapReduce job takes. In this case, this is quite simple by only adding one line in the job configuration of the WordCount class. Of course, this means we need to have our own WordCount class that we can modify, compile, and then pack into a jar file to run on hadoop. The following is adapted from: http://www.cloudera.com/documentation/other/tutorial/CDH5/Hadoop-Tutorial.html

1. Copy the file WordCount.java from http://elearning.ist.unomaha.edu/isqa/isqa8050/isqa8080/wordcount.java.gz to you Downloads folder.
2. Open a file browser (Applications -> System Tools -> File Browser) and navigate to the Downloads folder.
3. Right-click on the file and choose to *Extract Here*. Right-click on the extracted file WordCount.java and copy and paste it into the cloudera directory.



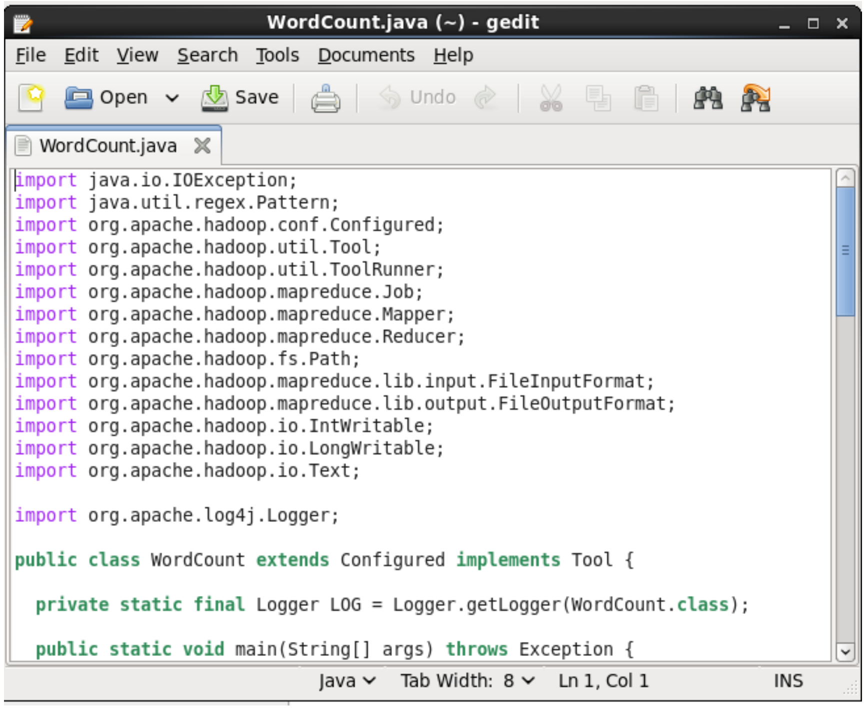
"WordCount.java" selected (2.5 KB)

4. Open a Terminal shell and enter `ls`. You should see WordCount.java in the cloudera directory.



```
[cloudera@quickstart ~]$ ls
cloudera-manager   eclipse                     Music      Videos
cm_api.py          enterprise-deployment.json  parcels    WordCount.java
Desktop            express-deployment.json     Pictures   workspace
Documents          kerberos                    Public
Downloads          lib                         Templates
[cloudera@quickstart ~]$
```

5. Type in `gedit WordCount.java`. This will open a text editor to view WordCount.java. An explanation of the file can be found here: http://www.cloudera.com/documentation/other/tutorial/CDH5/Hadoop-Tutorial/ht_wordcount1_source.html.



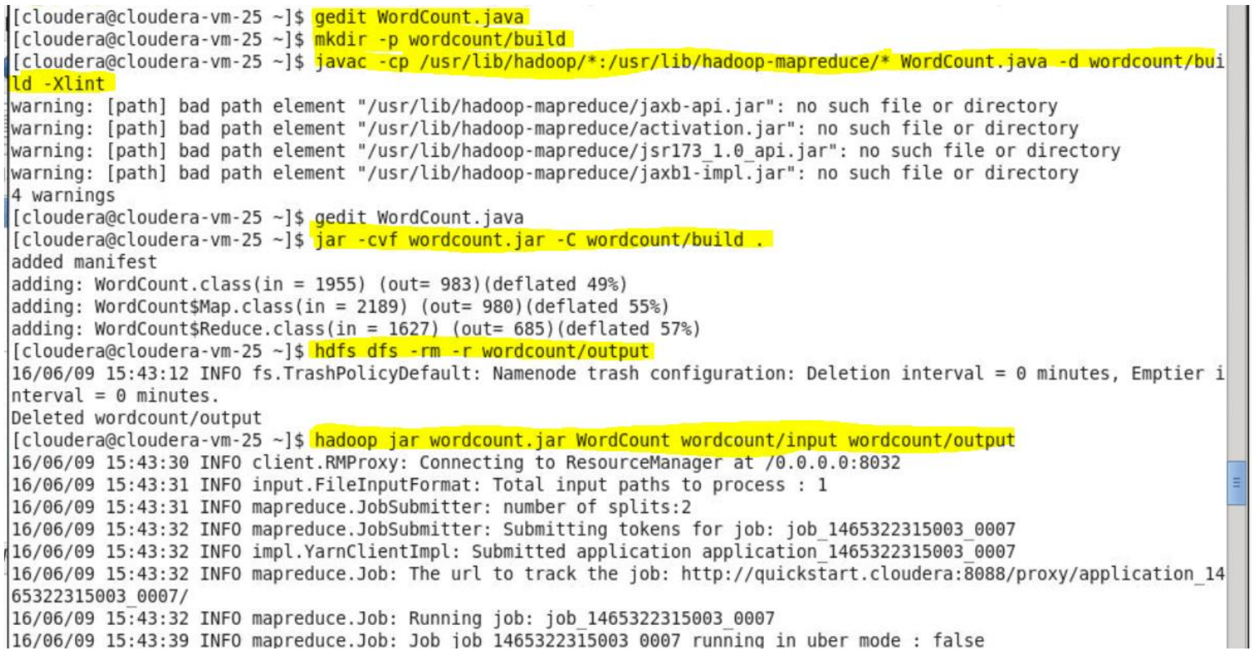
WordCount.java (~) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
WordCount.java
Java
Tab Width: 8
Ln 1, Col 1
INS

```
import java.io.IOException;
import java.util.regex.Pattern;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

import org.apache.log4j.Logger;

public class WordCount extends Configured implements Tool {

  private static final Logger LOG = Logger.getLogger(WordCount.class);

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

6. We are not changing anything yet, but first learn how to compile the class and create and run a jar file. So, close the text editor for now.
7. We need to compile the WordCount class. Type in `mkdir -p wordcount/build` to create a new directory build and then `javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/* WordCount.java -d wordcount/build -Xlint` to compile the class. You should receive four warnings, but no other errors. (WordCount.java is case sensitive, so type in WordCount.java not wordcount.java.)
8. Then, we create a jar file by entering `jar -cvf wordcount.jar -C wordcount/build .` (don't forget the last dot).
9. Before we run the WordCount example, we need to remove the output directory because hadoop will generate an error if we don't. Alternatively, you can specify a new output directory. So, we first type in `hdfs dfs -rm -r  wordcount/output` and then `hadoop jar wordcount.jar WordCount wordcount/input wordcount/output`. So, this time we use the jar file we created instead of the examples that hadoop comes with.

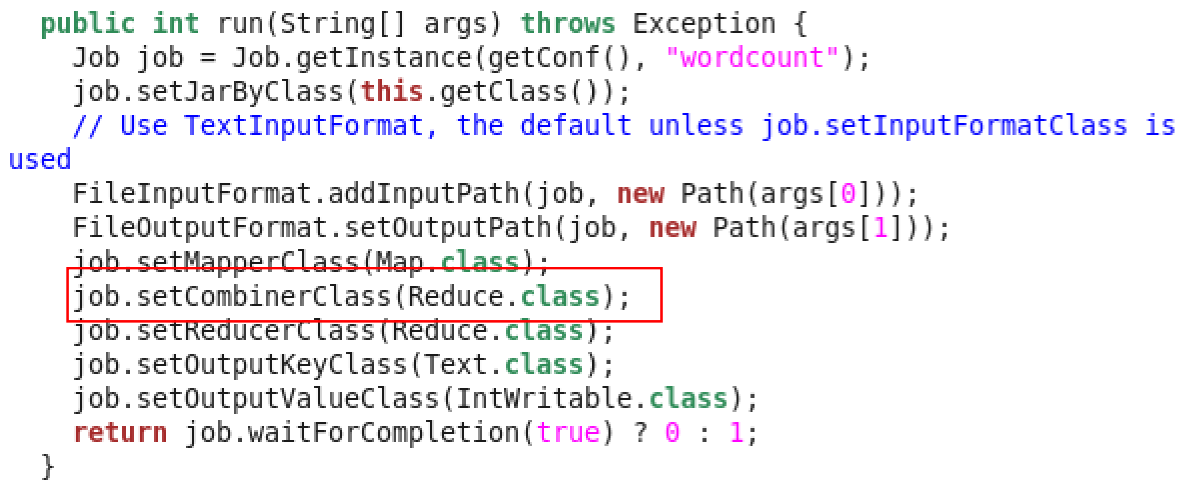
Explanation:

- hadoop jar wordcount.jar – run the wordcount.jar in hadoop
- wordcount/input – use the files in this directory as input files
- wordcount/output – write the output file(s) in this directory



```
[cloudera@cloudera-vm-25 ~]$ gedit WordCount.java
[cloudera@cloudera-vm-25 ~]$ mkdir -p wordcount/build
[cloudera@cloudera-vm-25 ~]$ javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/* WordCount.java -d wordcount/bui
ld -Xlint
warning: [path] bad path element "/usr/lib/hadoop-mapreduce/jaxb-api.jar": no such file or directory
warning: [path] bad path element "/usr/lib/hadoop-mapreduce/activation.jar": no such file or directory
warning: [path] bad path element "/usr/lib/hadoop-mapreduce/jsr173_1.0_api.jar": no such file or directory
warning: [path] bad path element "/usr/lib/hadoop-mapreduce/jaxb1-impl.jar": no such file or directory
4 warnings
[cloudera@cloudera-vm-25 ~]$ gedit WordCount.java
[cloudera@cloudera-vm-25 ~]$ jar -cvf wordcount.jar -C wordcount/build .
added manifest
adding: WordCount.class(in = 1955) (out= 983)(deflated 49%)
adding: WordCount$Map.class(in = 2189) (out= 980)(deflated 55%)
adding: WordCount$Reduce.class(in = 1627) (out= 685)(deflated 57%)
[cloudera@cloudera-vm-25 ~]$ hdfs dfs -rm -r wordcount/output
16/06/09 15:43:12 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier i
nterval = 0 minutes.
Deleted wordcount/output
[cloudera@cloudera-vm-25 ~]$ hadoop jar wordcount.jar WordCount wordcount/input wordcount/output
16/06/09 15:43:30 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/06/09 15:43:31 INFO input.FileInputFormat: Total input paths to process : 1
16/06/09 15:43:31 INFO mapreduce.JobSubmitter: number of splits:2
16/06/09 15:43:32 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1465322315003_0007
16/06/09 15:43:32 INFO impl.YarnClientImpl: Submitted application application_1465322315003_0007
16/06/09 15:43:32 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_14
65322315003_0007/
16/06/09 15:43:32 INFO mapreduce.Job: Running job: job_1465322315003_0007
16/06/09 15:43:39 INFO mapreduce.Job: Job job_1465322315003_0007 running in uber mode : false
```

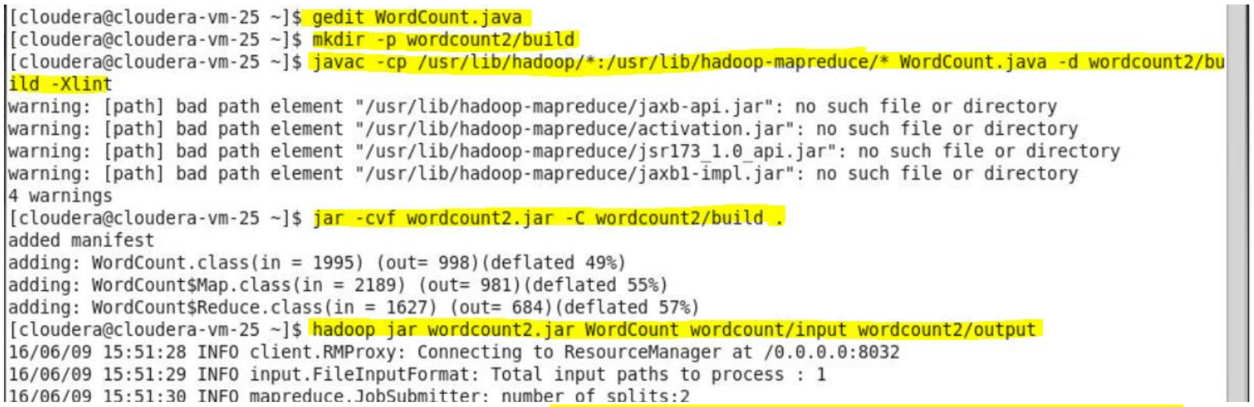
10. View the output file by entering `hdfs dfs -cat wordcount/output/part-r-00000`. It should be the same as before.
11. Let’s rebuild the jar file but with a slightly modified WordCount.java class. Open the WordCount.java file with gedit `gedit WordCount.java`. (This is case sensitive, if you type in gedit wordcount.java you open an empty file.)
12. The first thing we want to change is that we want to see what happens if we add a combiner. We need to add the line job.setCombinerClass(Reduce.class) in the run method. See picture below.



```
  public int run(String[] args) throws Exception {
    Job job = Job.getInstance(getConf(), "wordcount");
    job.setJarByClass(this.getClass());
    // Use TextInputFormat, the default unless job.setInputFormatClass is
used
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    return job.waitForCompletion(true) ? 0 : 1;
  }
```

13. Save and close the class.

14. We need to compile the WordCount class. Type in `mkdir -p wordcount2/build` to create a new directory build and then `javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/* WordCount.java -d wordcount2/build -Xlint` to compile the class. You should receive four warnings, but no other errors. If you receive errors, open the WordCount2 file again and make adjustments until no error messages (only the warnings) appear.
15. Then, we create a jar file by entering `jar -cvf wordcount2.jar -C wordcount2/build .` (don't forget the last dot).



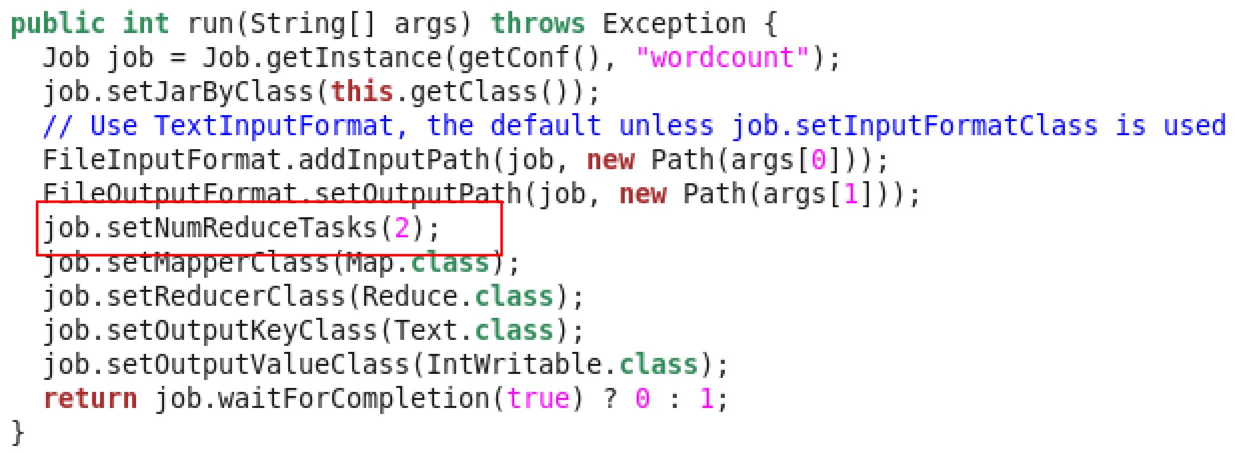
```
[cloudera@cloudera-vm-25 ~]$ gedit WordCount.java
[cloudera@cloudera-vm-25 ~]$ mkdir -p wordcount2/build
[cloudera@cloudera-vm-25 ~]$ javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/* WordCount.java -d wordcount2/bu
ild -Xlint
warning: [path] bad path element "/usr/lib/hadoop-mapreduce/jaxb-api.jar": no such file or directory
warning: [path] bad path element "/usr/lib/hadoop-mapreduce/activation.jar": no such file or directory
warning: [path] bad path element "/usr/lib/hadoop-mapreduce/jsr173_1.0_api.jar": no such file or directory
warning: [path] bad path element "/usr/lib/hadoop-mapreduce/jaxb1-impl.jar": no such file or directory
4 warnings
[cloudera@cloudera-vm-25 ~]$ jar -cvf wordcount2.jar -C wordcount2/build .
added manifest
adding: WordCount.class(in = 1995) (out= 998)(deflated 49%)
adding: WordCount$Map.class(in = 2189) (out= 981)(deflated 55%)
adding: WordCount$Reduce.class(in = 1627) (out= 684)(deflated 57%)
[cloudera@cloudera-vm-25 ~]$ hadoop jar wordcount2.jar WordCount wordcount/input wordcount2/output
16/06/09 15:51:28 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/06/09 15:51:29 INFO input.FileInputFormat: Total input paths to process : 1
16/06/09 15:51:30 INFO mapreduce.JobSubmitter: number of splits:2
```

16. We run the Wordcount2 example by typing in `hadoop jar wordcount2.jar WordCount wordcount/input wordcount2/output`. Review the output. You should see that the job ran faster than before, at least for the reduce part.
17. View the output file by entering `hdfs dfs -cat wordcount2/output/part-r-00000`.

Task: Modify the WordCount example – change the number of reducer tasks

When you look at the output, you notice that one reducer task is called. In the example in the lecture, we did have several reducers. We change the number of reducers to two in our next example.

1. Open a text editor to view WordCount.java by typing in `gedit WordCount.java`.
2. The first thing we want to change is that we want to see what happens if we use not one but two reduce jobs. We need to add the line job.setNumReduceTasks(2); in the run method.

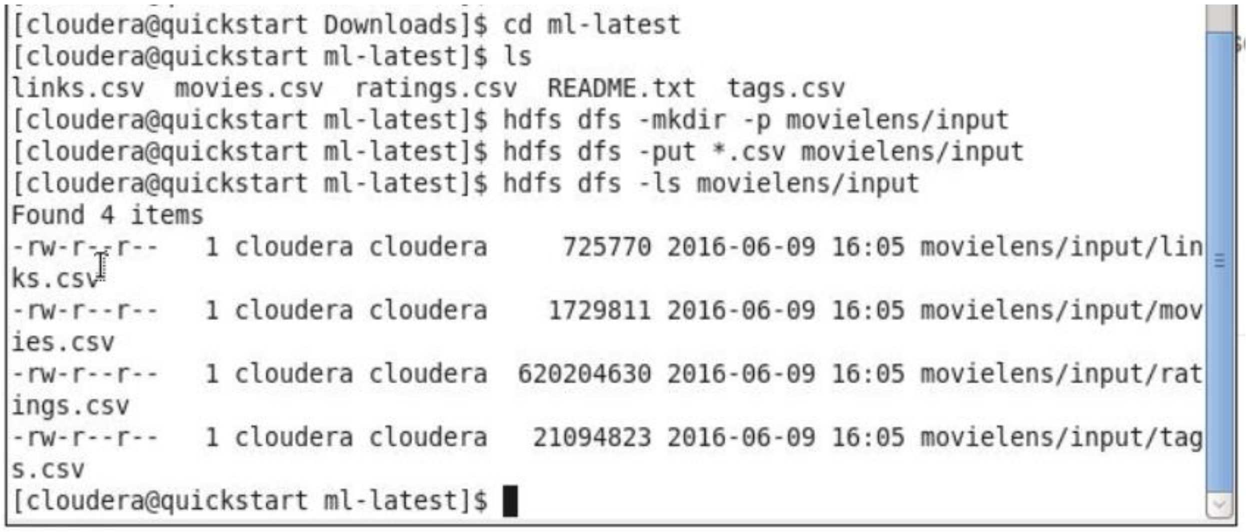


```
public int run(String[] args) throws Exception {
  Job job = Job.getInstance(getConf(), "wordcount");
  job.setJarByClass(this.getClass());
  // Use TextInputFormat, the default unless job.setInputFormatClass is used
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  job.setNumReduceTasks(2);
  job.setMapperClass(Map.class);
  job.setReducerClass(Reduce.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  return job.waitForCompletion(true) ? 0 : 1;
}
```

3. Save and close the class.
4. We need to compile the WordCount3 class. Type in `mkdir -p wordcount3/build` to create a new directory build and then `javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/* WordCount.java -d wordcount3/build -Xlint` to compile the class. You should receive four warnings, but no other errors. If you receive errors, open the WordCount2 file again and make adjustments until no error messages (only the warnings) appear.
5. Then, we create a jar file by entering `jar -cvf wordcount3.jar -C wordcount3/build .` (don't forget the last dot).
6. We run the Wordcount2 example by typing in `hadoop jar wordcount3.jar WordCount wordcount/input wordcount3/output`
7. View the output directory by entering `hdfs dfs -ls wordcount3/output`. You should see two files, one generated by each reducer. Look at both by typing in `hdfs dfs -cat wordcount3/output/part-r-00000` and `hdfs dfs -cat wordcount3/output/part-r-00001`. You can see that each reducer wrote one output file.

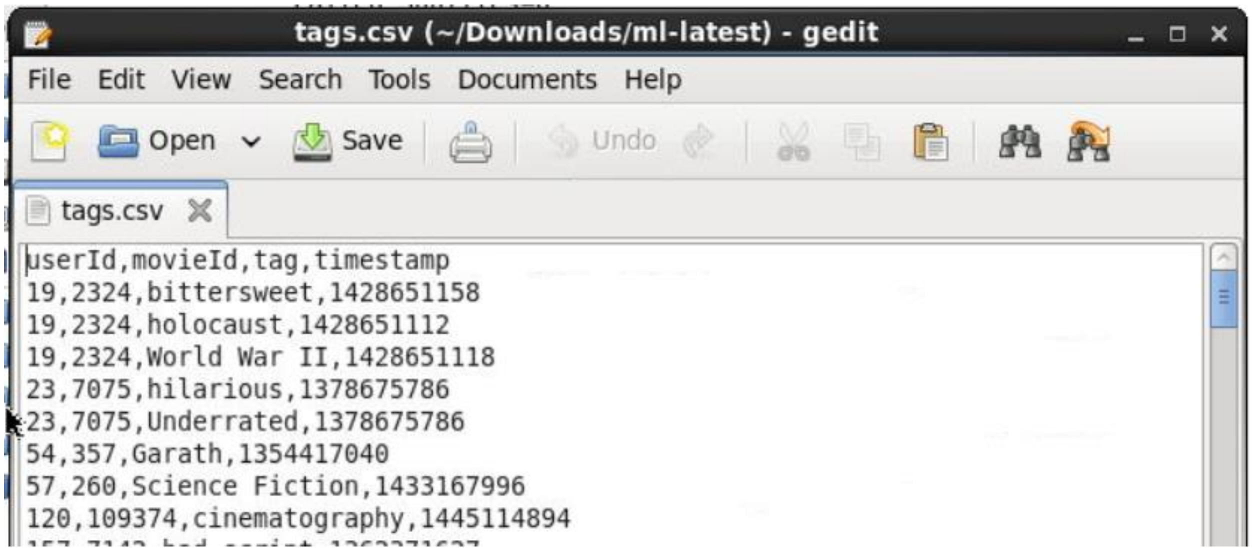
Task: MovieLens DataSet

Let's use the data from the Movielens example. Go to http://grouplens.org/datasets/movielens/ and download the ml-latest.zip file to the Downloads directory. Extract the zip file. In the folder ml-latest you should see four csv files. Make a new directory in Hadoop hdfs with `hdfs dfs -mkdir -p movielens/input`. Load the .csv file into Hadoop `hdfs with hdfs dfs -put *csv movielens/input`. Don't forget that you need to be in the folder ml-latest in order to use the put command. Check that you successfully loaded the files.



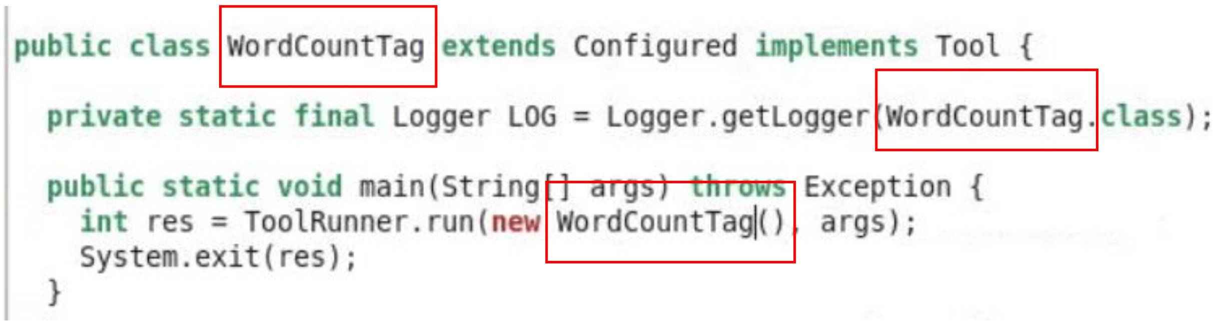
```
[cloudera@quickstart Downloads]$ cd ml-latest
[cloudera@quickstart ml-latest]$ ls
links.csv  movies.csv  ratings.csv  README.txt  tags.csv
[cloudera@quickstart ml-latest]$ hdfs dfs -mkdir -p movielens/input
[cloudera@quickstart ml-latest]$ hdfs dfs -put *.csv movielens/input
[cloudera@quickstart ml-latest]$ hdfs dfs -ls movielens/input
Found 4 items
-rw-r--r--   1 cloudera cloudera     725770 2016-06-09 16:05 movielens/input/lin
ks.csv
-rw-r--r--   1 cloudera cloudera    1729811 2016-06-09 16:05 movielens/input/mov
ies.csv
-rw-r--r--   1 cloudera cloudera  620204630 2016-06-09 16:05 movielens/input/rat
ings.csv
-rw-r--r--   1 cloudera cloudera   21094823 2016-06-09 16:05 movielens/input/tag
s.csv
[cloudera@quickstart ml-latest]$ 
```

Let's first use the WordCount example to find the most often used tags for movies. First, in your command shell navigate to the cloudera folder (you probably need to type in `cd ..` twice in order to navigate from the ml-latest to the cloudera folder. We need to copy the WordCount.java to WordCountTag.java (`cp WordCount.java WordCountTag.java`) and then open it with gedit. The tags.csv file consists userId, movieID, tag, timestamp in each line. The first line is the header.


tags.csv (~/Downloads/ml-latest) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
tags.csv

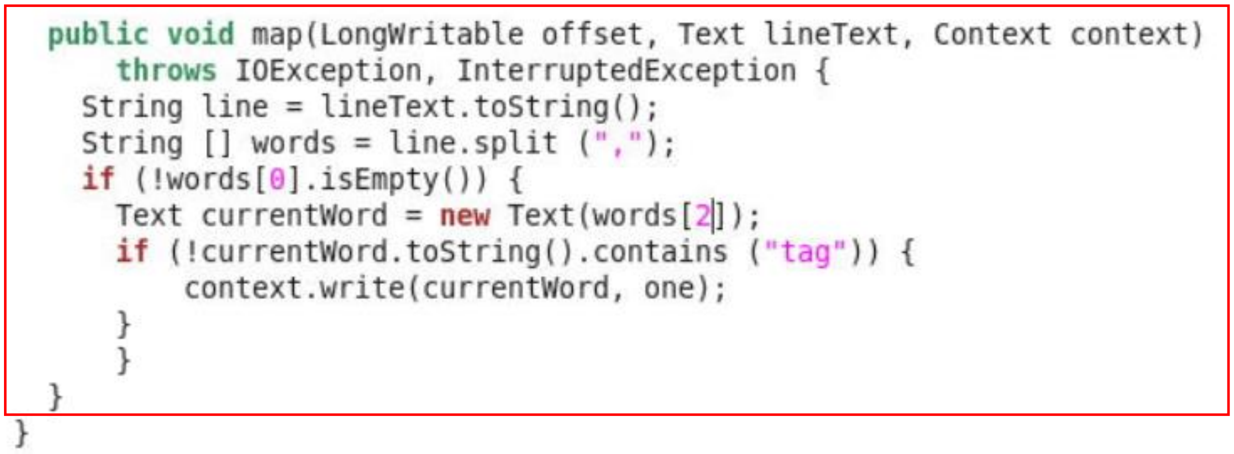
```
userId,movieId,tag,timestamp
19,2324,bittersweet,1428651158
19,2324,holocaust,1428651112
19,2324,World War II,1428651118
23,7075,hilarious,1378675786
23,7075,Underrated,1378675786
54,357,Garath,1354417040
57,260,Science Fiction,1433167996
120,109374,cinematography,1445114894
```

We first need to modify the Mapper so that it only reads the tags. What we also need to do is to change the class name in three places as seen below.



```
public class WordCountTag extends Configured implements Tool {

  private static final Logger LOG = Logger.getLogger(WordCountTag.class);

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

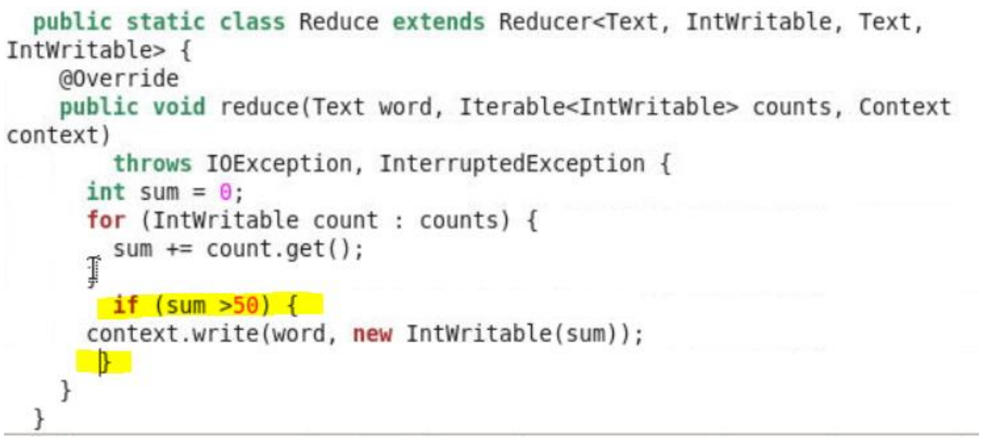
Then, modify the Mapper.



```
    public void map(LongWritable offset, Text lineText, Context context)
        throws IOException, InterruptedException {
      String line = lineText.toString();
      String [] words = line.split (",");
      if (!words[0].isEmpty()) {
        Text currentWord = new Text(words[2]);
        if (!currentWord.toString().contains ("tag")) {
            context.write(currentWord, one);
        }
        }
    }
  }
```

Again, save the file. Create a build directory with `mkdir -p wordcounttag/build` and compile with `javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/* WordCountTag.java -d wordcounttag/build -Xlint` to compile the class. You should receive four warnings, but no other errors. If you receive errors, open the Rating.java file again and make adjustments until no error messages (only the warnings) appear. Then, we create a jar file by entering `jar -cvf wordcounttag.jar -C wordcounttag/build .` (don't forget the last dot). We run the rating by typing in `hadoop jar wordcounttag.jar WordCountTag movielens/input/tags.csv movielens/output/wordcounttag`. View the output file by entering `hdfs dfs -cat movielens/output/wordcounttag/part-r-00000`. If there are more than one output file, review them.

There were quite a lot of different tags. Let's change the Reducer to only show words with more than 50 counts. Change the following:



```
  public static class Reduce extends Reducer<Text, IntWritable, Text,
IntWritable> {
    @Override
    public void reduce(Text word, Iterable<IntWritable> counts, Context
context)
        throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable count : counts) {
        sum += count.get();
      
       if (sum >50) {
      context.write(word, new IntWritable(sum));
       }
    }
  }
```

Build it, create the jar file, remove the output path, run it again in Hadoop. Review the output files.

Average Rating for each movie

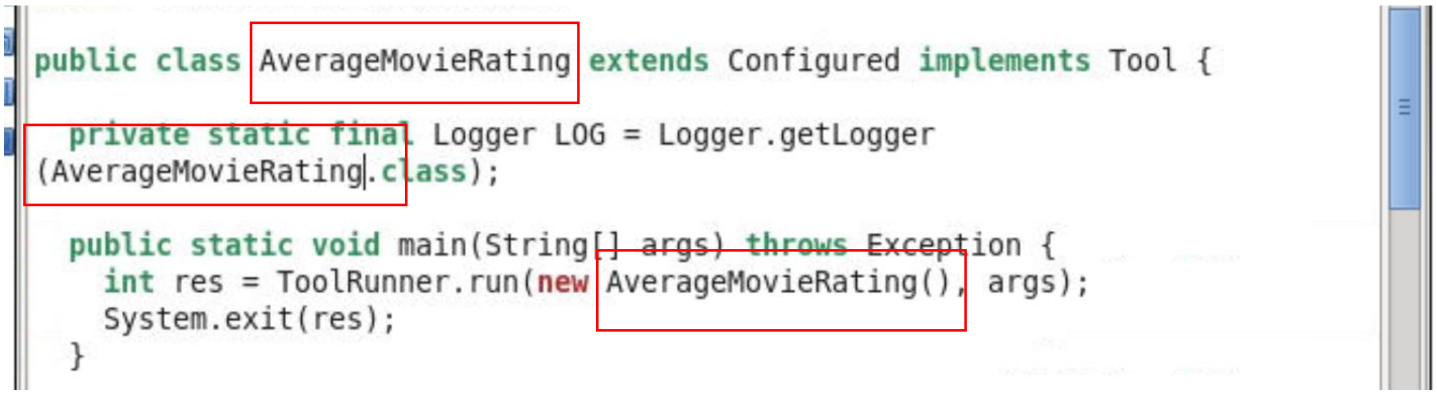
Next, let's calculate the average rating for each movie. This would correspond to a select statement: SELECT AVG(rating) FROM ratings GROUP BY movieID. First, we need to think about what key-value pairs we need. In the previous example, the word (text (i.e., a String)) was the key, and the variable one (defined as IntWritable (i.e., int) and set to 1) was the value. This allowed us to count the number of words. In order to get the average rating for each movie, we need to sum up all ratings for a movie and then divide it by the number of ratings. Since we need to do this for each movie, our key will be the movieID (a long), and the rating (double) will be the value.

<offset, line> ⇨ **map** ⇨ list (<movieID, rating>) ⇨ **combine** ⇨<movieID, list (<rating>) ⇨ **reduce** ⇨ <movieID, avRating>

Mapper: We need as input file the ratings.csv file. Each record (except the first one with the header) has the structure userId, movieId, rating, timestamp, separated by a comma. So, we need to split each line by a comma and then select the movie ID and the rating and write both as output.

Reducer: We get the list with ratings for a key and need to iterate through them, counting the number of ratings, and sum them up. Then, we need to calculate the average for the movie and write both as output.

So, let's copy the WordCountTag.java file. First, let's copy it into a new file with `cp WordCountTag.java AverageMovieRating.java`. Use gedit to open the AverageMovieRating.java file. First, let's change the class names as shown below.

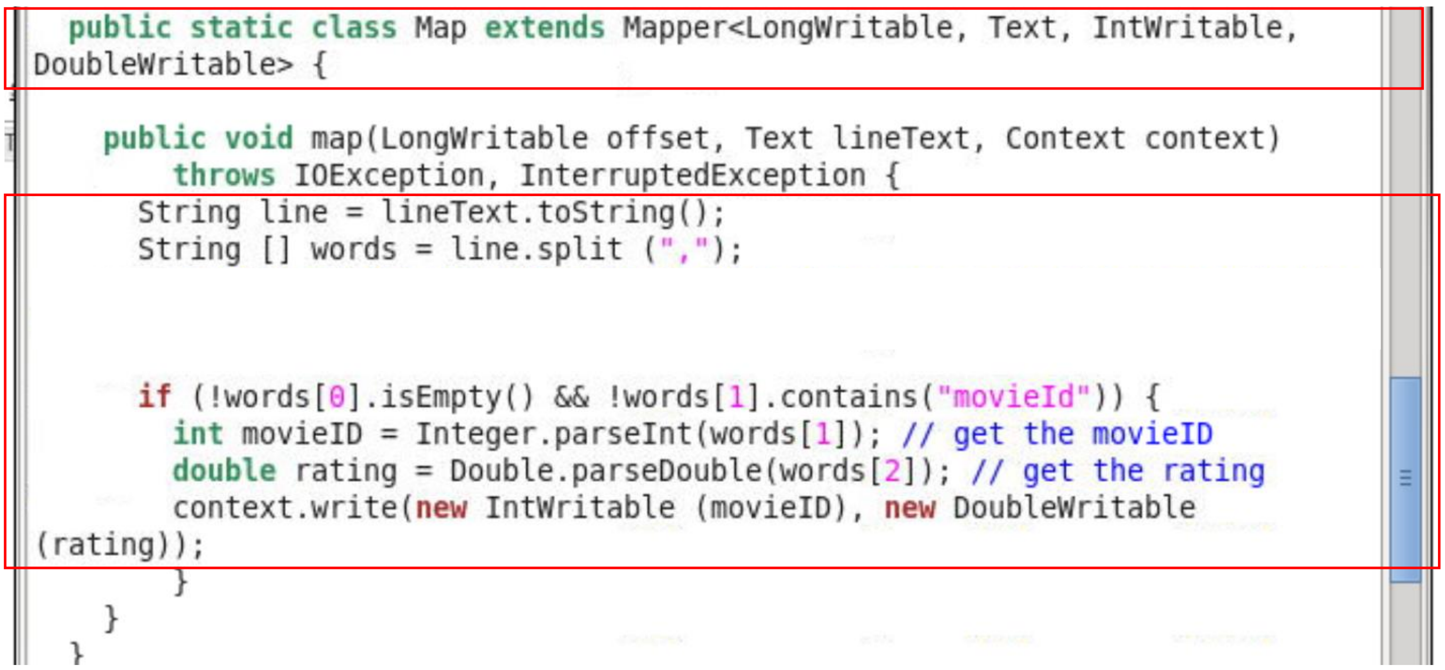


```
public class AverageMovieRating extends Configured implements Tool {

  private static final Logger LOG = Logger.getLogger
(AverageMovieRating.class);

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

Input and Output data type have changed, so we need to change the input and output classes in the run method as well as method signatures. Let's first revise the signatures. What classes are accepted? The table below shows the Java primitives and Writable Implementations.

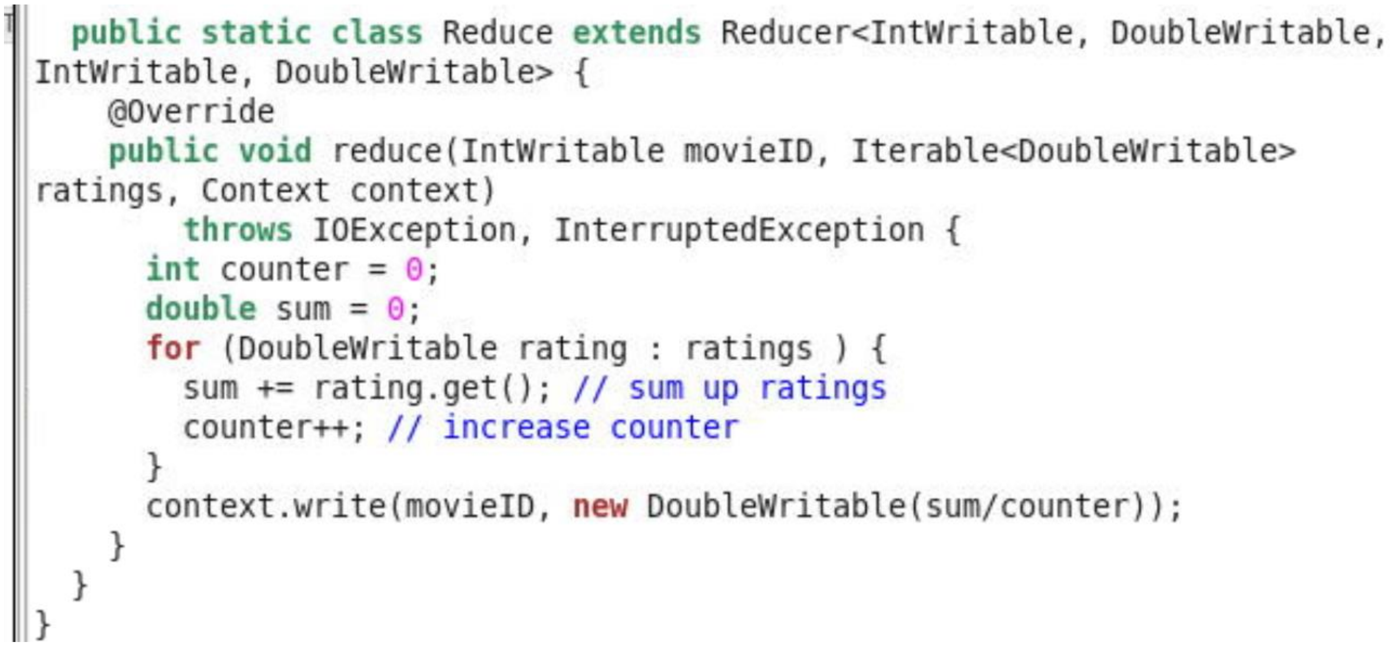
Java primitive	Writable implementation
String	Text
double	DoubleWritable
int	IntWritable
long	LongWritable
float	FloatWritable
byte	ByteWritable
boolean	BooleanWritable

The Mapper takes as input LongWritable and text (offset of line and line text) and as output IntWritable and DoubleWritable (movieID and rating). We can also remove some variables from the class such as WORD_BOUNDARY, word, numRecords, and one. The method will first split the input lineText into the components and then change the types from text to IntWritable and DoubleWritable. Make the changes as seen below. Work carefully – typos, missing punctuation, wrong upper and lower letters, etc. happen easily. Make sure to write contains("MovieId") instead of MovieID (last d is lower letter) – that was a typo that cost me some time to fix. !words[1].contains("movieId") makes sure that we do not read in the header.



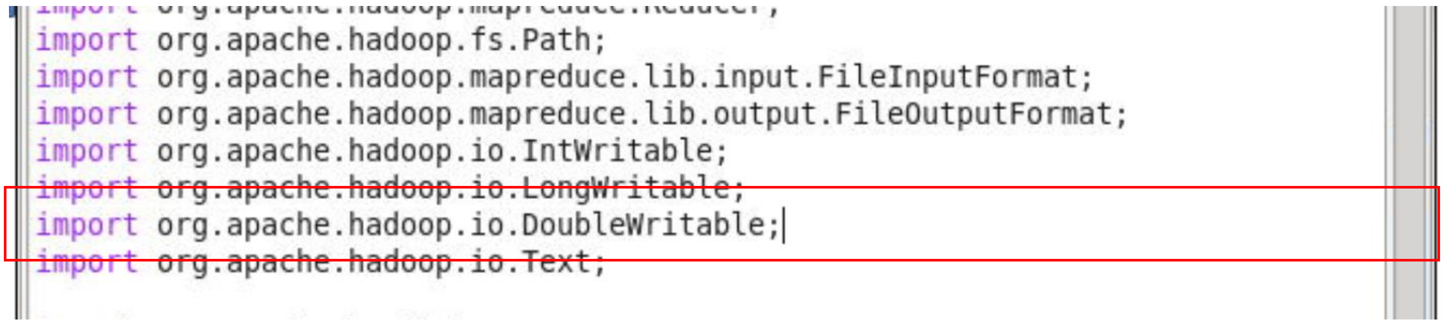
```
  public static class Map extends Mapper<LongWritable, Text, IntWritable,
DoubleWritable> {

    public void map(LongWritable offset, Text lineText, Context context)
        throws IOException, InterruptedException {
      String line = lineText.toString();
      String [] words = line.split (",");



      if (!words[0].isEmpty() && !words[1].contains("movieId")) {
        int movieID = Integer.parseInt(words[1]); // get the movieID
        double rating = Double.parseDouble(words[2]); // get the rating
        context.write(new IntWritable (movieID), new DoubleWritable
(rating));
        }
    }
  }
```

Now, let's move on to the Reducer. We need to change the input and output key-value classes of the Reduce class. Then, we need to change the input into the reduce method and the code. Make the changes as below. Since almost everything changes, I did not highlight it with a red box. Do not forget to change the input and output classes in the method signature.



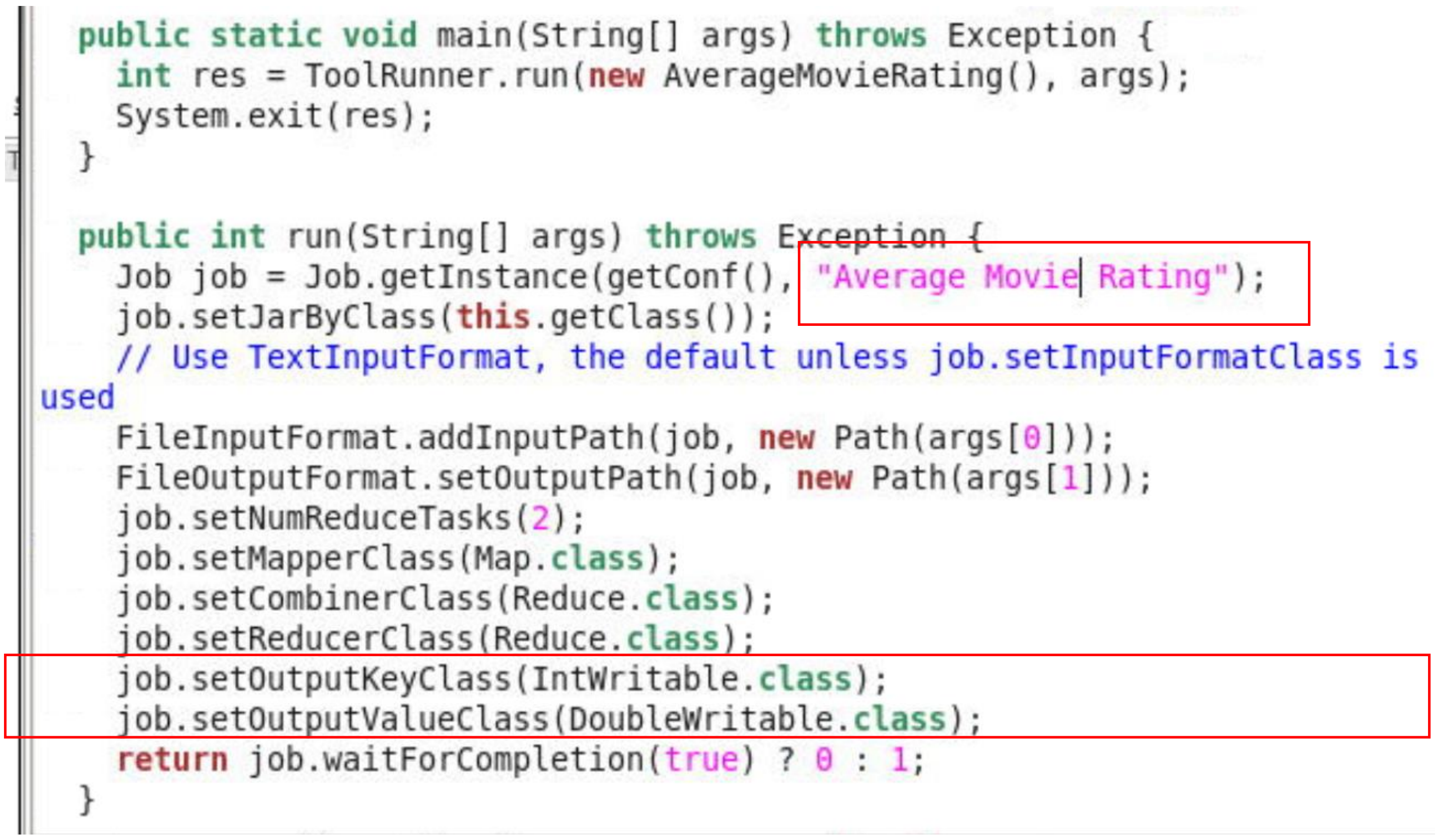
```
  public static class Reduce extends Reducer<IntWritable, DoubleWritable,
IntWritable, DoubleWritable> {
    @Override
    public void reduce(IntWritable movieID, Iterable<DoubleWritable>
ratings, Context context)
        throws IOException, InterruptedException {
      int counter = 0;
      double sum = 0;
      for (DoubleWritable rating : ratings ) {
        sum += rating.get(); // sum up ratings
        counter++; // increase counter
      }
      context.write(movieID, new DoubleWritable(sum/counter));
    }
  }
}
```

Since we use DoubleWritable we need to import it, so add an import statement at the beginning.



```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
```

The output key-value into the Mapper are defined by setMapOutputKeyClass and setMapOutputValueClass. You only need to set those, if they differ from the key-value output of the Reducer. The output key-value are defined with setOutputKeyClass and setOutputValueClass. Since the key-value classes (IntWritable, DoubleWritable) are the same for both output key-value pairs of Mapper and Reducer, we only need to define setOutputKeyClass and setOutputValueClass. We also need to give the Job a new name.

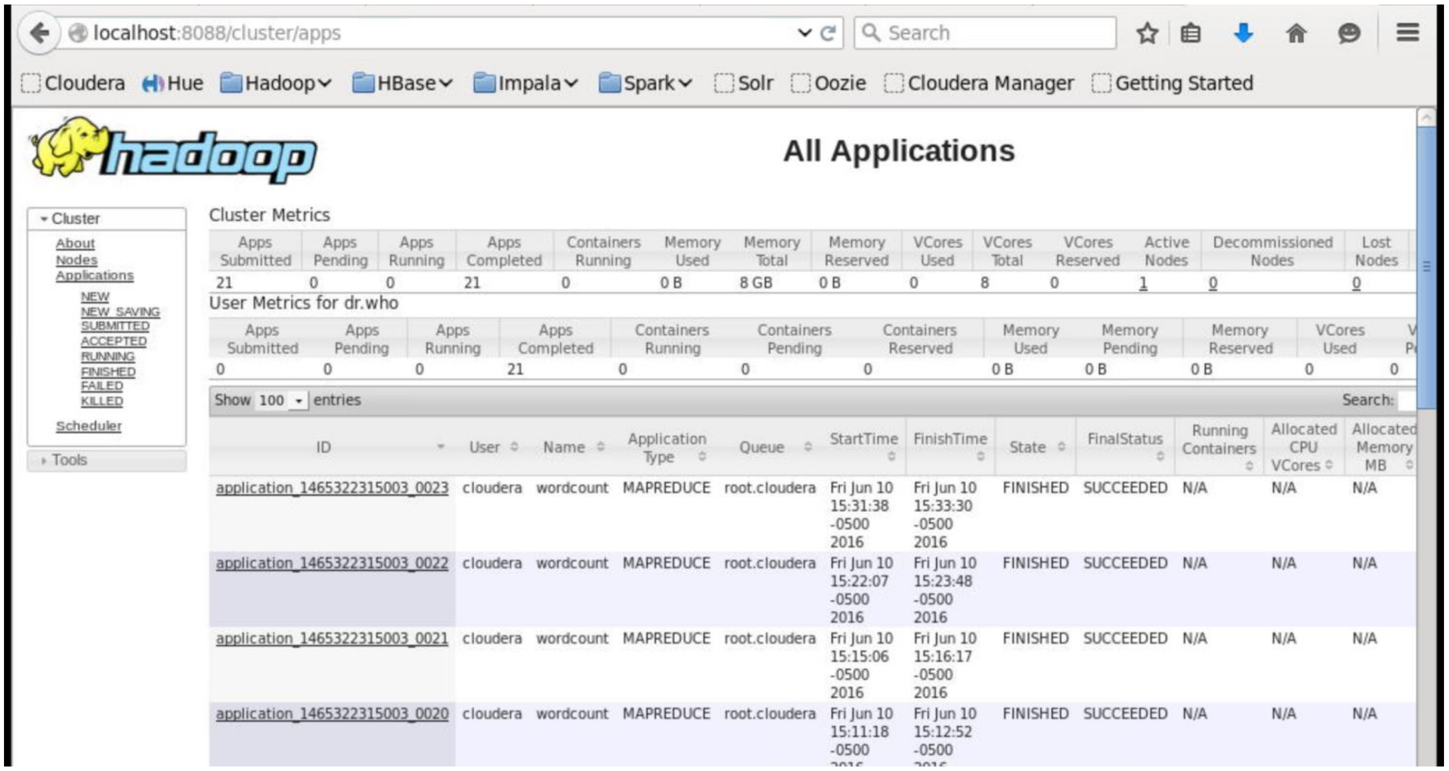


```
  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new AverageMovieRating(), args);
    System.exit(res);
  }

  public int run(String[] args) throws Exception {
    Job job = Job.getInstance(getConf(), "Average Movie Rating");
    job.setJarByClass(this.getClass());
    // Use TextInputFormat, the default unless job.setInputFormatClass is
used
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setNumReduceTasks(2);
    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(DoubleWritable.class);
    return job.waitForCompletion(true) ? 0 : 1;
  }
```

Now, we are ready to test it. Compile it, create the jar file, run it, check the output file.

Monitoring Jobs

We can monitor Map Reduce jobs via the MapReduce Web UI at localhost:8088/cluster. Click on All Applications and you should see the MapReduce jobs you ran.



All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes
21	0	0	21	0	0 B	8 GB	0 B	0	8	0	1	0	0

User Metrics for dr.who

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved	VCores Used	V P
0	0	0	21	0	0	0	0 B	0 B	0 B	0	0

Show 100 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB
application_1465322315003_0023	cloudera	wordcount	MAPREDUCE	root.cloudera	Fri Jun 10 15:31:38 -0500 2016	Fri Jun 10 15:33:30 -0500 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1465322315003_0022	cloudera	wordcount	MAPREDUCE	root.cloudera	Fri Jun 10 15:22:07 -0500 2016	Fri Jun 10 15:23:48 -0500 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1465322315003_0021	cloudera	wordcount	MAPREDUCE	root.cloudera	Fri Jun 10 15:15:06 -0500 2016	Fri Jun 10 15:16:17 -0500 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1465322315003_0020	cloudera	wordcount	MAPREDUCE	root.cloudera	Fri Jun 10 15:11:18 -0500	Fri Jun 10 15:12:52 -0500	FINISHED	SUCCEEDED	N/A	N/A	N/A

I hope you get an idea how the MapReduce framework is used in Hadoop. And you probably also got the idea that programming the classes can be quite complicated. So, we need both the conceptual understanding of what kind of map and reducer tasks we need and programming knowledge. Also, more complicated questions may need to require to chain MapReduce jobs together with the output of the reduce tasks being the input in the next map tasks.

Deliverable:

Your tasks:

Obviously, in order to do the deliverable, you should go through the examples above as they enable you to solve the following. You also need to have the AverageMovieRating.java file created. So, do not skip on the examples. We did most of them in the lab already.

1. Create a new .java file based on AverageMovieRating.java and name it with *your first name*, e.g., Martina.java.
2. Open the new .java file and change the Mapper and Reducer in a way that the MapReduce job will calculate the average rating done by each user instead of for each movie. Then, only the userID and average ratings of users with more than X ratings will be printed to the output file (X = the number of your virtual machine user name; e.g., if you VM user name is user16, then X = 16).
3. Name your job *your first name*, e.g., "Martina". (In your run method: Job job = Job.getInstance(getConf(), "Martina");)
4. Remove the job.setCombinerClass line in the run method. Your code might not work with it. It didn't for my code
5. Compile the file and create a jar file with *your first name*, e.g., martina.jar. If you get errors, debug. If you receive the four warnings, ignore them.
6. Run the jar file.
7. Read the output from the output file(s).
8. Make the required screenshots for your deliverable (see next page) and paste them in a word document.
9. Write a reflection to the discussion board (see next page).

Tip: Read the error message that the compiler or hadoop produces. They will likely tell you what's wrong. If you cannot figure it out, please post on the Learner-to-learner community discussion board on Blackboard.

Collaboration: You may ask for help and develop the code on the Learner-to-learner community discussion board on Blackboard. I do not expect everyone to have the Java skills to make the changes to the file. That's why you may share your code with others or ask for help debugging your code. Please share your code or questions on Blackboard, that way we have one common platform where we can help others and compile a list of common problems and their solutions.

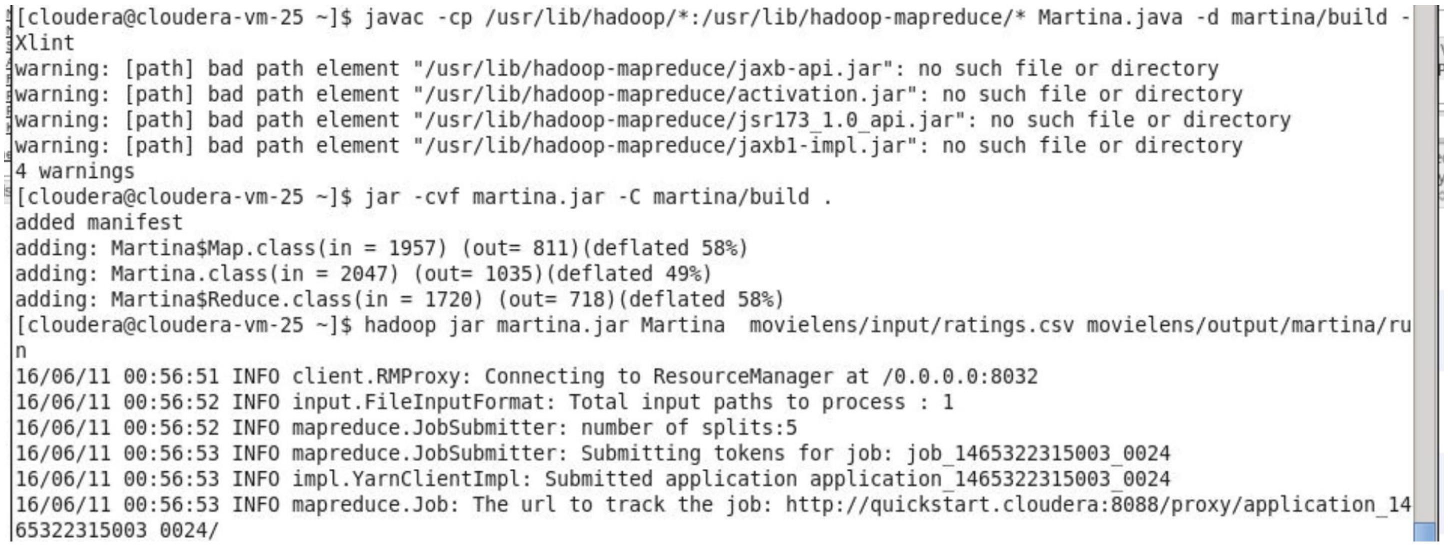
No Collaboration on the following: What you may not do is to write the changes to the .java file for someone else. You may not run the program for somebody else. I want every student to go through the task to make the changes, compile the Java file, create the jar file, and run the jar file.

What you should submit: (see next page)

What you should submit:

- The *yourFirstName*.java file with the updated Mapper and Reducer. Since the file is in your VM, you may want to email the file to yourself and then download it to your local machine. I am not aware of a better way to get files in or out of the VM other than downloading, uploading, or emailing them. Let me know if there is a better way. (45 points)
- A word document that contains a screenshot of each of the following commands together with the first couple of lines. An example of what kind of screenshot I expect is provided below. (45 points)
    - javac …
    - jar …
    - hadoop jar …
    - hdfs dfs -cat ... (show first and last 10 lines - | head -10 and | tail -10 allow you to do so)
    - The first screen of your job log on localhost:8088/cluster/apps
- Your 200-250 word reflection on the Reflection discussion board. (10 points)

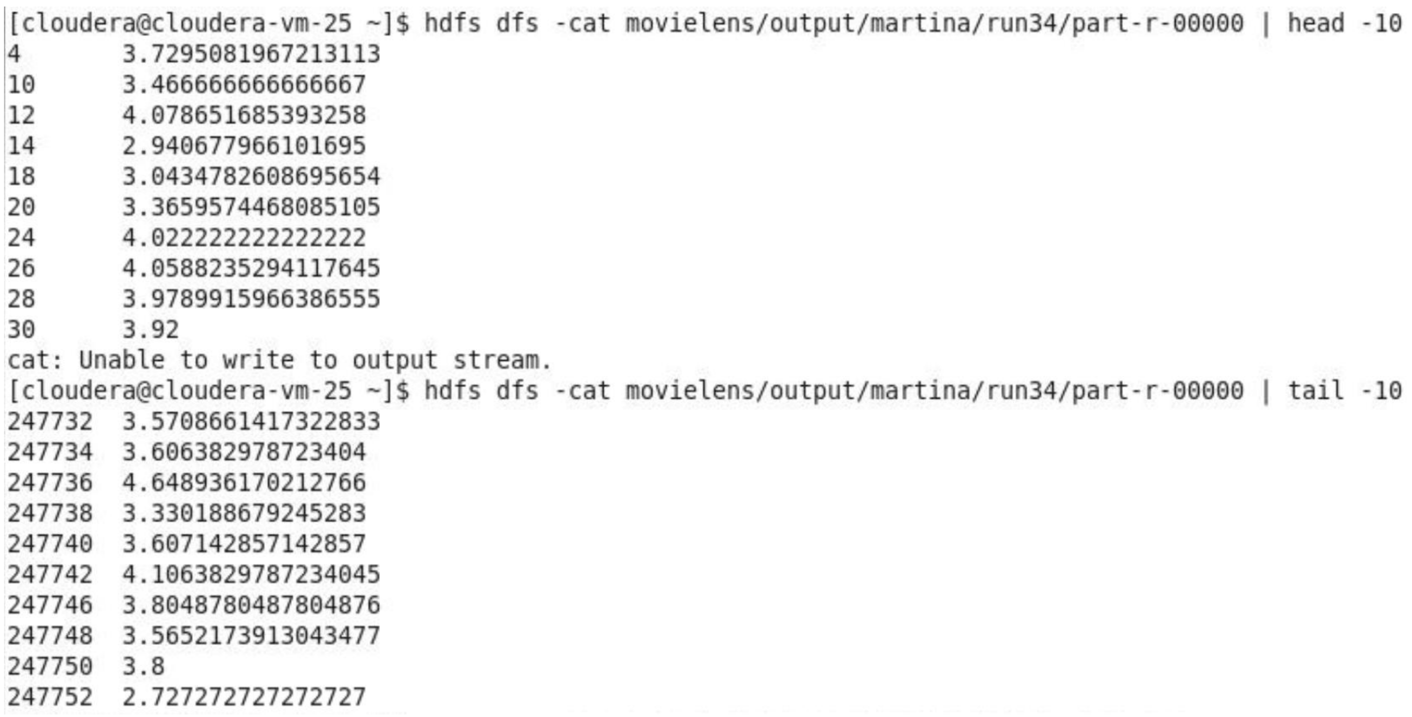
Example: Screenshot with the first three commands:



```
[cloudera@cloudera-vm-25 ~]$ javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/* Martina.java -d martina/build -
Xlint
warning: [path] bad path element "/usr/lib/hadoop-mapreduce/jaxb-api.jar": no such file or directory
warning: [path] bad path element "/usr/lib/hadoop-mapreduce/activation.jar": no such file or directory
warning: [path] bad path element "/usr/lib/hadoop-mapreduce/jsr173_1.0_api.jar": no such file or directory
warning: [path] bad path element "/usr/lib/hadoop-mapreduce/jaxb1-impl.jar": no such file or directory
4 warnings
[cloudera@cloudera-vm-25 ~]$ jar -cvf martina.jar -C martina/build .
added manifest
adding: Martina$Map.class(in = 1957) (out= 811)(deflated 58%)
adding: Martina.class(in = 2047) (out= 1035)(deflated 49%)
adding: Martina$Reduce.class(in = 1720) (out= 718)(deflated 58%)
[cloudera@cloudera-vm-25 ~]$ hadoop jar martina.jar Martina  movielens/input/ratings.csv movielens/output/martina/ru
n
16/06/11 00:56:51 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/06/11 00:56:52 INFO input.FileInputFormat: Total input paths to process : 1
16/06/11 00:56:52 INFO mapreduce.JobSubmitter: number of splits:5
16/06/11 00:56:53 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1465322315003_0024
16/06/11 00:56:53 INFO impl.YarnClientImpl: Submitted application application_1465322315003_0024
16/06/11 00:56:53 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_14
65322315003_0024/
```

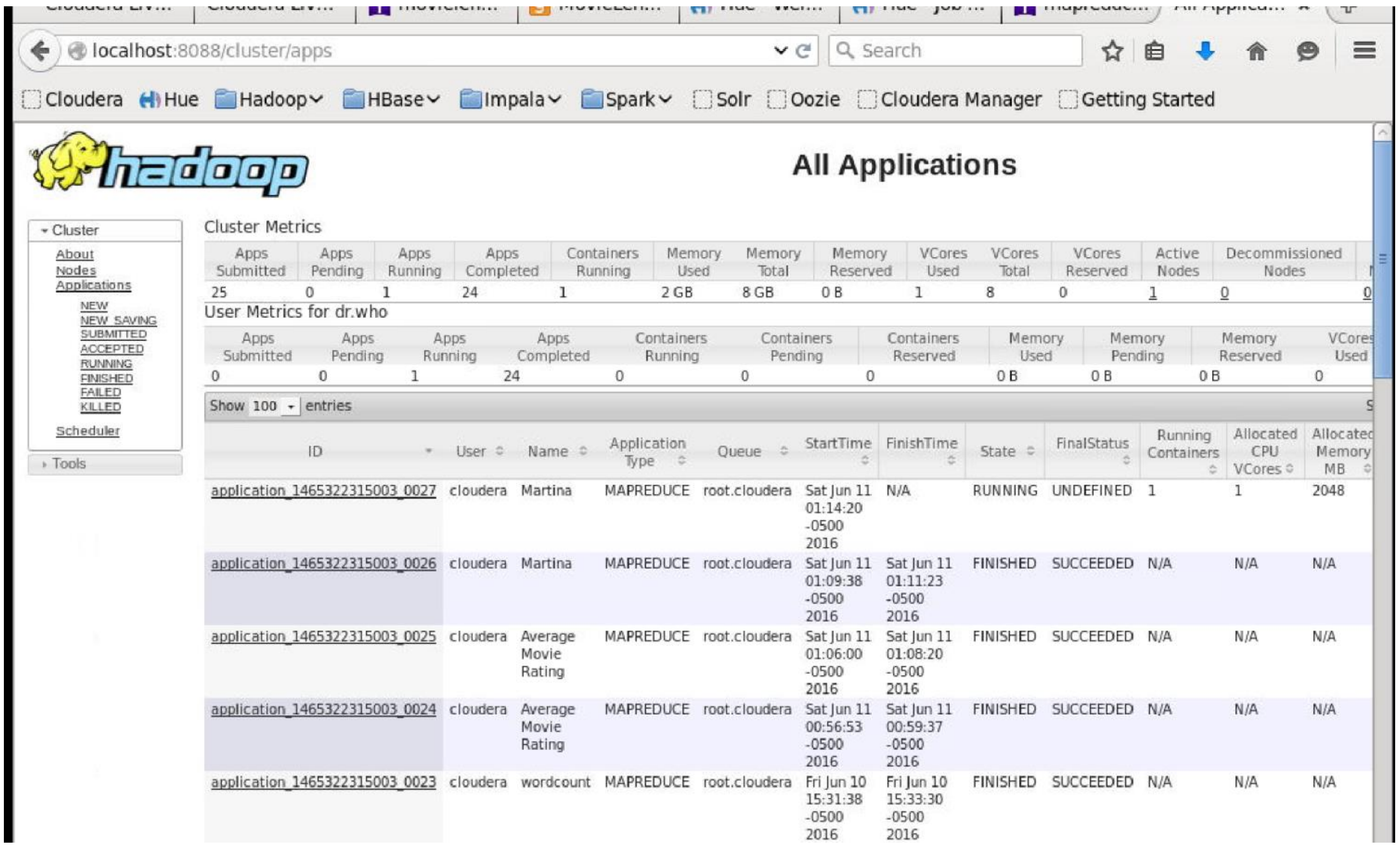
(Rest of screenshots on next pages)

Example: Screenshot of the fourth command (yes, it took me many runs until I realized that I have to remove the combiner):



```
[cloudera@cloudera-vm-25 ~]$ hdfs dfs -cat movielens/output/martina/run34/part-r-00000 | head -10
4       3.7295081967213113
10      3.466666666666667
12      4.078651685393258
14      2.940677966101695
18      3.0434782608695654
20      3.3659574468085105
24      4.022222222222222
26      4.0588235294117645
28      3.9789915966386555
30      3.92
cat: Unable to write to output stream.
[cloudera@cloudera-vm-25 ~]$ hdfs dfs -cat movielens/output/martina/run34/part-r-00000 | tail -10
247732  3.5708661417322833
247734  3.606382978723404
247736  4.648936170212766
247738  3.330188679245283
247740  3.607142857142857
247742  4.1063829787234045
247746  3.8048780487804876
247748  3.5652173913043477
247750  3.8
247752  2.727272727272727
```

Example: Screenshot of my job log:



All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes
25	0	1	24	1	2 GB	8 GB	0 B	1	8	0	1	0

User Metrics for dr.who

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved	VCores Used
0	0	1	24	0	0	0	0 B	0 B	0 B	0

Show 100 entries

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB
application_1465322315003_0027	cloudera	Martina	MAPREDUCE	root.cloudera	Sat Jun 11 01:14:20 -0500 2016	N/A	RUNNING	UNDEFINED	1	1	2048
application_1465322315003_0026	cloudera	Martina	MAPREDUCE	root.cloudera	Sat Jun 11 01:09:38 -0500 2016	Sat Jun 11 01:11:23 -0500 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1465322315003_0025	cloudera	Average Movie Rating	MAPREDUCE	root.cloudera	Sat Jun 11 01:06:00 -0500 2016	Sat Jun 11 01:08:20 -0500 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1465322315003_0024	cloudera	Average Movie Rating	MAPREDUCE	root.cloudera	Sat Jun 11 00:56:53 -0500 2016	Sat Jun 11 00:59:37 -0500 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1465322315003_0023	cloudera	wordcount	MAPREDUCE	root.cloudera	Fri Jun 10 15:31:38 -0500 2016	Fri Jun 10 15:33:30 -0500 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A