MapReduce & YARN
Hands-on Lab Exercise 1
Simple MapReduce program in Java

Contents

Lab 1 MapReduce using Java

For this exercise, you must have installed PuTTY and WinSCP.

In this exercises, the student develops a MapReduce application that finds the highest average monthly temperature using Java.

After completing this hands-on lab, you'll be able to:

- Code a MapReduce application using

Java Allow 45 to 60 minutes to complete this lab.

This version of the Hands -On Lab has been updated to use IBM Analytics Engine on IBM Cloud.

For a tutorial on setting up PuTTY, WinSCP, and IBM Analytic Engine resource .
Please refer to the Lab Setup Instructions on your course page.

The assumption is that you have secured and downloaded the file **BDU_MapReduce_and_YARN.tar** data in that tar file is required by this exercise.

Since you will have access to only a command-line, you can use notepad and transfer the files over with WinSCP or you can use the default editor vi that comes with UNIX (Tutorial here) to edit your text files.

Since Eclipse is not currently installed in this VM Image, we will create our files using an ordinary editor

(vi) and not the Eclipse Java environment. If you are familiar enough with Eclipse, including installation in Linux, you can choose to use that instead.

In addition, please also check the below links for how to use PuTTY and WinSCP :

How to use PuTTY on windows

https://www.ssh.com/ssh/putty/windows/

How to use PuTTY on Linux

https://www.ssh.com/ssh/putty/linux/

How to use PuTTy on Mac

https://www.ssh.com/ssh/putty/mac/

How to use WinSCP

https://winscp.net/eng/docs/guides

1.1 Load and examine the sample data

__1. You should start with a window like this after you log in

```
leowu-macBookPro:~ leowu$ ssh clsadmin@165.192.74.27
The authenticity of host '165.192.74.27 (165.192.74.27)' can't be established.
ECDSA key fingerprint is SHA256:LwCIvuaZQlNbkC9G+Nb573OyDXYm8gFMhImpnd3Suh4.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '165.192.74.27' (ECDSA) to the list of known hosts.

clsadmin@165.192.74.27's password:
Last login: Sun Nov 11 13:05:24 2018
[clsadmin@chs-qyw-262-mn003 ~]$
```

__2. In the PuTTY window, examine the directory structure of the hadoop file system of your directory. Type:

```
hdfs dfs -ls
```

```
[clsadmin@chs-qyw-262-mn003 ~]$ hdfs dfs -ls /
Found 11 items
drwxrwxrwx   - yarn     hadoop          0 2018-11-11 13:05 /app-logs
drwxr-xr-x   - hdfs     bihdfs          0 2018-11-11 13:05 /apps
drwxr-xr-x   - yarn     hadoop          0 2018-11-11 13:05 /ats
drwxr-xr-x   - hdfs     bihdfs          0 2018-11-11 13:05 /hdp
drwx------   - livy     bihdfs          0 2018-11-11 13:06 /livy2-recovery
drwxr-xr-x   - mapred   bihdfs          0 2018-11-11 13:05 /mapred
drwxrwxrwx   - mapred   hadoop          0 2018-11-11 13:05 /mr-history
drwx------   - clsadmin biusers         0 2018-11-11 13:07 /securedir
drwxrwxrwx   - spark    hadoop          0 2018-11-11 13:34 /spark2-history
drwxrwxrwx   - hdfs     bihdfs          0 2018-11-11 13:05 /tmp
drwxr-xr-x   - hdfs     bihdfs          0 2018-11-11 13:06 /user
[clsadmin@chs-qyw-262-mn003 ~]$
```

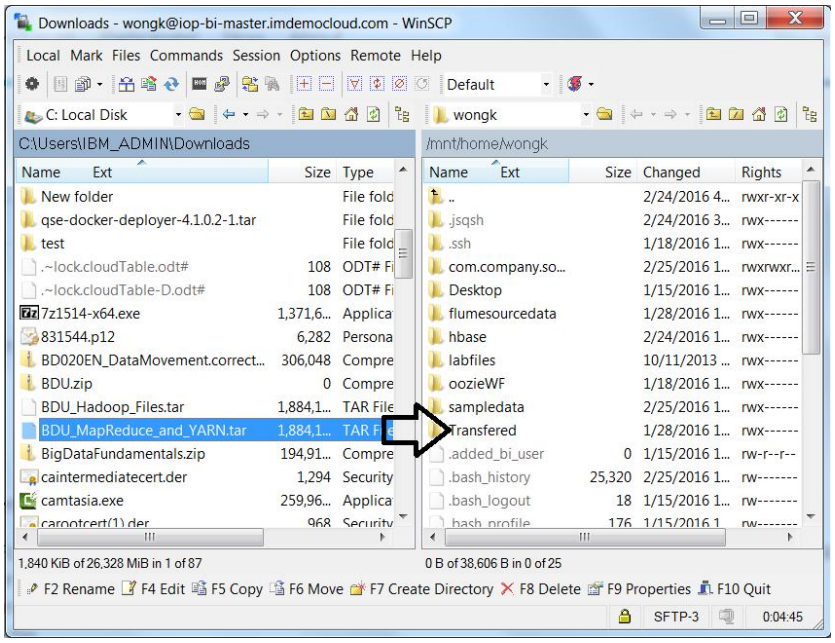
You want to make a new directory to store the sample data files that you will use for this exercise.

__3. Make a new directory called sampledata/, and verify that it was created. Type:

```
hdfs dfs -mkdir sampledata
hdfs dfs -ls
hdfs dfs -chmod 777 sampledata
```

```
[clsadmin@chs-qyw-262-mn003 ~]$ hdfs dfs -mkdir sampledata
[clsadmin@chs-qyw-262-mn003 ~]$ hdfs dfs -ls /
Found 11 items
drwxrwxrwx   - yarn     hadoop          0 2018-11-11 13:05 /app-logs
drwxr-xr-x   - hdfs     bihdfs          0 2018-11-11 13:05 /apps
drwxr-xr-x   - yarn     hadoop          0 2018-11-11 13:05 /ats
drwxr-xr-x   - hdfs     bihdfs          0 2018-11-11 13:05 /hdp
drwx------   - livy     bihdfs          0 2018-11-11 13:06 /livy2-recovery
drwxr-xr-x   - mapred   bihdfs          0 2018-11-11 13:05 /mapred
drwxrwxrwx   - mapred   hadoop          0 2018-11-11 13:05 /mr-history
drwx------   - clsadmin biusers         0 2018-11-11 13:07 /securedir
drwxrwxrwx   - spark    hadoop          0 2018-11-11 13:35 /spark2-history
drwxrwxrwx   - hdfs     bihdfs          0 2018-11-11 13:05 /tmp
drwxr-xr-x   - hdfs     bihdfs          0 2018-11-11 13:06 /user
[clsadmin@chs-qyw-262-mn003 ~]$ hdfs dfs -chmod 777 sampledata
[clsadmin@chs-qyw-262-mn003 ~]$
```

__4. Now, to move **BDU_ MapReduce_and_YARN.tar** file into cloud. Open up WinSCP and transfer the file to your home directory. Just drag the file from the left side to the home directory on the right side.



__5. Then extract the file.

```
cd ~
tar -xvf BDU_MapReduce_and_YARN.tar
```

```
labfiles/MapReduce/LoadMaxTemp/src/com/
labfiles/MapReduce/LoadMaxTemp/src/com/ibm/
labfiles/MapReduce/LoadMaxTemp/src/com/ibm/dw61/
labfiles/MapReduce/LoadMaxTemp/src/com/ibm/dw61/MaxTempReducer.java
labfiles/MapReduce/LoadMaxTemp/src/com/ibm/dw61/MaxTempMapper.java
labfiles/MapReduce/LoadMaxTemp/src/com/ibm/dw61/MaxMonthlyTemp.java
labfiles/MapReduce/LoadMaxTemp/.project
labfiles/MapReduce/LoadMaxTemp/.settings/
labfiles/MapReduce/LoadMaxTemp/.settings/org.eclipse.core.resources.prefs
labfiles/MapReduce/LoadMaxTemp/.biginsights
labfiles/MapReduce/LoadMaxTemp/.textanalytics
labfiles/SampleData/
labfiles/SampleData/reviews.csv
labfiles/SampleData/books.csv
labfiles/SampleData/Twitter Search.json
labfiles/SampleData/bookreviews.json
labfiles/SampleData/pig_bookreviews.json
labfiles/examples/
labfiles/examples/WordCount.java
labfiles/examples/WordCount.jar
labfiles/copyright.txt
labfiles/GutenbergDocs/
labfiles/GutenbergDocs/last_of_the_mohicans.txt
labfiles/GutenbergDocs/walden.txt
```

Now your window should show something similar when it's done extracting the files.

__6. Now that the files are extracted, we will upload the temperature data from the local file system to the HDFS using the following commands:

```
hdfs dfs -put ~/labfiles/SumnerCountyTemp.dat sampledata
```

```
[wongk@iop-bi-master ~]$ hdfs dfs -put ~/labfiles/SumnerCountyTemp.dat sampledat
a
[wongk@iop-bi-master ~]$
```

__7. Test to see that the file was uploaded correctly by typing the following command:

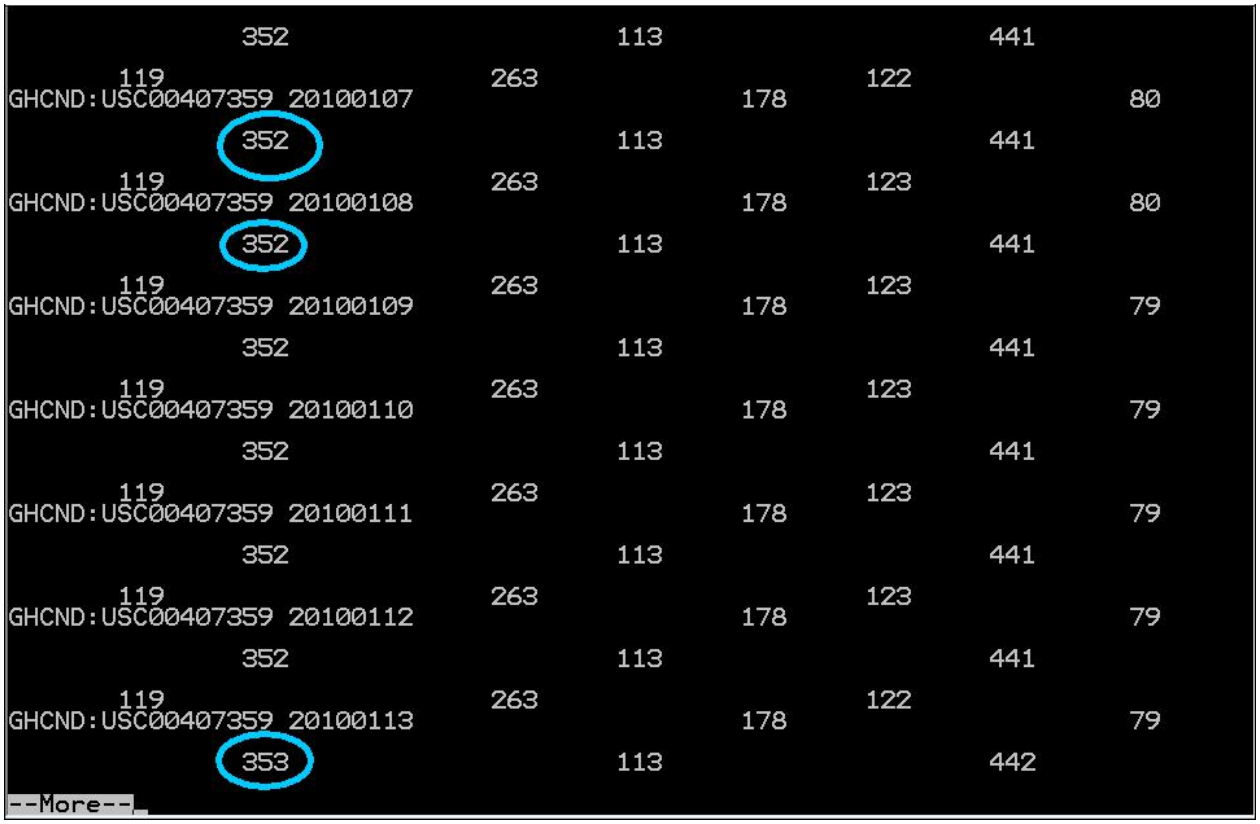
```
hdfs dfs -ls sampledata
```

```
[wongk@iop-bi-master ~]$ hdfs dfs -ls sampledata
-rw-rw----+  3 wongk wongk     240900 2016-02-25 13:25 sampledata
[wongk@iop-bi-master ~]$
```

Notice that your SumnerCountyTemp.dat files was uploaded correctly.

__8. You can view this data by executing the following command:

```
hdfs dfs -cat sampledata/SumnerCountyTemp.dat | more
```



The values in the 95th column (354, 353, 353,353, 352, ...) are the average daily temperatures. They are the result of multiplying the actual average temperature value times 10. (Incidentally, that way you don't have to worry about working with decimal points.)

__9. Press the spacebar a few times to scroll through the data and observe the temperature patterns by date. When you are satisfied, press **Ctrl+c** to break out of the piped output.

1.2 Start your Java project

__1. Create a directory to hold the three Java files that you will be making and make it accessible. The directory will be used to hold program artifacts and to separate it from the other things in the file system.

```
cd ~
mkdir com.company.name
cd com.company.name
```

1.3 Create the Java file for the mapper class

__1. Create a new Java file, **MaxTempMapper.java**:

```
vi MaxTempMapper.java
```

There is a standard set of imports that will also be used for the other two Java files that you create.

The data type for the input key to the mapper will be *LongWritable.* The data itself will be of type *Text.* The output key from the mapper will be of type *Text.* And the data from the mapper (the temperature) will be of type *IntWritable.*

You need a public class with name **MaxTempMapper**. For this class,

___a. You will need to import *java.io.IOException*.

___b. Extend **Mapper<LongWritable, Text, Text, IntWritable>**

___c. Define a public class called map.

___d. Your code should look like the following:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MaxTempMapper extends
     Mapper<LongWritable, Text, Text, IntWritable> {
   @Override
   public void map(LongWritable key, Text value, Context
           context) throws IOException, InterruptedException {
   }
}
```

In the next section you will define the map method.

Note: *You can also create the .java file in notepad and transfer it via WinSCP*

1.4 Complete the mapper

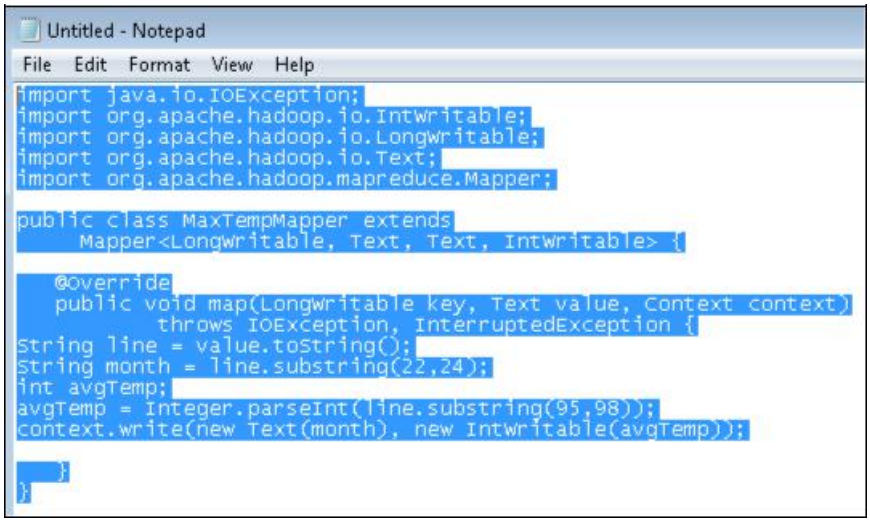
Your program will read in a line of data as a string so that you can do string manipulation. You will want to extract the month and average temperature for each record.

The month begins at the 22th character of the record (zero offset) and the average temperature begins at the 95th character. (Remember that the average temperature value is three digits, with implied one decimal place).

__1. In the *map* method, add the following code (or whatever code you think is required):

```
String line = value.toString();
String month = line.substring(22,24);
int avgTemp;
avgTemp = Integer.parseInt(line.substring(95,98));
context.write(new Text(month), new IntWritable(avgTemp));
```

__2. From this document, you may wish to copy the entire content of the file into Windows Notepad where you can insert the code for the map method. Then transfer the java file (for ex: *MaxTempMapper.java*) to your com.company.name folder.



If you are using vi, press **Esc**, and then type **:wq** to write and exit the vi editor and write your file

or

Press Esc then hit Shift + z twice.

1.5 Create the reducer class

__1. Create a new Java file, **MaxTempMapper.java**:

```
vi MaxTempReducer.java
```

You need a public class with name **MaxTempReducer** and the data type for the input key to the reducer will be *Text.* The data itself will be of type *IntWritable.* The output key from the reducer will be of type *Text.* And the data from the reducer will be of type *IntWritable.*

For your class,

___a. You will need to import **java.io.IOException**

___b. Extend **Reducer<Text, LongWritable, Text, IntWritable>**

___c. Define a public class called ***reduce*.**

___d. Your code should look like the following:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MaxTempReducer extends
     Reducer<Text, IntWritable, Text, IntWritable> {
   @Override
   public void reduce(Text key, Iterable<IntWritable> values, Context
context)
               throws IOException, InterruptedException {
   }
}
```

1.6 Complete the reducer

__1. For the reducer, you want to iterate through all values for a given key. For each value found, check to see if it is higher than any of the other values.

Add the following code (or your variation) to the *reduce* method.

```
int maxTemp = Integer.MIN_VALUE;
for (IntWritable value: values) {
maxTemp = Math.max(maxTemp, value.get());
}
context.write(key, new IntWritable(maxTemp));
```

__2. Assemble your file in the vi editor, notepad or any way you choose, and remember to save your work.

1.7 Create the driver

__1. Create a new Java file, **MaxMonthTemp.java**:

```
vi MaxMonthTemp.java
```

You need a public class with name **MaxMonthTemp** and the standard set of import files.

The *GenericOptionsParser()* will extract any input parameters that are not system parameters and place them in an array. In your case, two parameters will be passed to your application. The first parameter is the input file. The second parameter is the output directory. (This directory must not exist or your MapReduce application will fail.)

Your code should look like this:

```
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.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class MaxMonthTemp {

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

      { Configuration conf = new Configuration();

      String[] programArgs =
          new GenericOptionsParser(conf,
      args).getRemainingArgs(); if (programArgs.length != 2) {
         System.err.println("Usage: MaxTemp <in> <out>");
         System.exit(2);
      }
      Job job = Job.getInstance(conf, "Monthly Max
      Temp"); job.setJarByClass(MaxMonthTemp.class);
      job.setMapperClass(MaxTempMapper.class);
      job.setReducerClass(MaxTempReducer.class);

      job.setOutputKeyClass(Text.class);

      job.setOutputValueClass(IntWritable.class);

      FileInputFormat.addInputPath(job, new Path(programArgs[0]));

      FileOutputFormat.setOutputPath(job, new Path(programArgs[1]));

      // Submit the job and wait for it to finish.

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

   }
}
```

__2. Assemble your file in the vi editor any way you choose, and remember to save your work.

1.8 Compile your Java files & create the JAR file

__1. Compile all three Java files with one statement as the root user and then list your directory to see that you now have three Java source files and three Java class files. Type:

```
cd ~/com.company.name
javac -cp `hadoop classpath`  *.java
ls -l
```

Note that the quotes here are ***back-quotes*** (the key to top left corner of your keyboard, to the left of the one-key [“1”}). The command *hadoop classpath* is executed to list the required classpath information needed by the compiler; the result of this is passed to javac with the classpath option (-cp).

```
[wongk@iop-bi-master ~]$ cd ~/com.company.name
[wongk@iop-bi-master com.company.name]$ javac -cp `hadoop classpath`  *.java
[wongk@iop-bi-master com.company.name]$ ls -l
total 24
-rw------- 1 wongk wongk 1778 Feb 25 13:44 MaxMonthTemp.class
-rw------- 1 wongk wongk 1268 Jan 21 13:33 MaxMonthTemp.java
-rw------- 1 wongk wongk 1608 Feb 25 13:44 MaxTempMapper.class
-rw------- 1 wongk wongk  616 Jan 21 13:33 MaxTempMapper.java
-rw------- 1 wongk wongk 1673 Feb 25 13:44 MaxTempReducer.class
-rw------- 1 wongk wongk  549 Jan 21 13:33 MaxTempReducer.java
[wongk@iop-bi-master com.company.name]$
```

Note: If you get this error:

```
▪▪▪▪▪▪FileInputFormat.addInputPath(job, new Path(programArgs[0]));
   ^
MaxMonthTemp.java:29: error: unmappable character for encoding UTF8
▪▪▪▪▪▪FileInputFormat.addInputPath(job, new Path(programArgs[0]));
    ^
MaxMonthTemp.java:29: error: unmappable character for encoding UTF8
▪▪▪▪▪▪FileInputFormat.addInputPath(job, new Path(programArgs[0]));
     ^
MaxMonthTemp.java:31: error: unmappable character for encoding UTF8
▪▪▪▪▪▪FileOutputFormat.setOutputPath(job, new Path(programArgs[1]));
^
MaxMonthTemp.java:31: error: unmappable character for encoding UTF8
▪▪▪▪▪▪FileOutputFormat.setOutputPath(job, new Path(programArgs[1]));
 ^
MaxMonthTemp.java:31: error: unmappable character for encoding UTF8
▪▪▪▪▪▪FileOutputFormat.setOutputPath(job, new Path(programArgs[1]));
  ^
100 errors
```

This meant that you copied & pasted directly from this document while using a program that converted some of the whitespaces into a different format. To correct this, open notepad from windows and/or vi and delete the characters that give you an error and replace it with a typed character of itself. (Ex, replace the above boxes, which show up as spaces on your editor with spaces typed from your keyboard).

__2. Create a Java Archive File (jar cf, where c = create, f = file) from the three class files. Then list the manifest (jar tf) of the archive file:

```
jar cf MaxMT.jar *.class
ls
jar tf *.jar
```

```
[wongk@iop-bi-master com.company.name]$ jar cf MaxMT.jar *.class
[wongk@iop-bi-master com.company.name]$ ls
MaxMonthTemp.class  MaxTempMapper.class   MaxTempReducer.java
MaxMonthTemp.java   MaxTempMapper.java
MaxMT.jar           MaxTempReducer.class
[wongk@iop-bi-master com.company.name]$ jar tf *.jar
META-INF/
META-INF/MANIFEST.MF
MaxMonthTemp.class
MaxTempMapper.class
MaxTempReducer.class
```

__3. The Java Archive File was created in the directory where the .java and .class files reside. But when we use Hadoop MapReduce to run the jar, Hadoop does not like to have the .class files in the same directory. Therefore you want to move the file to the parent directory, where we will run it in the next step:

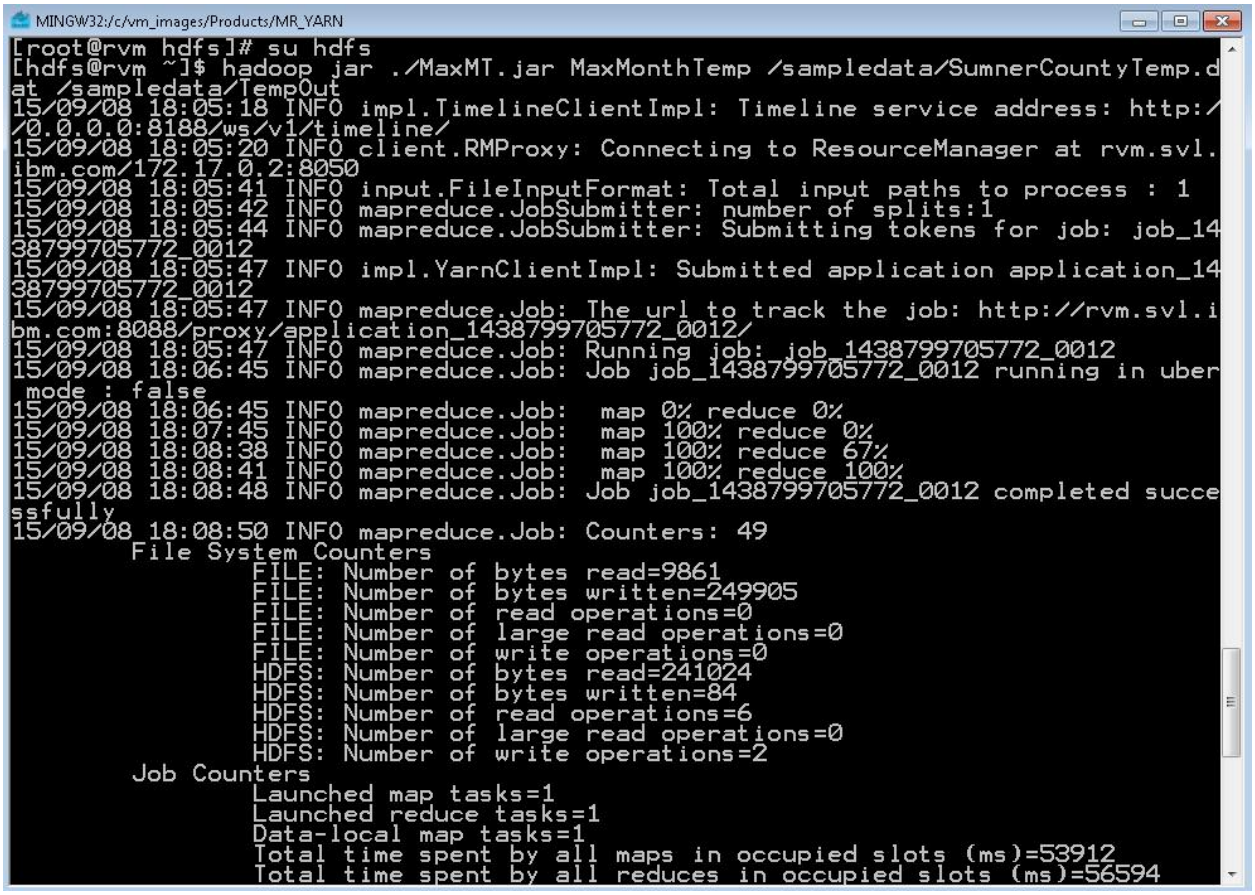
```
cp *.jar ..
cd ..
```

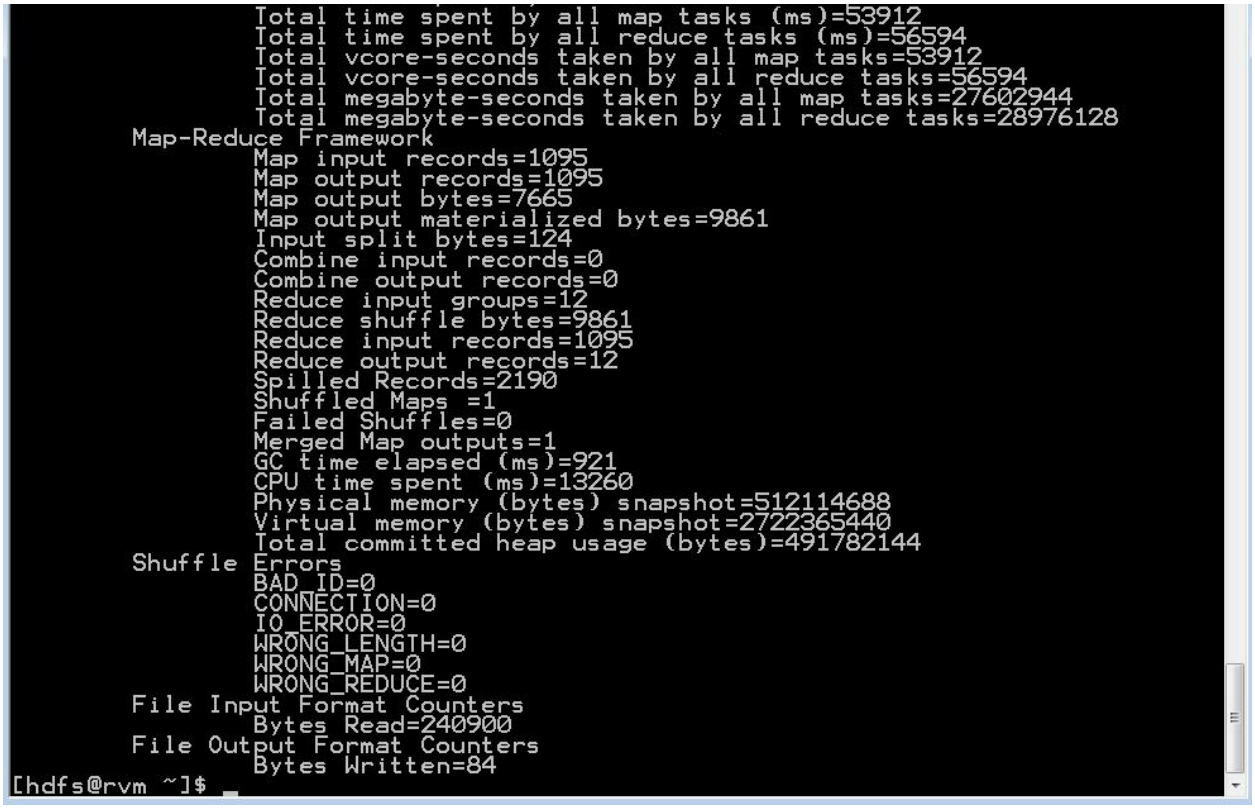
1.9 Run the JAR file

__1. Run the application. Type:

```
hadoop jar ./MaxMT.jar MaxMonthTemp
sampledata/SumnerCountyTemp.dat sampledata/TempOut
```

You will see the following output in that terminal window (but your results will be slightly different, of course):



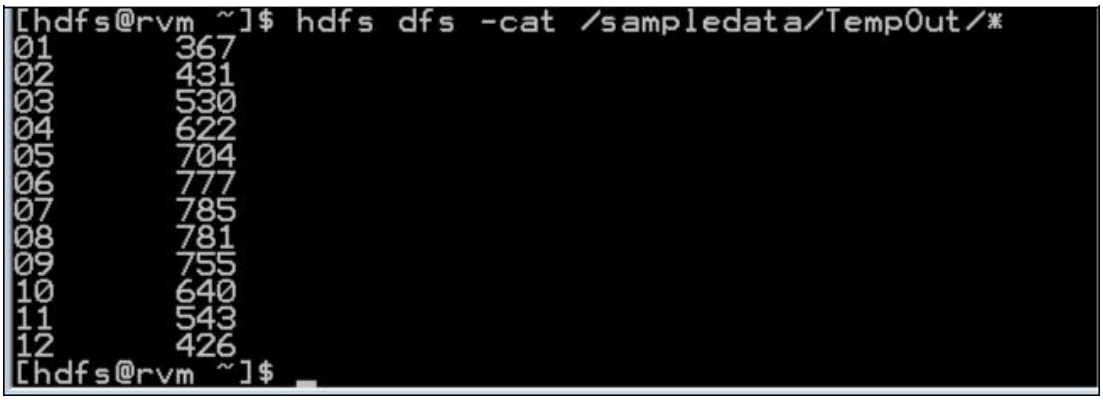


Your results are certainly different, but the final lines of output will probably be similar.

__2. You want to examine the output that was produced:

```
hdfs dfs -cat sampledata/TempOut/*
```

The result of this run should be similar to:



Be aware that if you cut & paste from this file, sometimes Microsoft and Adobe software change a single dash ("-") to an en-dash ("–") or an em-dash ("—"). Linux is not kind to these characters.

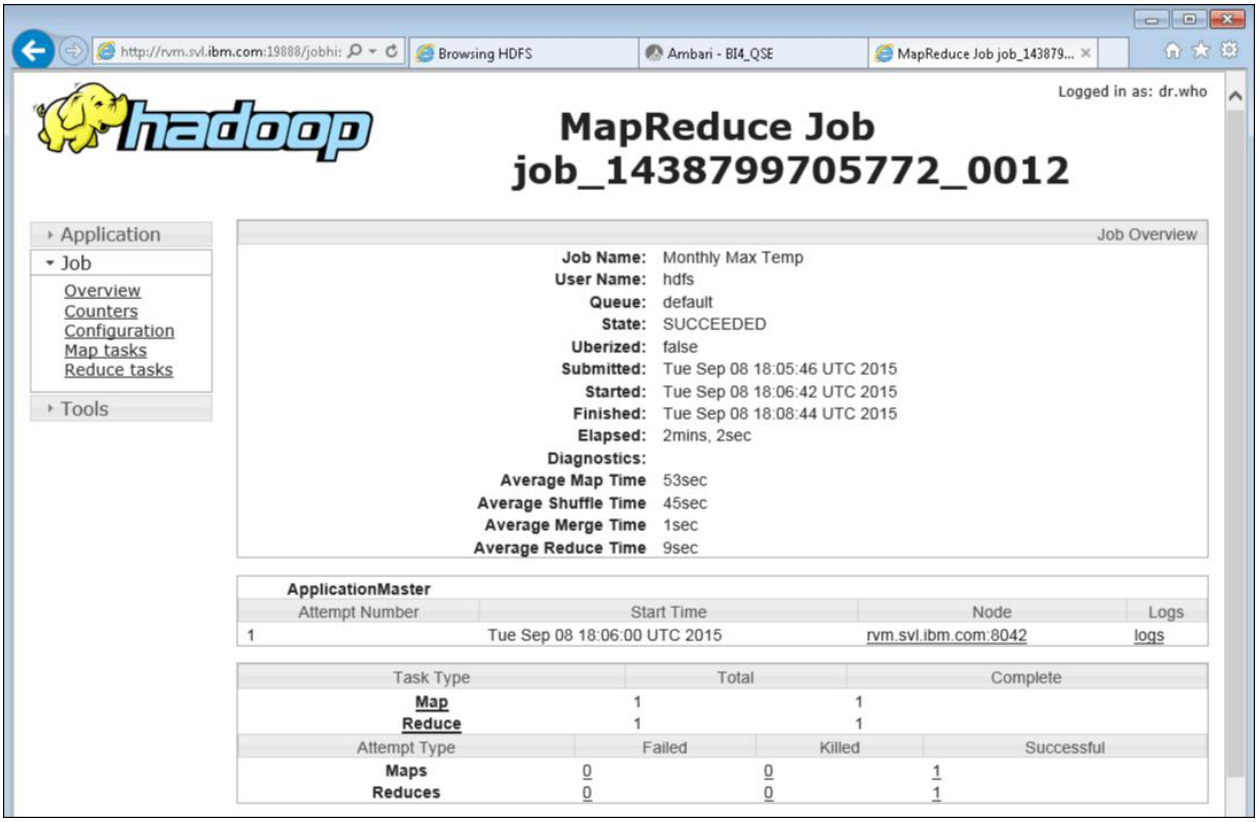
You can see that the maximum temperature for January (01) was 36.7 F — this is the coldest month of Winter in Sumner County as evidenced by the data — and the maximum temperature for July (07) was 78.5 F for the County (a rather cool summer, it appears).

__3. Examine the job status. Scroll up and notice that your output displayed a line of text similar to the following:

```
16/02/25 14:01:36 INFO mapreduce.Job: The url to track the job: http://iop-bi-ma
ster.imdemocloud.com:8088/proxy/application_1453821486875_0655/
```

Open a new tab in your Web browser and browse to the URL listed in YOUR output. Remember that it will differ slightly from the URL used in this example.

Note: *You may be able to access this, but here's what it would look like*



Next, you will make some changes to your program to make it more efficient in a large cluster. You will add a Combiner class and then re-run the application.

1.10 Add a combiner function

__1. Return to your Java development environment:

```
cd ~/com*
```

You will add a combiner function to your application. In a real world multi-node cluster, this would let reducing functions take place on the mapper node and lessen the amount of network traffic.

__2. Use the vi editor to examine the code for MaxMonthTemp.java, and then add the following statement after the *job.setMapperClass(MaxTempMapper.class);* statement.

```
job.setCombinerClass(MaxTempReducer.class);
```

This statement is added to the Driver code. Note that you have not changed the Mapper class, but this code will cause Hadoop to invoke the Reducer code inside the Mapper task before file(s) are passed from Mapper(s) to Reducer(s).

__3. Save your work (write, quit).

__4. Switch back to the root user and recompile this Java file:

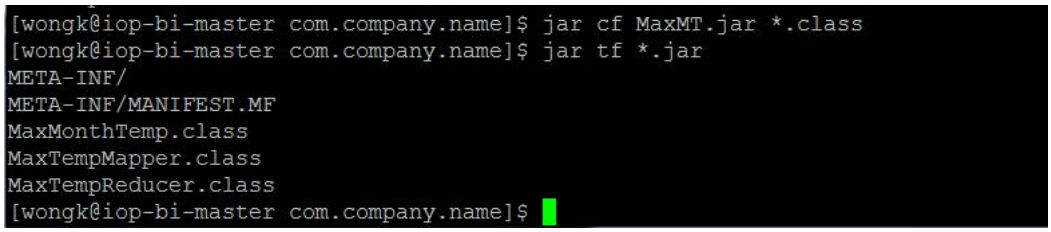
```
javac -cp `hadoop classpath` MaxMonthTemp.java
```

```
[wongk@iop-bi-master com.company.name]$ javac -cp `hadoop classpath` MaxMonthTem
p.java
```

1.11 Recreate the JAR file & re-run your application

__1. Create a Java Archive File (jar cf, where c = create, f = file) from the three class files. Then list the manifest (jar tf) of the archive file:

```
jar cf MaxMT.jar *.class
jar tf *.jar
```



__2. The Java Archive File was created in the directory where the .java and .class files reside. But when we use Hadoop MapReduce to run the jar, Hadoop does not like to have the .class files in the same directory. Thus will move the file to the parent directory, where we will run it in the next step:

```
cp *.jar ..
```
(type **y** at the overwrite confirmation if prompt)

```
cd ..
```

__3. Use the hdfs user to re-run this application now:

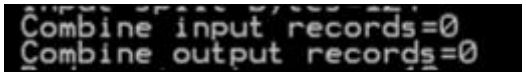
```
hadoop jar ./MaxMT.jar MaxMonthTemp
sampledata/SumnerCountyTemp.dat sampledata/TempOut2
```

Notice that we created a second output directory (TempOut2). MapReduce expects that the output directory does not exist.

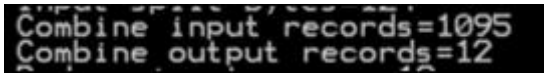

MINGW32:/c/vm_images/Products/MR_YARN
[hdfs@rvm ~]$ hadoop jar ./MaxMT.jar MaxMonthTemp /sampledata/SumnerCountyTemp.dat /sampledata/TempOut2
15/09/08 19:32:16 INFO impl.TimelineClientImpl: Timeline service address: http://0.0.0.0:8188/ws/v1/timeline/
15/09/08 19:32:19 INFO client.RMProxy: Connecting to ResourceManager at rvm.svl.ibm.com/172.17.0.2:8050
15/09/08 19:32:29 INFO input.FileInputFormat: Total input paths to process : 1
15/09/08 19:32:31 INFO mapreduce.JobSubmitter: number of splits:1
15/09/08 19:32:33 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1438799705772_0013
15/09/08 19:32:36 INFO impl.YarnClientImpl: Submitted application application_1438799705772_0013
15/09/08 19:32:37 INFO mapreduce.Job: The url to track the job: http://rvm.svl.ibm.com:8088/proxy/application_1438799705772_0013/
15/09/08 19:32:37 INFO mapreduce.Job: Running job: job_1438799705772_0013
15/09/08 19:33:49 INFO mapreduce.Job: Job job_1438799705772_0013 running in uber mode : false
15/09/08 19:33:49 INFO mapreduce.Job: map 0% reduce 0%
15/09/08 19:34:45 INFO mapreduce.Job: map 100% reduce 0%
15/09/08 19:35:34 INFO mapreduce.Job: map 100% reduce 100%
15/09/08 19:35:39 INFO mapreduce.Job: Job job_1438799705772_0013 completed successfully
15/09/08 19:35:40 INFO mapreduce.Job: Counters: 49
File System Counters
FILE: Number of bytes read=114
FILE: Number of bytes written=230711
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=241024
HDFS: Number of bytes written=84
HDFS: Number of read operations=6
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Launched map tasks=1
Launched reduce tasks=1
Data-local map tasks=1
Total time spent by all maps in occupied slots (ms)=48225
Total time spent by all reduces in occupied slots (ms)=47767
Total time spent by all map tasks (ms)=48225
Total time spent by all reduce tasks (ms)=47767
Total vcore-seconds taken by all map tasks=48225
Total vcore-seconds taken by all reduce tasks=47767
Total megabyte-seconds taken by all map tasks=24691200
Total megabyte-seconds taken by all reduce tasks=24456704
Map-Reduce Framework
Map input records=1095
Map output records=1095
Map output bytes=7665
Map output materialized bytes=114
Input split bytes=124
Combine input records=1095
Combine output records=12
Reduce input groups=12
Reduce shuffle bytes=114
Reduce input records=12
Reduce output records=12
Spilled Records=24
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=529
CPU time spent (ms)=11490
Physical memory (bytes) snapshot=492523520
Virtual memory (bytes) snapshot=2737389568
Total committed heap usage (bytes)=491257856
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=240900
File Output Format Counters
Bytes Written=84
[hdfs@rvm ~]$

__4. Examine the Combine input records and Combine output records of your output and notice the difference in using the combiner function.

The original execution produced this output:



The subsequent execution, using the combiner, produced this output:



__5. Feel free to use the GUI via the Web browser to validate your job.

__6. Feel free to *cat* the TempOut2 directory if you like to view the results.

End of this Hands-On Lab Exercise

NOTES

NOTES

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.