Project

SUBMISSION: Due on December 8, 2016 at 23:59. Submit code (java file(s), output file(s), jar file) and a report (in pdf format) via Bb.

INSTRUCTIONS: You are reminded to complete this project individually and to comply with NDSU's Academic honesty policy.

CRITERIA FOR ASSESSMENT
Your grade will be determined based on the following factors:

1. Completion of project
2. Correctness and quality of your work:
   - does your code work correctly
   - how well is your code designed/written
   - how well is your code documented
3. Your ability to evaluate your project and the presentation quality of your report

DESCRIPTION
You are asked to implement MapReduce-based java code to calculate the average, minimum and maximum of the weighted sum of different classes. The input data file contains several entries with the following format: `value1 \t value2 \t value3 \t value4 \t value5 \t value6`, where `value1` represents the class, and `value2` to `value6` represent the x values of the weighted sum equation; `value2` is x_1, `value3` is x_2, …, `value6` is x_5. The weighted sum equation is calculated as follows:

$$sum = \sum_{i=1}^{5} w_i \times x_i$$

where $w_1 = 0.1$, $w_2 = w_3 = w_5 = 0.2$, and $w_4 = 0.3$.
The input file looks as follows:

```
6       11.86     39.74     67.98     17.36     76.94
9       5.47      31.47     94.72     64.97     26.74
5       87.46     36.41     76.48     28.14     63.71
...
```

The output should list the average, minimum, and maximum per class of weighted sums. For example, the output should look like this:

```
Class 1: Average = 68.45
Class 1: Min = 17.65
Class 1: Max = 97.43
Class 2: Average = 57.64
Class 2: Min = 21.49
...
```

The output should be stored in output file(s). Your program should work for any dataset of similar structure.
Once your program is up and running, you are asked to conduct a scalability analysis by using different numbers of nodes to process your code on (this determines how many mappers/reducers are used). You should use 2, 4, 6, 8, 10, 12, 14 nodes for each run to split your dataset to be sent to the mappers. You should record the running time using different numbers of nodes (place a time stamp at the beginning and another one at the end of your main method and take the difference to report the running time – you can use `System.currentTimeMillis()`). In order to perform the splitting of the dataset to be sent to the different mappers you have to specify the following:

For the mapper setup:

```
long dataLength = fs.getContentSummary(new Path(dataFile)).getLength();
FileInputFormat.setMaxInputSplitSize(job, (long) (dataLength / numNodes));
```

where `dataFile` is your input file, and `numNodes` is the number of mappers to be used.
For the reducer setup:

```
job.setNumReduceTasks(numNodes/2);
```

Since node failure can happen (which Hadoop handles), you should perform 10 runs and take the average to report the running times for the different numbers of nodes.

REPORT

Your report should:

1. Explain how you have designed and implemented your code
2. Present the evaluation (for the scalability analysis it is best to use a graph)
3. Explain your findings as well as what you have learnt from doing this project

GETTING STARTED

The best is to run the word count example to get a feel of how MapReduce works. In order to do this, follow the steps in this wiki page (pdf version is copied to project folder on Bb as well): http://wiki.cs.ndsu.nodak.edu/deptlab/hadoop/gettingstarted
For writing your own code, you can use an IDE such as Eclipse or compile directly on the Hadoop cluster (zoidberg). If you use an IDE then you have to add the necessary Hadoop libraries files as external jar files to your Java build path. I have provided a zip folder with all the necessary jar files on Bb as `Hadoop.zip`. For testing purposes on your local machine, I have provided a small input file (e.g., `DataSetTest.txt` available on Bb).
Once all is running, you can then create a jar file of your code and copy the jar file to your home directory on zoidberg. The datafile `DataSet.txt` to be analyzed is available in the `/home/siludwig/projectfiles` directory on zoidberg. However, you need to copy the file to HDFS before you can run your program. To do this you can use the following command: `hadoop fs -put /home/siludwig/projectfiles/DataSet.txt /home/<user>/.`
Also note that for the input and output path on zoidberg you should use e.g.: `/user/<username>/input/inputFile.txt` and `/user/<username>/output/` where `inputFile.txt` is the file to be analyzed.

RESOURCES

https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
Useful commands:
Log in to Hadoop cluster:

```
ssh <username>@zoidberg.cs.ndsu.nodak.edu
```

Query HDFS:

```
hadoop fs -ls /user/<username>
```

Copy file from local folder to HDFS in:

```
hadoop fs -put test4.txt /user/<username>/input
```

Copy file from HDFS to local folder:

```
hadoop fs -get /user/<username>/output/part-r-00000 /user/<username>/output
```