COMP 543 Assignment #3

Due March 7th at 11:55PM.

1 Description

The goal of this assignment is to implement two MapReduce programs in Java (using Apache Hadoop). Specifically, your MapReduce jobs will analyzing a data set consisting of medication prescriptions in the United Kingdom from July 2016 to September 2017.

2 Rx Dataset

The data set itself is a set of simple text files. Each prescription/prescribing practice is a different line in a file. The attributes present on each line of the files are, in order:

Attribute	Description
SHA	Area team identifier
PCT	Clinical commissioning group identifier
PRACTICE	Practice identifier
BNF_CODE	British National Formulary (BNF) code
BNF_NAME	BNF name
ITEMS	Number of prescription items dispensed
NIC	Net ingredient cost (pounds and pence)
ACT_COST	Actual cost (pounds and pence)
QUANTITY	Quantity - whole numbers
PERIOD	YYYYMM

The data files are in comma separated values (CSV) format. I've uploaded the data to Rice's Box cloud storage. It is stored as fifteen different files located at the location:

```
s3://chrisjermainebucket/comp543_A3/
```

This is about 21 GB of data in all. Important note: be aware that the URLs may not copy-and-paste from this PDF correctly, as you may lose underscore characters. This problem may happen with the other commands below.

Using Amazon EMR, you can use those files as an input to a Hadoop MapReduce job in exactly the same way that you use files stored in HDFS as input to a Hadoop MapReduce job. For example, if you wanted to run a word count, you could just use the command:

```
[hadoop-10-182-96-202]$ hadoop jar myWordCount.jar -r 8 s3://chrisjermainebucket/comp543_A3 outputfile
```

You can list the files from your master node using:

```
[hadoop-10-182-96-202]$ aws s3 ls s3://chrisjermainebucket/comp543_A3/
```

For testing and development, you can run your Hadoop MapReduce jobs on just one of the fifteen files. For example, from your master node:

```
[hadoop-10-182-96-202]$ hadoop jar myWordCount.jar -r 8 s3://chrisjermainebucket/comp543_A3/T201706PDPI+BNFT.c
outputfile
```

A super-small subset of the first file (only about 1000 lines) is available for download (see Canvas). If you want, you can use this file for testing and debugging by loading it into HDFS (just like you did in lab) and then running your MapReduce program over it.

3 The Tasks

There are two separate programming tasks associated with this assignment; both involve analyzing this data.

3.1 Task 1

Write a MapReduce program that checks all of the files and computes the total "net ingredient cost" of prescription items dispensed for each PERIOD in the data set (total pounds and pence from the NIC field).

As you do this, be aware that this data (like all real data) can be quite noisy and dirty. The first line in the file might describe the schema, and so it doesn't have any valid data, just a bunch of text. You may or may not find lines that do not have enough entries on them, or where an entry is of the wrong type (for example, the NIC or ACT_COST cannot be converted into a Java `double`). Basically, you need to write robust code. If you find any error on a line, simply discard the line. Your code should still output the correct result.

3.2 Task 2

Write a MapReduce program that computes the 5 practices that issued the prescriptions with the highest total net ingredient cost in the data set.

Computing this is going to require you to write a sequence of at least a couple of MapReduce jobs. One job is going to need to compute the total net ingredient cost of the prescription prescribed by each practice, so the result will be a file containing a number of `(practice, total_net_ingredient_cost)` pairs. The second job is going to need to compute the five practices that prescribed prescriptions with the highest total net ingredient cost using the first data set. There are a lot of ways to do this, but the most efficient (and a rather simple way) is to have a priority queue as a private member within your `Mapper` class. Every call to `map` just inserts the next practice into the priority queue, organized by total net ingredient cost. Whenever the queue has more than five entries in it, throw out the worst entry so that you only have five. But your `map` method *will not actually emit any data*. Then, your `Mapper` class will implement the `cleanup` method:

```
public void cleanup(Context context) throws IOException, InterruptedException ...
```

This is a method that any `Mapper` class is free to implement, that is automatically called when the mapper is about to be finished. Here, you'll simply emit the contents of the priority queue, writing them to the `Context`. In this way, your `Mapper` class will only write out the five most practices with the highest net ingredient cost prescribing practices processed by that mapper. Essentially, we are using the mapper as a filter. The reducer then computes the top five out of the top five found by every mapper. To do this, the key emitted by your `Mapper` class should just be the same value, like `1`. Then all of the practices' output by all mappers will go to that one reducer. That reducer can collect all of the results and emit the five best at the end.

Note that if you do this, you need to be careful and you cannot insert the Hadoop mutable data types into your queue, since they will be reused. Insert the standard Java mutable types instead. In the end, we are interested in computing a set of `(practice, total_net_ingredient_cost)` pairs.

Another common problem on this task is blindly using your reducer class as a combiner (the word count implementation that was provided uses the `WordCountReducer` class as a combiner, which means that Hadoop may use the class during the shuffle to eagerly attempt to reduce the amount of data). Make sure that you understand the implication of doing this, or else just remove the line that provides a combiner class.

4 Important Considerations

4.1 Machines to Use

One thing to be aware of is that you can choose virtually any configuration for your EMR cluster—you can choose different numbers of machines, and different configurations of those machines. And each is going to cost you differently! Pricing information is available at:

```
http://aws.amazon.com/elasticmapreduce/pricing/
```

Since this is real money, it makes sense to develop your code and run your jobs on a small fraction of the data. Once things are working, you'll then use the entire data set. We are going to ask you to run your Hadoop jobs over the "real" data using two c3.2xlarge machines as workers.

This provides 8 cores per machine (16 cores total) so it is quite a bit of horsepower. You'll want to run 16 reducers, since you have 16 cores.

Be very careful, and shut down your cluster as soon as you are done working. You can always create a new one easily when you begin your work again.

4.2 Monitoring Your Cluster

When you are running your jobs, you'll frequently have questions: what is my job doing? How many mappers are running? How many reducers? Fortunately, Hadoop has a very nice web interface that will allow you to monitor your job. Basically, Hadoop uses your master node as a web server and allows you to connect to display the status of the cluster.

Unfortunately, for security reasons, it's not as easy as simply going to a web address when you want to monitor your EMR Hadoop cluster. Amazon does not allow you to connect directly. Instead, you need to use an SSH tunnel. Fortunately, this should just take a few minutes to set up. See the instructions at:

```
http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-connect-ui-console.html
```

Once you set up the tunnel, you can view the status of your cluster by going to the web page:

```
http://ec2-107-21-183-54.compute-1.amazonaws.com:8088/
```

(you will replace the `ec2-107-21-183-54` with the address of your own master node).

5 Turnin

Create a single document that has results for both tasks. For each task, copy and paste the result that your last MapReduce job wrote out, and include the textual output that you got from Hadoop when you ran the job. Please zip up all of your code and your document (use .gz or .zip only, please!), or else attach each piece of code as well as your document to your submission individually.

6 Grading

Each problem is worth one half of the overall grade. If you get the right answer and your code is correct, **and you make use of the 16 cores available in your cluster** to get parallelism using the MapReduce framework, you get all of the points. If you don't get the right answer or your code is not correct or you don't make use of the cores, you won't get all of the points; partial credit may be given at the discretion of the grader.