BIG DATA – AX II

Andrei-Mihai Nicolae
2147392n

INSTRUCTIONS

In order to run the code, please follow the following instructions (also outlined in the instructions text file):

```
$ cd task1
$ export HADOOP_CLASSPATH=ax2_task1.jar
$ javac-run.sh $(find . -name '*.java')
$ jar -cvf ax2_task1.jar $(find . -name '*.class')
$ java-run.sh MainTaskOne {input_file} {output_file} {start_date} {end_date}
```

```
$ cd task2
$ export HADOOP_CLASSPATH=ax2_task2.jar
$ javac-run.sh $(find . -name '*.java')
$ jar -cvf ax2_task2.jar $(find . -name '*.class')
$ java-run.sh MainTaskTwo {input_file} {output_file} {timestamp}
```

DESIGN

As the assignment was split into 2 tasks, I divided the work into 2 different packages. Both of them have a Main class that sets everything up, among which:

- they both take the input and output files from the command line as first arguments
- they set up the reducer classes, the job name, output format etc.
- they initialize the table mapper job along with the mapper used, the class of the key and value as well as the job

Afterwards, in Task 1 we have a custom mapper and a custom reducer. The reducer is rather simply implemented, adding all revisions into an ArrayList from which, after it's sorted, we create a string containing all revision IDs in a sorted fashion. In the end, we write the key, the value's size and the actual value pair where the key is the articleID, the size is the number of different revisions we've found and the value is the list above-mentioned.

The Task 1 mapper first gets the 2 timestamps from the command line as the last 2 arguments. Then, it checks if the current timestamp is between the 2 provided by the user and, if so, we write the key value pair.

In Task 2, the reducer has been developed such that it performs as optimally as possible. As it loops through the revisions we have, eventually it will select the one that is the closest to the timestamp provided as an argument. In the end, it is formatted from milliseconds to the ISO8601 format requested by the spec sheet and the key value pair is written. I've decided to use a helper class (i.e. UtilityPairRevisionTimestamp) that is in fact a pair formed of a timestamp and a revisionID. This made it easier to set the output class and manipulate the data wherever needed.

Task 2 mapper gets the timestamp from the command line arguments and checks if the timestamp residing in value is before the one passed by the user. If so, we create a new utility pair object and write the key, composite value pair.

HARDSHIPS & LESSONS LEARNED

The biggest hardship in my opinion was the implementation of the utility class for task two. I was not sure how to proceed with getting an easy to use data structure that can store the value, but in the end I managed to come up with the current solution. Another hardship was, without doubt, the huge latency from Hadoop – if there were even a couple of students running jobs, the output would be produced extremely slow. I had to wait until times when the load was lighter as well so that I could test my programs in all possible conditions. Another slightly hard task was to be sure that the revision IDs are sorted, but in the end this was also solved by implementing the necessary

As in task one, I learned that the network latency was the biggest problem in evaluating efficiency. Moreover, I learned that even having arguments that would produce larger outputs (by that I mean, for example, in task 2 having the timestamp as close to the current date as possible, so that there would be a lot more revisions to be checked) was not a factor that influenced the overall performance as much as the network traffic.

In the end, I believe the course was one of the best I've taken so far as it has:

- taught me invaluable knowledge and experience with big data
- made me get my "hands dirty" with a real-world Hadoop cluster
- how to implement my own reducers, mappers and combiners
- showed me how to work with a NoSQL database on top of Hadoop
- how to evaluate my overall performance and how to explain each aspect in detail

RESULTS

Task I

Query Processing Time	Bytes Read from HDFS	Bytes Transferred over Network
Mean=185.6s SD=55.9s	1052	79677768

Task II

Query Processing Time	Bytes Read from HDFS	Bytes Transferred over Network
Mean=248.7s SD=62.3s	1052	30782900

Both tasks were run 5 times and, as mentioned before, the network influenced the performance tremendously, hence the high standard deviation.

The commands I used to run my programs are:

- Task I: $ java-run.sh MainTaskOne BD4:enwiki-perftest hdfs:///user/2147392n/ax2_task1 2005-12-06T17:44:47Z 2008-01-05T14:48:05Z
- Task II: $ java-run.sh MainTaskTwo BD4:enwiki-perftest hdfs:///user/2147392n/ax2_task2 2005-12-06T17:44:47Z