Assignment 2: Big Data Analytics

Due date: Thursday November 4th at 11:00pm Waterloo time

ECE 454/751: Distributed Computing

Instructor: Dr. Wojciech Golab
wgolab@uwaterloo.ca

A few house rules

Collaboration:

- groups of size 1, 2, or 3

Managing source code:

- do keep a backup copy of your code outside of ecelinux
- do not post your code in a public repository (e.g., GitHub free tier)

Software environment:

- test on ecehadoop
- use the provided software versions
  (Java 1.8, Hadoop 3.2.1, Scala 2.12, Spark 3.0.1)

Overview

- In this assignment you will perform analytical computations over a data set of movie ratings.
- The assignment has several parts. All parts use the same input data format. Each part defines a different analytical task.
- For each task, you will implement a Hadoop-based solution using Java and a Spark-based solution using Scala.
- Sample inputs are provided in HDFS under /a2_inputs.
- This assignment is worth 18% of your final course grade.

Data set

The input data set is a matrix of movie ratings. Each row represents one movie, and each column represents one user. Movie titles are provided in the leftmost column, and the remaining columns represent numerical ratings. Columns are delimited by commas, and you may assume that any commas in the movie titles have been stripped. Ratings are integers ranging from 1 to 5 inclusive. Blanks denote missing data.

Example data set with three movies and six users:

The Lord of The Rings,5,4,4,,3,2

Apocalypto,3,5,4,,5,4

Apollo 13,,,4,5,,5

Task 1: users with highest ratings

For each movie, output the column numbers of users who gave the highest rating. If the highest rating is given by multiple users, output all of them in ascending numerical order. Assume that there is at least one non-blank rating for each movie.

Example output for the example input shown earlier:

The Lord of The Rings,1

Apocalypto,2,5

Apollo 13,4,6

Task 2: total number of ratings

Compute the total number of (non-blank) ratings. Output the total as a single number on a single line.

Example output for the example input shown earlier:

13

Note: For this task, the program should generate output in a single file. (Hint: force the framework to use a single reducer.)

Task 3: ratings per user

For each user, output the user's column number and the total number of ratings for that user. If a user has no ratings then output 0 for that user.

Example output for the example input shown earlier:

1,2

2,2

3,3

4,1

5,2

6,3

Task 4: similarity between movies

For each pair of movies, compute the number of users who assigned the same (non-blank) rating to both movies.

Example similarity computation for Apocalypto and Apollo 13:

Ratings for Apocalypto: 3,5,4,,5,4

Ratings for Apollo 13: ,,4,5,,5

Similarity = 1 (because user #3 rated both movies as 4)

Task 4: similarity between movies

Example output for the example input shown earlier:

Apocalypto,Apollo 13,1

Apollo 13,The Lord of The Rings,1

Apocalypto,The Lord of The Rings,1

Note: Output the similarity score exactly once for a given pair of movies, and output the two movie titles of each movie pair in ascending lexicographic order (e.g., "Apocalypto" before "Apollo 13"). Produce output even if the similarity is zero.

Input and output

- Your programs will receive two command line arguments: an input file and an output path. The input file will be specified by the grader. The output path is a directory where your programs must place one or more output files.
- A Hadoop program generates output by emitting tuples in the reducer. If the last job in your workflow does not have a reducer, then the program generates output by emitting tuples in the mapper. A Spark program generates human-readable output by executing the saveAsTextFile method on an RDD. You should not be opening and writing output files using any other means.

Input and output (cont.)

- The input and output files must use comma-delimited records, meaning that consecutive elements of one record are separated by a comma and nothing else. (A record is a line of text.) Do not use tabs or spaces.
- The order of records in the input and output files does not matter. To compare your outputs with the sample outputs given in the starter code tarball, first coalesce and sort the output files, and then compare using the diff utility. Example:

```
cat outputA/* | sort > normalized_outputA.txt
cat outputB/* | sort > normalized_outputB.txt
diff normalized_outputA.txt normalized_outputB.txt
```

Packaging and submission

- The starter code includes sample implementations of word counting for Hadoop and Spark. Shell scripts are provided to build and run this code.
- For task number X, your Hadoop-based solution should be in a single Java file called TaskX.java, and your Spark-based solution should be in a single Scala file called TaskX.scala. All classes must be in the default package.
- For task number X, the mainline of the Hadoop solution should be in a Java class called TaskX, and the mainline of the Spark solution should be in a Scala object called TaskX.
- Each program will take as arguments the input file and the output path only. If you require temporary output directories for your Hadoop workflows, take the output path and append a suffix (e.g., given "myout", append "_tmp" to obtain "myout_tmp").

Packaging and submission (cont.)

- For submission, run the provided package.sh script in Linux to generate a .tar.gz file, and submit this file to the dropbox in LEARN before the deadline.

Testing and assessment criteria

- The grader will run your code on our own data sets, and compare your outputs against a reference implementation using the technique discussed earlier.
- For performance, focus on minimizing the number of job stages, the number of data shuffles, and the amount of time spent in garbage collection. Do not fiddle with tuning knobs.
- For Task 4, you should follow an approach similar to a map-side join. Use the Hadoop distributed cache, and Spark broadcast variables.
- Performance will be assessed partly by measuring the running times of your programs, and partly by inspecting various performance counters (number of bytes read, written, shuffled, etc.).

Grading scheme

Evaluation structure:

Correctness of outputs:	50%
Performance:	50%

Penalties of up to 100% will apply in the following cases:

- solution cannot be compiled or throws an exception during testing despite receiving valid input
- solution produces incorrect outputs, for example due to a logic error or wrong output format
- solution is improperly packaged
- you submitted the starter code instead of your solution
- one or more members of the group did not follow the house rules (see next slide)

Cluster house rules

- Details regarding how to access the Hadoop/Spark cluster will be provided separately.
- Storage space on the cluster is limited, so please keep your HDFS usage below 1GB.
- If you want to experiment with large inputs then consider placing them in HDFS under /tmp where they can be accessed by other students. This encourages sharing, and avoids duplication.
- Please limit yourself to running one Hadoop or Spark job at a time, especially during busy periods such as shortly before the deadline.
- Do not let any job run for more than 30 minutes on the cluster. After 60 minutes of running time, the teaching team reserves the right to kill your job. Repeat offenders will be penalized ☹