Lab 8: Intermediate Hadoop Programs

Due date: March 3, 11:59pm.

Lab Assignment

Assignment Preparation

This is a pair programming lab. You can pair with anyone in the class.

Overview

For this lab, you will implement a number of MapReduce jobs that run on the ThghtShre datasets that you generated.

1 ThghtShre Tasks

You will write two MapReduce tasks for the ThghtShre dataset.

Input. The input data for each of the three programs shall be a single JSON file containing the collection of JSON objects in the ThghtShre format. We will not worry about what your programs do if the input is different, although I recommend some form of graceful detection and exit. One thing to note is that the input file is a list of JSON objects, NOT a JSON array[1].

Note: Good input files for the programs below have relatively few users and multiple messages per single user. You can tweak your Lab 1 generators to produce such outputs, if needed.

[1] You can experiment. If you can make your MapReduce programs accept a JSON array and extract JSON objects from it one by one, feel free to use JSON Arrays as input. Otherwise, make sure your Lab 1 generator produces JSON objects without the JSON array syntax. The JSON objects can (and should be) multi-line.

All of your programs shall accept two input parameters: the location of the input JSON file, and the location of the output directory.

Program 1: Accounting

ThghtShre has decided to charge the users of the system for the communications (heh!). The charge model is as follows:

1. Each message incurs an origination charge of 5 cents.
2. Every 10 bytes (or any share of 10 bytes) of each message cost 1 cent.
3. If a message is longer than 100 bytes (characters), a surcharge of 5 cents is also assessed.
4. Users who write more than 100 messages get an overall 5% discount.

For example, the following message

```
Today is a wonderful day!
```

is 25 characters long. The cost of the message is the 5 cents origination charge plus 3 cents per byte charge, for a total of 8 cents.

Write a program `accounting.java` that computes for each user how much they owe the service.

Output. The output of this program is a collection of key-value pairs where the key is the user Id, and the value is the amount of money the user owes for their messages (in dollars and cents).

Program 2: Hashtagging.

`ThghtShre` decided to associate hashtags with each user account. The hashtags are the most popular words the users use in their messages, except for the words from the stopword list specified below. (If there is a single word most commonly used by a user, only one hashtag is assoicated with him/her. If there are two or more words used with exactly the same frequency, all these words form individual hashtags associated with the user).

The stopwords (i.e., words that do not count as potential hashtags) are:

```
a
the
in
on
I
he
```

```
she
it
there
is
```

Write a program `hashtags.java` that produces the hashtag assignment for each user of the service.

Output. The output of the program is a collection of key-value pairs, where the key is the unique Id of a user and a value is a comma-separated list of hashtags.

For example, if a user `u03243` has hashtags `"fast"` and `"fortune"` associated with them, then the appropriate output line shall look as follows:

```
u03243    fast, fortune
```

Hint. This *may* require multiple MapReduce jobs chained together.

MovieRatings Tasks

You will write two MapReduce tasks for the MovieRatings dataset.

Input. The input data for each of the three programs shall be a single JSON file containing the collection of JSON objects in the MovieRatings format. We will not worry about what your programs do if the input is different, although I recommend some form of graceful detection and exit. One thing to note is that the input file is a list of JSON objects, NOT a JSON array[2].

Additionally, a second file, storing the concordance between the movie names and the rating column Ids, can be used as the second input to the programs where required. Create the file manually. It should be named `movies.csv` and have the following format:

```
1,Star Wars: A New Hope
2,Godfather
...
13,Beverly Hills Cop
```

[2]You can experiment. If you can make your MapReduce programs accept a JSON array and extract JSON objects from it one by one, feel free to use JSON Arrays as input. Otherwise, make sure your Lab 1 generator produces JSON objects without the JSON array syntax. The JSON objects can (and should be) multi-line.

Problem 3: Favorite Movie by State

Find the favorite movie for each state. Report the output in the form

```
<State>     <MovieTitle>
```

For example, if `"Godfather"` winds up being the favorite movie of California, the output line for the state of California should look

```
...
CA    Godfather
...
```

Name the program `favoriteMovie.java`. The program shall take three input parameters: (1) location of input JSON file, (2) location of `movies.csv` file, (3) target output directory.

Note: This program may require both a join, and multiple chained MapReduce jobs.

Program 4: User Similarity

For this program, use **only** input files with **exactly 10 JSON objects in them**. You are also allowed to hardcode the number 10 (size of the input JSON list) into your program. You are also allowed to use files where `RID` values only range from 1 to 10.

Write a program, that takes as input a JSON file of movie ratings as described above, and outputs the *Pearson Correlation* similarity score between each pair of users. Only one output per pair of users shall be reported (i.e., if you are reporting the Pearson Correlation Coefficient for `Person1, Person2`, DO NOT output the line for `Person2, Person1`.

The output shall be in the format

```
Person1, Person2   Similarity
```

For example, if your input file has movie ratings for `Bob Smith` and `Alice Friendly` and their similarity score is 0.45, report this line of output as

```
...
Bob Smith, Alice Friendly   0.45
...
```

Submission

READ CAREFULLY!

Submit your Java programs. For each java program submit one JSON file on which you tested it, and submit the results of running your program on

that file. Name your input files `programName-input.json`, and your output files `programName-output.json`[3]. If you are using the same input file for multiple programs (e.g., for ThghtShre assignments) submit two copies under different names.

Submit a README file with the names of the team members. Also, to make my life easier, submit either a `Makefile` that compiles and runs (`make all`) all four of your programs with some default locations of the input data (put all input files in `test/` directory), or a `runme.sh` bash script that does the same thing.

If you need to sideload any jars except for the ones covered in class, please submit them as well and make sure to include them into the `$HADOOP_CLASSPATH`.

All submitted programs must contain your names in them.

Submit all your code in a single archive (zip or tar.gz).

Use `handin` to submit as follows:

```
$ handin dekhtyar lab08 <FILES>
```

Good Luck!

[3] If you obtain multiple output files for a specific job, add `-part1`, `part2` etc. to the output file names.