Introduction to Hadoop

The goal of this lab is to make you write your first MapReduce jobs. Since the code you develop for this lab is graded, please keep in mind that you should hand over a copy of the code that can be read, understood and rerun. This lab is particularly long, you are not expected to finish to get a good grade.

1 Introduction

The Movie Lens dataset `https://movielens.org/` is an open dataset of reviews so that users can get personalized recommendation. The original datasets used here can be downloaded at `https://grouplens.org/datasets/movielens/latest/` but there are also present on the school hadoop at the path `/datasets/movie_small/` for the "small" extract and at `/datasets/movielens/` for the more complete dataset (considering the dataset size it is recommended to test your programs mainly with the small dataset).

The two files of interest for us in these folders are the `ratings.csv` and the `movies.csv`. Both are csv file, which means that they describe tabular data (much like an excel sheet). The *header* of the file `ratings.csv` (the first line) is `userId,movieId,rating,timestamp` which means that there are 4 columns, separated by comma ','. For instance, the second line here is `1,307,3.5,1256677221` which indicates that the user with id 1 gave the rating 3.5 to the movie with id 307 at time `1256677221`. We will not use the timestamps.

The second file we will use is `movies.csv` because it contains the translation from movies id to movie names. The header of this second file is `movieId,title,genres` with a line of this file looking like `1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy`. We will only use the first (`movieId`) and second (`title`) columns of the file. Beware that some movie titles contain commas (which are also used to separate columns).

2 A first Hadoop program computing the average rating for each movie

In this section we will compute the average rating for each movie. To do this we need a simple map reduce phase. During the map each line of the csv file `ratings.csv` will be read, for each line except the header, we will output the `movieId` as the key and the `rating` as the value. During the reduce, for each keys, we will sum all ratings and divide by the number of items.

For the sake of simplicity, I provide you with a template. You need to fill the file `tpt/dk908a/ComputeAverage.java`. To test your program you need to upload your code to the serverm then compile it on the server using the command line `bash compile.sh` and the submit to hadoop with

```
hadoop jar code.jar tpt.dk908a.RunAverage /datasets/movie_small /tmp/$USER
```

3 Computing frequent itemset

Given two movies m_1 and m_2 we say that they *co-appear* when both movies have been graded by a user u with a rating greater than 3.5. The number of times a pair (m_1, m_2) co-appears is the number of users grading both movies with a score higher than 3.5.

Our goal is to determine for each movie m_1 the movie m_2 that co-appears the most with m_1 (the case of equality can be treated by selecting m_2 at random among the movie co-appearing the most).

For the sake of simplicity, it is recommended to decompose the problem into the following Map Reduce jobs :

1. The first job consists in computing the list of movies graded above 3.5 by a given user; we recommend to use the file `ExtractUserMovies`.
2. The second job consists in computing the number of times a given pairs appears in the dataset; the file associated is `CountPairs`.
3. The third one selects the best movie for a given movie (file `SelectPairs`).

4 Fancy output

You can add to the two previous MapReduce chains a fourth and a fifth phase dedicated to translating movie ids to their actual names. Those two Map Reduce can be made with the same Map and Reduce functions. First we add a dummy key at the beginning of each line in the third Map Reduce job and then use the following reduce and maps :

$$Map(\text{Ratings.csv}) : (key, movie_1, movie_2) \rightarrow (movie_2, movie_1)$$

$$Map(\text{Movie.csv}) : (movie, title) \rightarrow (movie, (", title))$$

```
reduce(k,l):
  toTranslate = {}
  title = ""
  for each v in l:
    if v is a pair:
      name = v.second
    otherwise:
      toTranslate.append(v)
```