Lab 5 – IMDB's Most Frequent Co-stars

CC5212-1

April 9, 2015

No more counting words!

Today we count the stars.

More specifically, we will count co-stars in movies that are listed in IMDb.[1] Your mission ... should you choose to accept it ... is to create a set of MapReduce jobs that count the pairs of actors or actresses that have co-starred in the most movies together. Hopefully it shouldn't be impossible.

The goal of the lab is to demonstrate that you can now independently code MapReduce jobs to process large amounts of data. Since we did something very similar on the board in the lecture on Monday and since last Wednesday, we already saw the syntax and steps needed to code and run a MapReduce job, in this lab I'm not going to give you detailed step-by-step instructions. Instead, I'll give you the instructions to get you started and will point you to the material that will help you complete the lab. The rest will be up to you.

- The instructions for logging into the server, for accessing and uploading data to HDFS, for building your code and running it, etc. are available from last weeks' instructions: `http://aidanhogan.com/teaching/cc5212-1/doc/lab4.pdf`.
- The data you need are on HDFS in the `/uhadoop/shared/imdb/` folder. The `imdb-stars.tsv` file is about 1GB and contains 13 million roles played by different actors in IMDb (pretty much the full database). Have a look inside the start of the file.[2] The header (not included in the file) is as follows:

Star Name	Movie Name	Year	Movie Number	Movie Type	Episode Name	Starring As	Role
...	...	...	...	...	...	...	...

Columns are tab delimited. Some values may not apply and may be `null`.

Star Name is the name of the star. The file is sorted alphabetically so you might see stars with weird names in there at the start of the file.

Movie Name is the name of the movie or tv series, etc., that the star appears in.

Year is ... well yes, the year the movie was released.

Movie Number is used when a movie with the same name appears in the same year (e.g., `http://www.imdb.com/title/tt0801505/` is the second movie called "Crash" listed in 2004 so it will have `II` here). Often this will be `null` (if there was only one movie with that name in that year).

Movie Type is the type of movie ... if it's a theatrical movie, a TV movie, a TV series, etc.[3]

Episode Name is the name of an episode if it was a TV series.

Starring As is the name of the actor/actress in the credits.

Role is the character they played.

[1] The Internet Movie Database ... `http://imdb.com/` ... I'm sure many of you know it?

[2] `hdfs dfs -cat [file] | more` pressing spacebar to scan, Ctrl+C to end.

[3] In the code later, the options will appear in `org.mdp.imdb.ActorMovieParser.MovieRole.MovieType`

We will mainly focus on the first five columns. The Star Name is unique for a person. **However, for movies, Movie Name is not unique ... only the combination of Movie Name, Year and Movie Number are enough together to uniquely identify a movie.** To make a unique key for a movie, you'll need to concatenate the values of these three columns. We only want to consider movies: specifically you should only consider input lines where the Movie Type is equal to `THEATRICAL_MOVIE`; if it is something else, skip it. Let's look for info about a specific actor: type `hdfs dfs -cat /uhadoop/shared/imdb/imdb-stars.tsv | grep -e "^Pacino, Al" | more`, flick through some results with spacebar, `Ctrl+C` to end.

- Download the code project from `http://aidanhogan.com/teaching/cc5212-1/code/mdp-lab5.zip` and open it in Eclipse. In this project there is a `Main` class, an example of the `WordCount` for you to adapt, and the class `ActorMovieParser`, which was used to parse the raw data from IMDb.[4] You do not need to do anything with `Main` or `ActorMovieParser` ... feel free to ignore them.

- The exercise for today is to adapt the `WordCount` example provided in the package[5] to count the number of movies that each pair of actors/actresses have co-starred in. The output should look something like:

`De Niro, Robert##Pacino, Al`	x
`Cage, Nicholas##Kidman, Nicole`	y
...	...

  ... where x is the number of theatrical movies that both Robert De Niro and Al Pacino star in, etc. (`##` is just a delimiter; you can use a tab or anything unambiguous). Some tips:

  - You will have to create your own classes today. Follow the `WordCount` example.
  - As mentioned previously, only include theatrical movies and make sure the movie key you use is unique (see above).
  - If you need multiple jobs, it's okay to run them manually one-by-one. Check the output of one before moving onto the next. The output of one task becomes the input of the next.
  - When you're calling your jar from the command line, the first argument is the name of the class with the main method you need. The second argument should be the location of the input file. The third argument should be the location of the output file.
  - Do not copy the input data into your personal folder ... it's too big to make 30–35 copies. Just pass the input location `/uhadoop/shared/imdb/imdb-stars.tsv`. But do write the output to your personal folder on HDFS (e.g., `/uhadoop/[username]/imdb/out1/`).

- Optional: if you have the time, try using MapReduce to get the top-10 most frequent co-stars. You can do this by sorting everything and taking the head of the results, or by implementing the top-k pattern we spoke about in the lecture on Monday.

- Add the new class files that you created to a ZIP and submit to u-cursos before the lecture on Monday.

[4] See the `actors.1k.list` file in the project root folder for a small example of the raw format that came from IMDb. This would be really hard to work with. The `ActorMovieParser` class converts the raw IMDb data into the tabular format that you will work with.

[5] Note that this class uses a newer version of the Hadoop API so there might be minor differences from last week.