HWA. MapReduce Programming

Ignacio M. Llorente, v1.0 - 27 October 2020

Abstract

The objective in this homework is to develop practical skills in parallel data processing for computational and data science. The focus is on scaling data-intensive computations using functional parallel programming over distributed architectures. The goal of the exercises is to practice some of the most frequent MapReduce programming patterns. MapReduce is more of a framework than a tool. You have to fit your application into the execution pattern of map and reduce, which in some situations might be challenging. Design patterns can make application design and development easier allowing problems to be solved in a reusable and general way.

Guidelines

- The **datasets** needed to do the exercises are available for download from Google Classroom.
- **MapReduce/Hadoop**
  - Both the mapper and the reducer should be python executable scripts that read the input from stdin (line by line) and emit the output to stdout.
  - Exercises 1-5 provide the linux command that will be used to test the scripts. It is very important that you follow an efficient MapReduce programming pattern.
- **Submission**

Any computing experiment that cannot be replicated cannot be considered as a valid submission

  - Upload on **Google Classroom** the files specified in each assignment.
  - The grade on this assignment is **10% (100 points) of the final grade.**
  - There are **no late days.**

Table of Contents

1. Distributed Grep (20 points)

Develop a distributed version of the grep tool to search for words in very large documents. Use the design patterns explained in class. The output should be the lines that contain a given *word*.

You should use as an input file the input text used in the word count example described in class (eBook of Moby Dick). Your scripts will be tested using the following linux command.

```
$ ./P1_mapper.py word < input.txt | sort | ./P1_reducer.py
```

Submission

- `P1_mapper.py`: Mapper script
- `P1_reducer.py`: Reducer script

2. Count URL Access Frequency (20 points)

Develop a MapReduce job to find the frequency of each URL in a web server log. Use the design patterns explained in class. The output should be the URLs and their frequency.

You should use as input file the sample Apache log file `access_log` (downloaded from http://www.monitorware.com/en/logsamples/apache.php). Your scripts will be tested using the following linux command.

```
$ ./P2_mapper.py < access_log | sort | ./P2_reducer.py
```

Submission

- `P2_mapper.py`: Mapper script
- `P2_reducer.py`: Reducer script

3. Stock Summary (20 points)

Write a MapReduce job to calculate the average daily stock price at close of Alphabet Inc. (GOOG) per year since 2009. Use the design patterns explained in class. The output should be the year and the average price.

You should use as input file the daily historical data `GOOGLE.csv` (downloaded from Yahoo Finance https://finance.yahoo.com/quote/GOOG/history?ltr=1). You have to preprocess the `GOOGLE.csv` file to remove the head line. Your scripts will be tested using the following linux command.

```
$ ./P3_mapper.py < GOOGLE.csv | sort | ./P3_reducer.py
```

Submission

- `P3_mapper.py`: Mapper script
- `P3_reducer.py`: Reducer script

4. Movie Rating Data (20 points)

GroupLens Research has collected and made available rating data sets from the MovieLens web site (http://movielens.org). The data sets were collected over various periods of time, depending on the size of the set. Before using these data sets, please review their README files for the usage licenses and other details. You can use as input file the small version of the dataset `ml-latest-small.zip` (downloaded from https://grouplens.org/datasets/movielens/).

Develop a MapReduce job to show movies ids with an average rating in the ranges:
Range 1: 1 or lower
Range 2: 2 or lower (but higher than 1)
Range 3: 3 or lower (but higher than 2)
Range 4: 4 or lower (but higher than 3)
Range 5: 5 or lower (but higher than 4)

Use the design patterns explained in class. The job should have two MapReduce phases. The output of the first phase should be the movie ids and their average rating. The output of the second phase should be ranges and the movie's ids.

Your scripts will be tested using the following linux command.

```
$ ./P4a_mapper.py < ratings.csv | sort | ./P4a_reducer.py | ./P4b_mapper.py
| sort | ./P1b_reducer.py
```

Submission
- `P4a_mapper.py`: Mapper script first phase
- `P4a_reducer.py`: Reducer script first phase
- `P4b_mapper.py`: Mapper script second phase
- `P4b_reducer.py`: Reducer script second phase

5. Meteorite Landing (20 points)

The NASA's Open Data Portal hosts a comprehensive data set from The Meteoritical Society that contains information on all of the known meteorite landings. The Table `Meteorite_Landings.csv` (downloaded from https://data.nasa.gov/Space-Science/Meteorite-Landings/gh4g-9sfh) consists of 34,513 meteorites and includes fields like the type of meteorite, the mass and the year. Write a MapReduce job to calculate the average mass per type of meteorite. Use the design patterns explained in class.

You have to preprocess the `Meteorite_Landings.csv` file to remove the head line and do some data cleansing work. Your scripts will be tested using the following linux command.

```
$ ./P5_mapper.py < Meteorite_Landings.csv | sort | ./P5_reducer.py
```

Submission
- `P5_mapper.py`: Mapper script
- `P5_reducer.py`: Reducer script