INSTITUTE FOR APPLIED COMPUTATIONAL SCIENCE AT HARVARD UNIVERSITY

HARVARD School of Engineering and Applied Sciences

HW: C. Parallel Data Processing

Due Monday, April 15, 2019

Ignacio M. Llorente, v1.0 - 22 March 2019

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.

Contributors

The author is grateful for constructive comments and suggestions from David Sondak, Charles Liu, Matthew Holman, Zudi Lin, Nicholas Stern and Kar-Tong Tan

Guidelines

- The datasets needed to do the exercises are available for download from the course Canvas repository.
- **AWS**
  - **First you should have followed the Guide "First Access to AWS"**. It is assumed you already have an AWS account and a key pair, and you are familiar with the AWS EC2 environment.

> If you are using AWS, we strongly recommend you use the same instance type for all the experiments (**m4.xlarge**) so you can compare the performance results achieved with the different programming models and platforms

- **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.
  - Codes (exercises 1.1 - 1.5) can be easily developed and debugged locally by executing the following linux command.

    ```
    $ mapper.py < input.txt | sort | reducer.py
    ```

  - For the execution on a Hadoop cluster (exercise 1.6) we will use Hadoop streaming on EMR AWS, which is an utility that comes with the Hadoop distribution that allows you to create and run MapReduce jobs with any executable or script as the mapper and/or the reducer. Its is strongly recommended to use **m4.xlarge** instances and firstly follow the Guide "Hadoop Cluster on AWS" in order to get familiar with the Hadoop environment and know how to set up a cluster. Ensure you terminate the cluster after the homework.
- **Spark**
  - Programs should be developed in Python.
  - Install a local version of Spark, by following the Guide "Install Spark Cluster in Local Mode", to develop your programs (exercises 2.1 - 2.5) on a single AWS VM (or your local computer). As Spark's local mode is fully compatible with the cluster mode; programs written and tested locally can be run on a cluster with just a few additional steps.
  - For the execution on a Spark cluster (exercise 2.6) we will use EMR on AWS. Its is strongly recommended to use **m4.xlarge** instances and firstly follow the Guide "Spark Cluster on AWS" on how to set up a cluster, to login, to use the HDFS and to run Spark scripts. Ensure you terminate the cluster after the homework.
- **Submission**

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

  - **Your performance results should be replicable[1] results. Sometimes the equivalent term Repeatability is used for this experimental property, so you should provide ALL the information of the system and the environment needed to repeat your tests.** At the beginning of each PDF answering exercises related to performance (1.6 and 2.6) you must provide, at least:
    - Specs of the system (model, number of CPUs, number of cores per cpu, clock rate, cache memory, and main memory)
    - If using a cluster, number of systems and specs of networking (latency and bandwidth)

[1] The attribute **Replicability** describes the ability to repeat a computer based experiment and to come to the same results and performance.

  - Operating System (Linux distro and kernel version)
  - Compiler (name and version, and flags)
  - Libraries (name and version)
  - Any other configuration needed to replicate the experiments and achieve exactly the same result and performance?

- Upload on **Canvas** the files specified in each assignment.

- The grade on this assignment is **15% (150 points) of the final grade.**

Table of Contents

1. MapReduce Programming (75 points)

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. In these exercises we will practice some of the most frequent **MapReduce programming patterns that have been described in class**. Exercises from 1.1 to 1.5 can be tried locally, 1.6 requires a EMR AWS cluster.

1.1. Distributed Grep (10 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 match a given pattern. You can use as an input file the input text used in the word count example described in class (eBook of Moby Dick).

Submission

- `P11_mapper.py`: Mapper script
- `P11_reducer.py`: Reducer script
- `P11.pdf`: The command line that you used to execute the job and any information required to reproduce the execution

1.2. Count URL Access Frequency (10 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 can use as input file the sample Apache log file `access_log` (downloaded from http://www.monitorware.com/en/logsamples/apache.php).

Submission

- `P12_mapper.py`: Mapper script
- `P12_reducer.py`: Reducer script
- `P12.pdf`: The command line that you used to execute the job and any information required to reproduce the execution

1.3. Stock Summary (10 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 can use as input file the daily historical data `GOOGLE.csv` (downloaded from Yahoo Finance https://finance.yahoo.com/quote/GOOG/history?ltr=1)

Submission

- `P13_mapper.py`: Mapper script

- `P13_reducer.py`: Reducer script
- `P13.pdf`: The command line that you used to execute the job and any information required to reproduce the execution

1.4. Movie Rating Data (10 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.

Submission

- `P14a_mapper.py`: Mapper script first phase
- `P14a_reducer.py`: Reducer script first phase
- `P14b_mapper.py`: Mapper script second phase
- `P14b_reducer.py`: Reducer script second phase
- `P14.pdf`: The command line that you used to execute the job and any information required to reproduce the execution

1.5. Meteorite Landing (10 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.

Submission

- `P15_mapper.py`: Mapper script
- `P15_reducer.py`: Reducer script
- `P15.pdf`: The command line that you used to execute the job and any information required to

reproduce the execution

1.6. Parallel Execution (25 points)

Use the Distributed Grep MapReduce code developed in exercise 1.1 and the large version of the movielens data set (224MB) to show the ratings with 5.0 stars.

- Execute the MapReduce code on a cluster with 2, 4 and 8 m4.xlarge instances and calculate the speedup. You can start with a cluster of 2 nodes and then dynamically resize it to include more nodes (Hardware tab).
- To achieve better performance you have to try tuning different parameters, like the number of map and reduce tasks, on the larger cluster with 8 nodes

Submission
- `P16.pdf`: Description of the experiment, discussion about the performance and speed-up, and description of any tuning developed

2. Spark Programming (75 points)

In these exercises we will practice Spark programming with special emphasis on data parallel processing. Exercises from 2.1 to 2.5 can be tried locally, 2.6 requires a EMR AWS cluster.

2.1. Distributed Grep (10 points)

Develop a Spark version of the grep tool to search words in very large documents. The output should be the lines that match a given pattern. You can use as input file the input text used in the word count example described in class (eBook of Moby Dick).

Submission
- `P21_spark.py`: Spark script
- `P21.pdf`: The command line that you used to execute the job and any information required to reproduce the execution

2.2. Count URL Access Frequency (10 points)

Develop a Spark script to find the frequency of each URL in a web server log. The output should be the URLs and their frequency. You can use the `access_log` file from Problem 1.2.

Submission
- `P22_spark.py`: Spark script
- `P22.pdf`: The command line that you used to execute the job and any information required to reproduce the execution

2.3. Stock Summary (10 points)

Write a Spark script to calculate the average daily stock price at close of Alphabet Inc. (GOOG) per year since 2009. The output should be the year and the average price. You can use the `GOOGLE.csv` file from Problem 1.3.

You may need to use **DataFrames** to simplify the processing of the data. A DataFrame is a Dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood.

Submission
- `P23_spark.py`: Spark script
- `P23.pdf`: The command line that you used to execute the job and any information required to reproduce the execution

2.4. Movie Rating Data (10 points)

Develop a Spark version of the job to show movies 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)

You can use the `ml-latest-small.zip` file from Problem 1.4. You may need to use **DataFrames** to simplify the processing of the data.

Submission
- `P24_spark.py`: Spark script
- `P24.pdf`: The command line that you used to execute the job and any information required to reproduce the execution

2.5. Meteorite Landing (10 points)

Develop a Spark version of the job to calculate the average mass per type of meteorite. You can use the `Meteorite_Landings.csv` dataset with 34,513 meteorites from Problem 1.5 downloaded from the NASA's Open Data Portal. You may need to use **DataFrames** to simplify the processing of the data.

Submission
- `P25.py`: Spark script
- `P25.pdf`: The command line that you used to execute the job and any information required to reproduce the execution

2.6. Parallel Execution (25 points)

Use the Distributed Grep Spark script developed in exercise 2.1 and the large version of the movielens data set `ml-latest.zip` (224MB) to show the ratings with 5.0 stars.

- Evaluate performance and speed-up when using up to 4 cores in the local mode installation of Spark on a m4.xlarge instance.
- Evaluate performance and speed-up when using a cluster with 2 and 4 m4.xlarge instances and 1 and 2 cores per node. You can start with a cluster of 2 nodes and then dynamically resize it to include more nodes (Hardware tab).
- To achieve better performance you have to try tuning different parameters of the Spark configuration on the larger cluster with 4 nodes.

Submission
- `P26.pdf`: Description of the experiment, discussion about the performance and speed-up, and description of any tuning developed