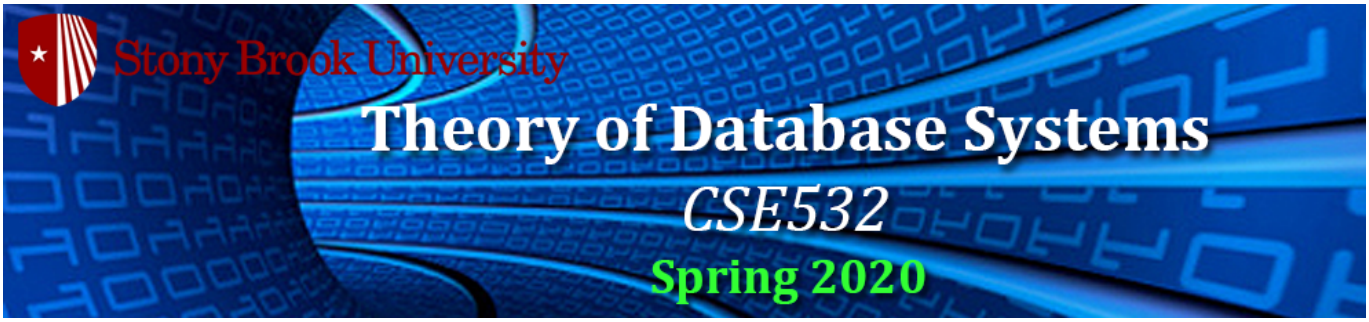

Home

CSE532

Home
Syllabus
Schedule
Homework
Textbooks
Examples
Instructor
Resources

Homework >

Homework4hadoop

Please start early. Although it may seem trivial, the programming aspect might be time sensitive and especially debugging is NOT easy.
Total credit: 25; extra credit: 10

MapReduce on Pseudo-Distributed Hadoop

This is just to warm you up. In reality, Hadoop is usually deployed over actual distributed cluster nodes. However, for the purpose of this course and get an idea of the environment, we're starting with a local single node Hadoop setup (HDP sandbox, Cloudera quickstart VM or Docker).

Even in real world, it is always a good idea to start small and build - Keep It Simple, Stupid (KISS Principle)

COVID-19 Data Analysis

As of April 8, 2020, 1 506 061 cases and 88 085 deaths due to coronavirus disease 2019 (COVID-19), caused by the novel severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2), had been reported worldwide.

The epidemic began in mainland China, with a geographical focus in the city of Wuhan, Hubei. However, on Feb 26, 2020, the rate of increase in cases became greater in the rest of the world than inside China. Substantial outbreaks are occurring in the USA (425 469 cases), Spain (148 220 cases) and Italy (139 422 cases) and geographical expansion of the epidemic continues.

Task Requirements

(1) [5 points] Building on the simple WordCount example done in class and Hadoop tutorial, your task is to perform simple processing on provided COVID-19 dataset.

- The first task is to count the **total number of reported cases** for every country/location till April 8th, 2020 (NOTE: There data does contain case rows for Dec 2019, you'll have to filter that data)
- Name your program `Covid19_1.java`
- Program arguments description
  - The HDFS path to your input data file
  - [true | false] Include "World" data in the result or not. True will include total number of reported cases for "World" in the result, False will ignore the rows with `location/country = world`
  - The HDFS output path for your program. (NOTE: You should remove the output path after every execution of your program. Hadoop cannot start a job if output directory is already created)

(2) [10 points] General analysis usually require processing in multiple modes. For instance, for COVID-19 data, analysis can be performed on different date ranges.

- Your second task is to modify your program to report **total number of deaths** for every location/country in between a given range of dates.
- Name your program `Covid19_2.java`
- Note that you'll have to perform error handling for invalid dates. Input dataset contains data from Dec, 2019 to April, 2020. Perform your error handling accordingly
- Also, your result should contain data including start and end dates
- Program arguments description

- The HDFS path to your input data
- Start date (YYYY-MM-DD)
- End date (YYYY-MM-DD)
- The HDFS output path for your program. (NOTE: You should remove the output path after every execution of your program. Hadoop cannot start a job if output directory is already created)

(3) [10 points] Often times we can encounter cases where we have to join multiple datasets for analysis. One particular case is when one of the joining dataset is static and may have to be read frequently multiple times. In such cases, Hadoop provides DistributedCache to efficiently manage read-only files and avoid unnecessary copying.

- Your third task is to output the total number of cases per 1 million population for every country
- Name your program `Covid19_3.java`
- Add `population.csv` file to Hadoop `DistributedCache` (See Examples)
- Use formula

```
(total_number_of_country_cases/country_population) * 1,000,000
```

- Program arguments description
  - The HDFS path to your input data file (`covid19_full_data.csv`)
  - The HDFS path to `populations.csv`
  - The HDFS output path for your program. (NOTE: You should remove the output path after every execution of your program. Hadoop cannot start a job if output directory is already created)

Dataset

Dataset Description

The dataset contains COVID-19 case information from the start of this year till April 8, 2020 for most of the countries of the world in .csv (comma separated value). Each line consists of number of new cases, new deaths, total number of cases and total deaths for a given country on given day. The data format is as follows

```
[date],[location/country],[new_cases],[new_deaths]
```

The second dataset, required for task-3, consists of population information for world countries as of the year 2020. The format of the data is as follows

```
[country],[location],[continent],[population_year],[population]
```

Dataset Download

You can download sample dataset using following terminal commands

(covid19 dataset for task-1, task-2 and task-3)

```
wget http://bmidb.cs.stonybrook.edu/publicdata/cse532-s20/covid19_full_data.csv
hdfs dfs -mkdir -p /cse532/input
hdfs dfs -put covid19_full_data.csv /cse532/input/
```

(population dataset for task-3)

```
wget http://bmidb.cs.stonybrook.edu/publicdata/cse532-s20/populations.csv
hdfs dfs -mkdir -p /cse532/cache
hdfs dfs -put populations.csv /cse532/cache/
```

Examples

To execute Hadoop jobs using Java API (Terminal Commands)

```
hadoop Covid19.jar Covid19_1 /cse532/input/covid19_full_data.csv true /cse532/output/
hadoop Covid19.jar Covid19_1 /cse532/input/covid19_full_data.csv false /cse532/output/
hadoop Covid19.jar Covid19_2 /cse532/input/covid19_full_data.csv 2020-01-01 2020-03-31 /cse532/output/
```

To view output (Terminal Commands)

(list contents of HDFS output directory)

```
hdfs dfs -ls /cse532/output/
```

(print out the contents of output files to terminal)

```
hdfs dfs -cat /cse532/output/part-*
```

Distributed Cache (Code)

(Add following to your main method, after putting `population.csv` on hdfs)

```
Configuration conf = getConf();
Job job = Job.getInstance(conf);
job.addCacheFile(new Path("hdfs://localhost:9000/populations.csv").toUri());
```

(Add following function/method to your mapper and/or reducer class)

```
public void setup(Context context){
    URI[] files = context.getCacheFiles();
    for (URI file : files){
        // read population.csv file data
    }
}
```

Submission

Hadoop

1. **Submit Covid19_1.java** for task-1 (It should contain the map and reduce functions)
2. **Submit Covid19_2.java** for task-2 (It should contain the map and reduce functions)
3. **Submit Covid19_3.java** for task-2 (It should contain the map and reduce functions)

[Extra Credit] Pseudo Distributed Spark

Similar to Hadoop, Apache Spark also supports pseudo distributed mode aka. single node cluster or Spark standalone mode. While this obviously would not have performance comparable to Spark deployed on actual distributed cluster nodes, the processing pipeline is more or less similar in both modes. The code written for pseudo distributed mode can safely be considered deployable on actual distributed cluster without much modifications.

Task Requirements

(1) [4 points] Implement Task-2 on Spark using the same dataset with same arguments
(2) [4 points] Implement Task-3 on Spark using the same dataset with same arguments
(3) [2 points] Time and compare performance of your Spark and Hadoop implementations for Task-2 and Task-3

Examples

To execute Spark jobs using Java API (Terminal Commands)

```
spark-submit --class Covid19_1 SparkCovid19.jar /cse532/input/covid19_full_data.csv /cse532/output/
spark-submit --class Covid19_1 --master local[2] SparkCovid19.jar /cse532/input/covid19_full_data.csv false
/cse532/output/
spark-submit --class Covid19_2 SparkCovid19.jar /cse532/input/covid19_full_data.csv 2020-01-01 2020-03-31
/cse532/output/
```

To execute Spark jobs using Python API (Terminal Commands)

```
spark-submit SparkCovid19_1.py /cse532/input/covid19_full_data.csv true /cse532/output/
spark-submit --master local[2] SparkCovid19_1.py /cse532/input/covid19_full_data.csv false /cse532/output/
spark-submit SparkCovid19_2.py /cse532/input/covid19_full_data.csv 2020-01-01 2020-03-31 /cse532/output/
```

Submission

1. **Submit SparkCovid19_1.{java,py,scala}** (Depending on your implementation language)
2. **Submit SparkCovid19_2.{java,py,scala}** (Depending on your implementation language)
3. **Submit performance.txt** explaining how you timed both implementations and results

Comments

You do not have permission to add comments.