UE19CS322 Big Data Assignment 1

Analysis of US Road Accident Data using MapReduce

This is the first assignment for the UE19CS322 Big Data Course at PES University. The assignment consists of 2 tasks and focuses on running MapReduce jobs to analyse data recorded from accidents in the USA.

The files required for the assignment can be found here.

Assignment Objectives and Outcomes

1. This assignment will help students become familiar with the Map Reduce programming environment and the HDFS.
2. At the end of this assignment, the student will be able to write and debug MapReduce code.

Ethical practices

Please submit original code only. You can discuss your approach with your friends but you must write original code. All solutions must be submitted through the portal. We will perform a plagiarism check on the code and **you will be penalised if your code is found to be plagiarised**.

The Dataset

You will be provided with a link to the dataset on PESU Forum. You will be working with the following set of attributes.

Key	Type	Description
Severity	integer	Severity of the accident (between 1 - 4)
Start_Time	datetime	Start time of accident in local time zone
Start_Lat	float	Latitude as GPS coordinate of the start point
Start_Lng	float	Longitude as GPS coordinate of the start point
Description	string	Natural language description of the accident
Visibility(mi)	float	Visibility (in miles) during the accident
Precipitation(in)	float	Precipitation amount in inches, if there is any
Weather_Condition	string	Weather condition during the accident - rain, snow, thunderstorm, fog, etc
Sunrise_Sunset	String	Shows the period of day (i.e. day or night) during the accident

Software/Languages to be used:

1. Python `3.8.x`
2. Hadoop `v3.2.2` only

Marks

Task 1: 2 marks

Task 2: 2 marks

Report: 1 mark

Tasks Overview:

1. Load the data into HDFS.
2. Create `mapper.py` and `reducer.py` for Task 1 and Task 2
3. Run your code on the sample dataset until you get the right answer
4. Submit the files to the portal
5. Submit one page report based on the template and answer the questions on the report

Submission Link

Portal for Big Data Assignment Submissions

Submission Date

16th September, 11:59 PM

Submission Guidelines

You will need to make the following changes to your `mapper.py` and `reducer.py` scripts to run them on the portal

1. Include the following `shebang` on the first line of your code

```
#!/usr/bin/env python3
```

2. Convert your files to an executable

```
chmod +x mapper.py reducer.py
```

3. Convert line breaks in `DOS` format to `Unix` format (this is **necessary** if you are coding on Windows - your code will not run on our portal otherwise)

```
dos2unix mapper.py reducer.py
```

Check out a detailed list of submission guidelines here.

Task Specifications

Task 1

Problem Statement

Find record count per hour

Description

Find the number of accidents occuring per hour that satisfy a set of conditions and display them in sorted fashion.

All the following conditions must be satisfied by a record.

Attribute	Condition
Description	Accident should result in either a “lane blocked”, “shoulder blocked” or an “overturned vehicle”
Severity	>= 2
Sunrise_Sunset	Night
Visibility(mi)	<= 10
Precipitation(in)	>= 0.2 inches
Weather_Condition	Should either be "Heavy Snow", "Thunderstorm", "Heavy Rain", "Heavy Rain Showers" or "Blowing Dust"

Comments

Ignore records which do not satisfy the mentioned conditions. You do not require any command line arguments for this task. Additionally, if any of the required attributes contain `NaN`, ignore the record.

Recommended module:
`datetime`

Output Format

For each hour that contains accident data that satisfies the provided conditions, print the hour followed by the number of accidents in that hour on a separate line. For hours that do not contain any accident records, do not print anything.

Example,

```
3 1
4 1
5 4
6 4
7 1
18 1
19 2
20 1
21 3
22 1
23 1
```

Task 2

Problem Statement

Find record count per city and state

Description

Find the number of accidents occuring per city and state where the distance between the start coordinates of the accident and a given pair of coordinates - (`LATITUDE`, `LONGITUDE`) is within `D`. You will be using Euclidean Distance to find whether the distance calculated is within `D`.

For each record, you will be making a request to http://20.185.44.219:5000/ to obtain the city and state information. The IP accepts only `POST` requests, and expects a JSON payload containing a pair of start coordinates in the following format

```
{
    “latitude”: Start_Lat
    “longitude”: Start_Lng
}
```

The IP will send back a response containing a JSON payload containing city and state information in the following format.

```
{
    “city”: ...
    “state”: ...
}
```

Comments

Ignore records which do not satisfy the required distance condition. Do not round off any numerical values as this may lead to erroneous results. Additionally, if any of the required attributes contain `NaN`, ignore the record.

You are required to take in 3 command line arguments in your `mapper.py` script in the format given below.

`LATITUDE LONGITUDE D`

Recommended module:
`datetime`
`requests` (to be installed via `pip3`)

You will not be allowed to install any other libraries or use any other APIs to execute your code.

Output Format

For each state, you will first have to display the name of the state. Following this, you will have to determine the number of accidents that occur in each city in that state, and display each city's count on a separate line. You do not have to display cities where the count is zero. Finally, display the state again and the total number of accidents for that entire state.

Example,
`LATITUDE` = `40`
`LONGITUDE` = `-66`
`D` = `5.3`

Taking the last state `ME` as an example, the counts for the cities Ellsworth, Hope and Trenton are determined to be `3`, `1` and `1` respectively. Hence, the total count for the state is `5`.

```
MA
Brewster 1
Buzzards Bay 1
Carver 2
Cotuit 1
Duxbury 13
East Falmouth 2
East Freetown 1
East Sandwich 1
East Wareham 1
Eastham 1
Halifax 7
Hyannis 4
Kingston 1
Mashpee 4
Middleboro 1
New Bedford 3
North Dartmouth 2
Orleans 1
Osterville 1
Plymouth 4
Plympton 2
Rochester 1
Sandwich 2
South Dennis 2
Teaticket 1
Truro 2
Wareham 3
West Barnstable 1
West Wareham 1
MA 67
ME
Ellsworth 3
Hope 1
Trenton 1
ME 5
```

Helpful Commands

Running the MapReduce Job without Hadoop

A MapReduce job can also be run without Hadoop. Although slower, this utility helps you debug faster and helps you isolate Hadoop errors from code errors.

```
cat path_to_dataset | python3 mapper.py [command line arguments] | sort -k 1,1 | python3 reducer.py [command line arguments] > output.txt
```

Starting Hadoop

If you are running Hadoop for the first time, run

```
hdfs namenode -format
```

Hadoop can be started using the following command.

```
$HADOOP_HOME/sbin/start-all.sh
```

You can view all the Java processes running on your system using `jps`.
After running `jps` you should see the following processes running (in any order) along with their process IDs:

```
DataNode
SecondaryNameNode
Jps
ResourceManager
NameNode
NodeManager
```

HDFS Operations

The `HDFS` supports all file operations and is greatly similar to the file system commands available on Linux.
You can access `HDFS` on command line using `hdfs dfs` and use the `-` prefix before the file system command to execute general Linux file system commands.

Loading a file into HDFS

A file can be loaded into `HDFS` using the following command.

```
hdfs dfs -put path_to_file /hdfs_directory_path
```

Listing files on HDFS

Files can be listed on `HDFS` using

```
hdfs dfs -ls /hdfs_directory_path
```

Similarly, `HDFS` also supports `-mkdir`, `-rm` and more.

Running a MapReduce Job

A MapReduce job can be run using the following command

```
hadoop jar path-to-streaming-jar-file \
-input path_to_input_folder_on_hdfs \
-output path_to_output_folder_on_hdfs \
-mapper absolute_path_to_mapper.py command_line_arguments \
-reducer absolute_path_to_reducer.py command_line_arguments
```