CISC5950 – Big Data Programming – Spring 2022

Professor: Ying Mao
Project 1 Part 1: NY Parking Violations
Extended due date: 4/5/2022
Student: Yuliya Akchurina

MapReduce Data Analysis in Apache Hadoop
NY Parking Violations

Overview

This project will demonstrate several design examples to analyze data utilizing the MapReduce programming model running on a three node Hadoop Cluster hosted in Google Cloud. We will analyze a dataset of NYC Parking Violations provided by the NYC Department of Finance. The analysis will endeavor to provide general insight into the behavior of people who violate NYC parking ordinances. Specifically, we will attempt to answer the following four questions:

a) When are tickets most likely to be issued?
b) What are the most common years and types of cars to be ticketed?
c) Where are tickets most commonly issued?
d) Which color of the vehicle is most likely to get a ticket?

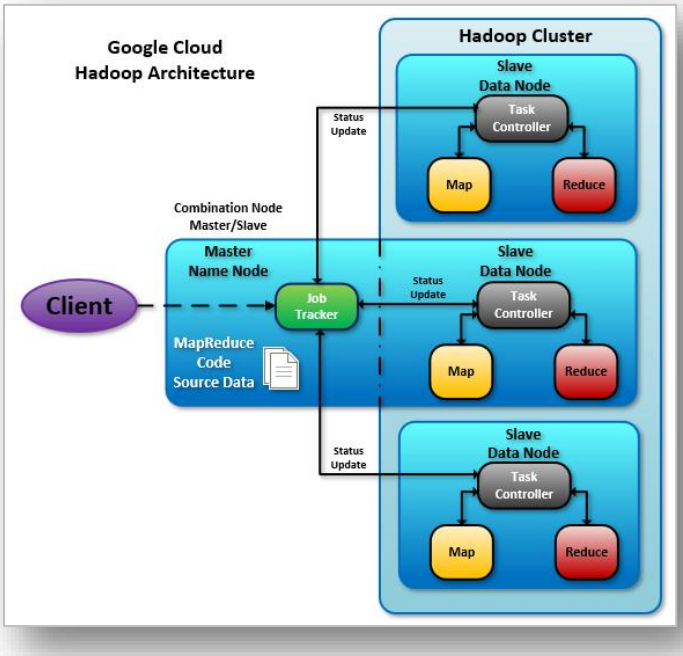
MapReduce Cloud Architecture

Apache Hadoop software is an open-source framework that allows for the distributed storage and processing of large datasets across clusters of computers using simple programming models. Hadoop is designed to scale up from a single computer to thousands of clustered computers, with each machine offering local computation and storage. In this way, Hadoop can efficiently store and process large datasets ranging in size from gigabytes to petabytes of data.

The Hadoop cluster architecture hosted in Google Cloud consists of the following components:

- 1 Combination Master-Slave Node = **Name node/Data node**
- 2 Slave Nodes = **Data Nodes**

The solution source code, developed to resolve the answers to the posed questions and the source data file, are stored and executed on the Name node. The Job Tracker is responsible for execution of the MapReduce job submitted.



The MapReduce process goes through four phases of execution:

Input Splits:

An input to a MapReduce in Big Data job is divided into fixed-size pieces called input splits Input split is a chunk of the input that is consumed by a single map

Mapping

This is the very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values.

Shuffling

This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output.

Reducing

In this phase, output values from the Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. This phase aggregates the complete dataset.

Dataset:

The dataset is publicly available on Link of [NYC Parking Violations Data]. The data used for analysis is fiscal year 2021. Figure 1 presents a snapshot view of the source data.

Summon..	Plate ID	Registra...	Plate Type	Issue Date	Violatio...	Vehicle ...	Vehicle ...	Issuing ...	Street C...	Street C...	Street C...	Vehicle E..	Violatio...
1471497410	HZH8177	NY	PAS	07/02/2020	20	SUBN	NISSA	P	62200	33525	33640	20220510	100
1471497630	JCX5781	NY	PAS	06/27/2020	20	P-U	DODGE	P	60790	34190	34240	20200909	100
1471497641	HEK2391	NY	PAS	06/27/2020	20	SUBN	KIA	P	60790	40404	40404	20220306	100
1471497653	GWY9859	NY	PAS	06/27/2020	20	SUBN	JEEP	P	60790	34190	34240	20210506	100
1471497665	HEZ5501	NY	PAS	06/27/2020	20	SUBN	SUBAR	P	60790	34190	34240	20220524	100
1471497677	JCG3480	NY	PAS	06/27/2020	20	SDN	ME/BE	P	60790	34190	34240	20201028	100
1471497689	HPE5961	99	PAS	07/02/2020	10	SUBN	FORD	P	64151	49295	48795	20210531	100
1471497835	GHP8476	NY	PAS	06/28/2020	10	SDN	TOYOT	P	64151	15635	49295	20210604	100
1471497847	JRE7501	NY	PAS	06/28/2020	10	SUBN	TOYOT	P	64151	15635	49295	20220315	100
1471497859	JDN9663	NY	PAS	06/28/2020	10	SUBN	BMW	P	64151	65617	15635	20201227	100
1471497884	GAR7305	NY	PAS	06/28/2020	10	SDN	SMART	P	64151	49295	65617	20210417	100
1471497896	JMG9616	NY	PAS	06/28/2020	10	SDN	HONDA	P	64151	49295	65617	20211020	100
1471497902	JNT9012	NY	PAS	06/28/2020	10	SUBN	SUBAR	P	64151	29790	49295	20220108	100
1471497938	JBH1504	NY	PAS	06/28/2020	10	SDN	HONDA	P	64151	29790	49295	20211023	100
1471497951	KHW0520	PA	PAS	06/28/2020	10	SDN	TOYOT	P	64151	29790	49295	20210228	100

Figure 1: snapshot of the data

The source data format is a csv file with 43 columns and 14,955,523 rows and just over 2GB in size.

Only selected relevant columns will be used in the computations to answer the 4 questions. The dataset csv file was uploaded to the Google Cloud Cluster and stored on the Name node:

/mapreduce-test/mapreduce-test-python/project1/ Parking_Violations_Issued_-_Fiscal_Year_2021.csv

Solutions

The mapper and reducer solution architecture for the parts **a**, **b**, **c**, and **d** are similar. Differences in the coded solutions account for the various required columns in the source dataset. To successfully run this code on the Google cloud cluster nodes, Python3 is required to support the string operator function that is not available on Python2.

Solution Part a

a) When are tickets most likely to be issued?

To answer this question, we are going to find the most commonly occurring date and time of the tickets issued. The columns required are:

- "**Issue Date**"
- "**Violation Time**"

Task Scheduling Control

Submission of the MapReduce code is controlled by a bash script run file named **test.sh**. This script coordinates the submission of the map and reduce jobs along with the source data file to the Hadoop Streamer and HDFS. The run file used for this solution is uploaded and stored on the name node in:

/mapreduce-test/mapreduce-test-python/Part1Q1/test.sh

Mapping

The mapper code is uploaded and stored in:

/mapreduce-test/mapreduce-test-python/Part1Q1/mapper.py

mapper.py reads csv data file as standard input stream, line-by-line, and performs the following steps for each line:

- Strips whitespace
- Splits the columns on a comma
- Extracts data from columns "Issue Date" – column 4, and "Violation Time" – column 19
- Checks that the values are not empty
- Prints the issue date and violation time and count of 1 for each occurrence as standard output.
  Key = issue date, violation time Value = 1
  i.e., **09/12/2020,1025A: 1**

Shuffling & Sorting

Upon completion of the mapper process, Hadoop initiates an intermediate process to receive the mapper output, sort generate a key-value list and sort the list. Data is then handed off to the reducer.

Reducing

The reducer code is uploaded and stored in:

/mapreduce-test/mapreduce-test-python/Part1Q1/reducer.py

The reducer receives the streaming results of the mapper via standard input stream, and performs the following:

- Strips whitespace
- Splits the passed-on key and value on a tab into two variables:
  - **date_time**
  - **count**
- Creates a dictionary where the key is a unique combination of the **date** and **time** values, with the value being equal to the aggregate sum of occurrences of the ticket.

- Upon completion of aggregation in the reducer the dictionary is sorted in descending order by aggregate sum and most frequent date, then time is printed along with the corresponding count to standard output. The result is printed as standard output.

Results Part a

The output of the MapReduce process is:

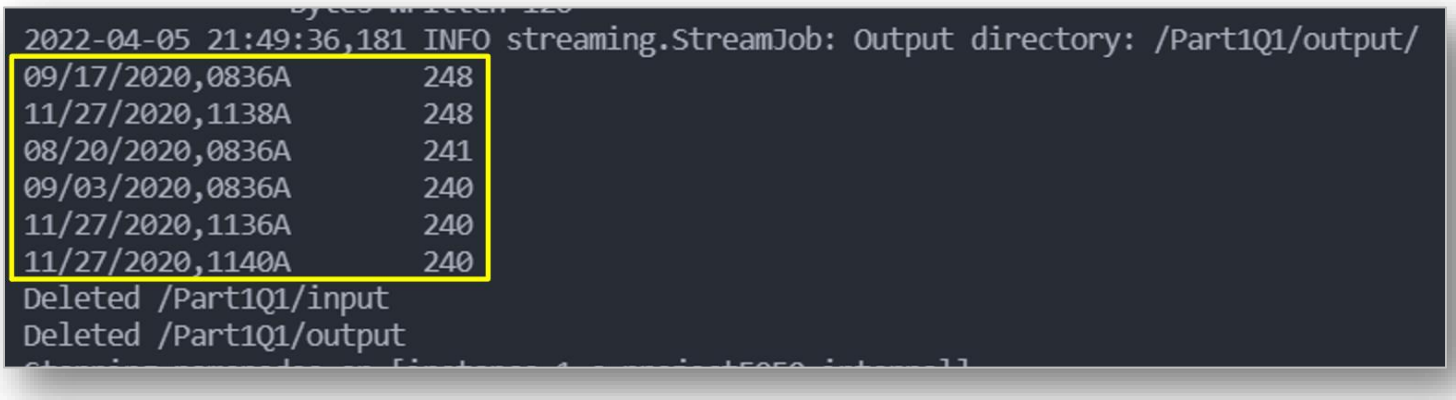


Figure 2: Question A Results

Having selected a range of top 3 dates with top tickets issued we can make several inferences regarding periods of high violations. November 27, 2020, has the highest overall issuance of summons by police with 728. Checking the calendar, we find that this date is Black Friday, the busiest shopping day of the year. We can infer that due to increased traffic from shoppers seeking Black Friday deals, and limited parking on NYC streets, shoppers may have taken extra risk in parking their vehicles to obtain deals.

The remaining top dates listed:

- 8/20/2020 8:36 AM
- 9/3/2020 8:36 AM
- 9/17/2020 8:36 AM

All occur on Thursdays. It is difficult to make any accurate inference with the day Thursday with respect to parking violations. It appears that an automated system may have batch reported violation counts from the previous day as they all have the identical timestamp of 8:36 AM.

Solution b

b) What are the most common years and types of cars to be ticketed?

To answer the question, we will determine the highest aggregate number of violations for given years and vehicle type.

The columns required are:

- **"Vehicle Body Type"**
- **"Vehicle Year"**

Task Scheduling Control

Submission of the MapReduce code is controlled by a bash script run file named **test.sh**. This script coordinates the submission of the map and reduce jobs along with the source data file to the Hadoop Streamer and HDFS. The run file used for this solution is uploaded and stored on the name node in:

/mapreduce-test/mapreduce-test-python/Part1Q2/test.sh

Mapping

The mapper code is uploaded and stored in:

/mapreduce-test/mapreduce-test-python/Part1Q2/mapper.py

mapper.py reads csv data file as standard input stream, line-by-line, and performs the following steps for each line:

- Strips whitespace
- Splits the columns on a comma
- Extracts data from columns “Vehicle Body Type”- column 6 and “Vehicle Year” – column 35
- Checks that the values are not empty. Filters the vehicle year to be between 1885 and 2022. The first car was manufactured in 1885 and the dataset is for 2021 fiscal year, therefore any year value that is outside of this range is invalid.
- Prints the vehicle body type and vehicle year and count of 1 for each occurrence as standard output. Key: vehicle body type and vehicle year. Value: 1.

  Key = vehicle body type, Vehicle Year Value = 1

  i.e., **PAS,2018: 1**

Shuffling & Sorting

Upon completion of the mapper process, Hadoop initiates an intermediate process to receive the mapper output, sort generate a key-value list and sort the list. Data is then handed off to the reducer.

Reducing

The reducer code is uploaded and stored in:

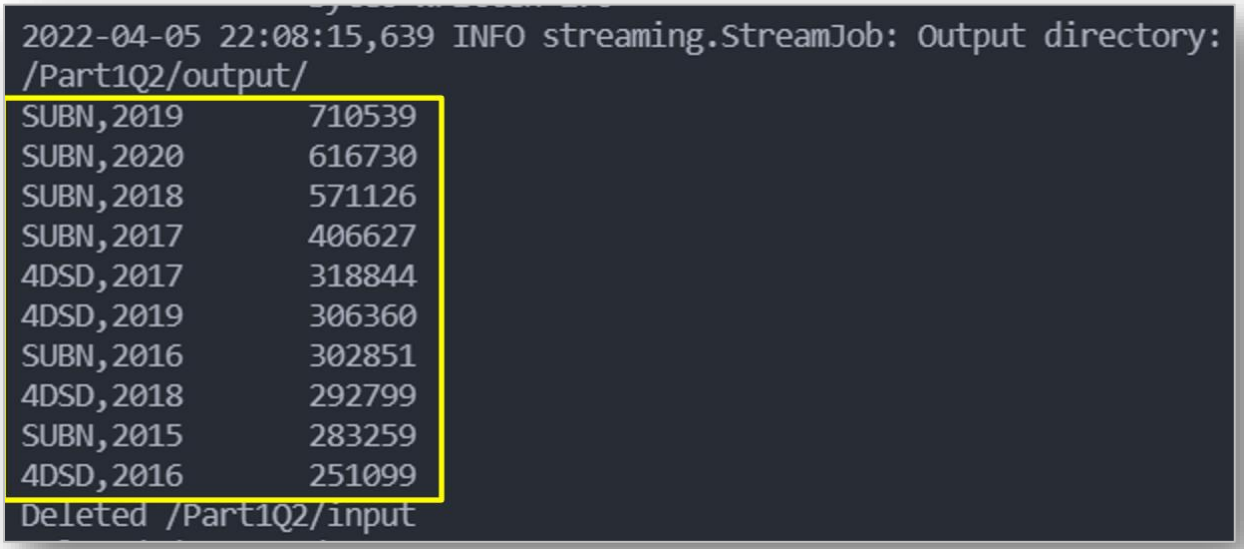
/mapreduce-test/mapreduce-test-python/Part1Q2/reducer.py

The reducer receives the streaming results of the mapper via standard input stream, and performs the following:

- Strips whitespace
- Splits the passed-on key and value on a tab into two variables **type_year** and **count**.
- Creates a dictionary where **type of vehicle** and **vehicle year** is a key and sum of count of occurrences is value. Once summation is completed the dictionary is sorted in descending order by sum of count and the most frequent **type of vehicle** and **vehicle year** is printed along with the corresponding count. The result is printed as standard output.

Results Part b

The output of the MapReduce process is:



By selecting the top 10 occurrences of vehicle type we can see that these are all larger passenger vehicles. Given that many people in NYC don't drive, we can infer that the high rate of violations among larger passenger vehicles could be luxury or high car services, such as Uber. One can routinely see these vehicles double parked on the busy streets on NYC.

Solution c

c) Where are tickets most commonly issued?

To answer the question, we will determine the highest aggregate number of violations by location attributes.

The columns required are:

- **"House Number"**
- **"Street Name"**

Task Scheduling Control

Submission of the MapReduce code is controlled by a bash script run file named **test.sh**. This script coordinates the submission of the map and reduce jobs along with the source data file to the Hadoop Streamer and HDFS. The run file used for this solution is uploaded and stored on the name node in:

/mapreduce-test/mapreduce-test-python/Part1Q3/test.sh

Mapping

The mapper code is uploaded and stored in:

/mapreduce-test/mapreduce-test-python/Part1Q3/mapper.py

mapper.py reads csv data file as standard input stream, line-by-line, and performs the following steps for each line:

- Strips whitespace
- Splits the columns on a comma
- Checks that the row has existing columns 23 and 24. Extracts data from columns “House Number” – column 23 and “Street Name” – column 24
- Checks that the values are not empty.
- Prints the house number and street name and count of 1 for each occurrence as standard output.
  Key: house number, street name Value: 1
  i.e., **123,Third St.: 1**

Reducing

The reducer code is uploaded and stored in:

/mapreduce-test/mapreduce-test-python/Part1Q3/reducer.py

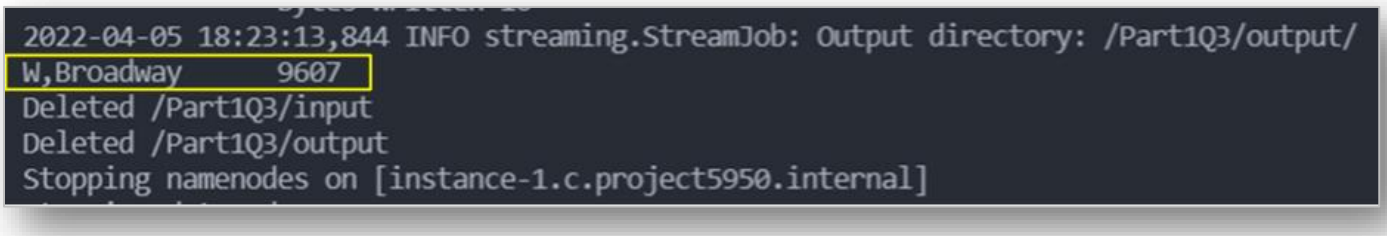
The reducer receives the streaming results of the mapper via standard input stream, and performs the following:

- Strips whitespace
- Splits the passed-on key and value on a tab into two variables **street_location** and **count**.
- Creates a dictionary where **house number** and **street name** is a key and sum of count of occurrences is value. Once summation is completed the dictionary is sorted in descending order by sum of count and the most frequent **house number** and **street name** is printed along with the corresponding count. The result is printed as standard output.

Results Part c

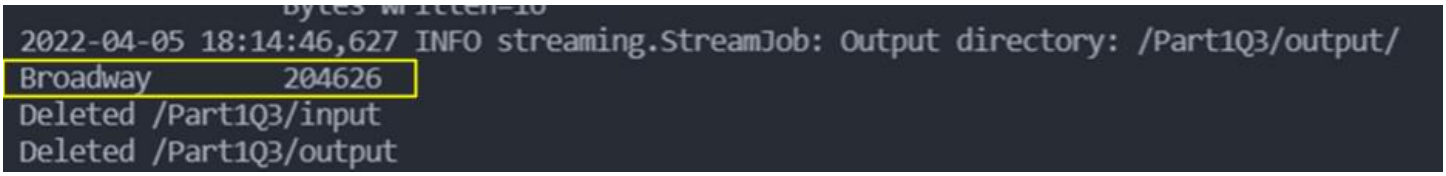
The results of the MapReduce process are:

- Most common house number: **W**
- Most common street name: **Broadway**
- Tickets issued: **9607**



If performing a MapReduce using only the street name column the results are:

- Most common street name: **Broadway**
- Tickets issued: **204626**



Solution d

d) **Which color of the vehicle is most likely to get a ticket?**

To answer the question, we will determine the highest aggregate number of violations for given vehicle color.

The column required is:

- **"Vehicle Color"**

The color data has 1656 different values with includes colors recoded in different ways, noise, and missing values. The data is cleaned in the mapper.py. The missing values and noise were removed from the dataset.

Task Scheduling Control

Submission of the MapReduce code is controlled by a bash script run file named **test.sh**. This script coordinates the submission of the map and reduce jobs along with the source data file to the Hadoop Streamer and HDFS. The run file used for this solution is uploaded and stored on the name node in:

/mapreduce-test/mapreduce-test-python/Part1Q4/test.sh

Mapping

The mapper code is uploaded and stored in:

/mapreduce-test/mapreduce-test-python/Part1Q4/mapper.py

mapper.py reads csv data file as standard input stream, line-by-line, and performs the following steps for each line:

- Strips whitespace
- Checks that the row has existing column 33 – "**Vehicle Color**" and if so, extracts the data.
- Checks that the values of column "**Vehicle Color**" - 33 are not empty and that the values are alphabet letters.
- Prints the vehicle color and count of 1 for each occurrence as standard output.
  Key: vehicle color. Value: 1.
  i.e., **Blu: 1**

Shuffling & Sorting

Upon completion of the mapper process, Hadoop initiates an intermediate process to receive the mapper output, sort generate a key-value list and sort the list. Data is then handed off to the reducer.

Reducing

The reducer code is uploaded and stored in:

/mapreduce-test/mapreduce-test-python/Part1Q4/reducer.py

The reducer receives the streaming results of the mapper via standard input stream, and performs the following:

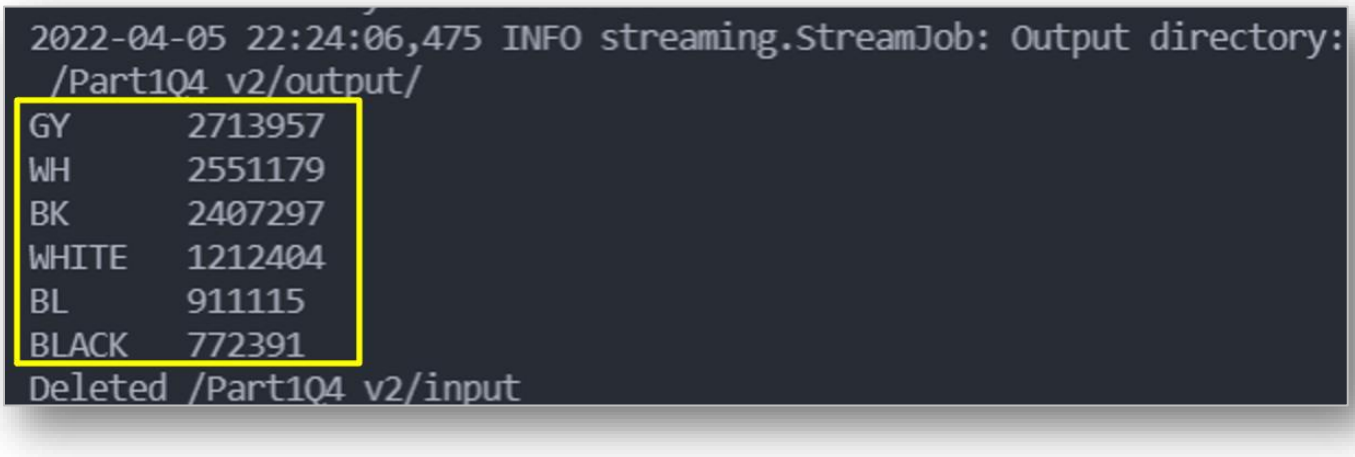
- Strips whitespace
- Splits the passed-on key and value on a tab into two variables color and count.
- Creates a dictionary where color is a key and sum of count of occurrences is value. Once summation is completed the dictionary is sorted in descending order by sum of count and the most frequent vehicle color is printed along with the corresponding count. The result is printed as standard output.

Results Part d

The results of the MapReduce process are:

- Most common vehicle color: **GY**
- Tickets issued: **2713957**

At first glance it appears that grey



At first glance it appears that grey “GY” is the most ticketed car color. By examining top data closer we can see that black has 3179688 tickets and white has 3763583 tickets. To obtain exact numbers the enormous amount of color codes listed in the source data must be normalized and then aggregated. Fully cleaning the data is not within the scope of this project.