Queen Mary
University of London

BIG DATA PROCESSING

ECS765P

COURSEWORK

ETHEREUM ANALYSIS

SUBMITTED BY:
Sivasankari Rajamanickam
Student ID: 210965243
MSc Big Data Science with ML Systems

Table of Contents

PART A – TIME ANALYSIS

Question:

Create a bar plot showing the number of transactions occurring every month between the start and end of the dataset. Create a bar plot showing the average value of transactions in each month between the start and end of the dataset.

Job 1:

Problem Statement : To find the number of transactions that occur every month between the start and end of the dataset

JOB ID :

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_5157/

Code file : partA1.py

Code file for plotting : partAplot.ipynb

Type of program : Hadoop MapReduce

Command : python partA1.py -r hadoop --output-dir PartA1_out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions

Explanation :

Input : data/ethereum/transactions

Output : part-00000.txt, part-00001.txt

Mapper : Extracts the 7th column from transactions, which is *block_timestamp.* This is in epoch format. *time.strftime()* is used to get the month and year from the epoch time.

Key: *month, year*

Value: *1*

Hence, for each transaction, its month-year and a count 1 is yielded

Reducer : Receives all items with same key(month-year) and sums the count, to get the number of transactions having the same month-year. This yields the following key and value, which gets printed on the output file.

Key: month, year

Value: count of transactions happening in that month-year

Sample Output :

["01", "17"]   1409664

["01", "19"]   16569597

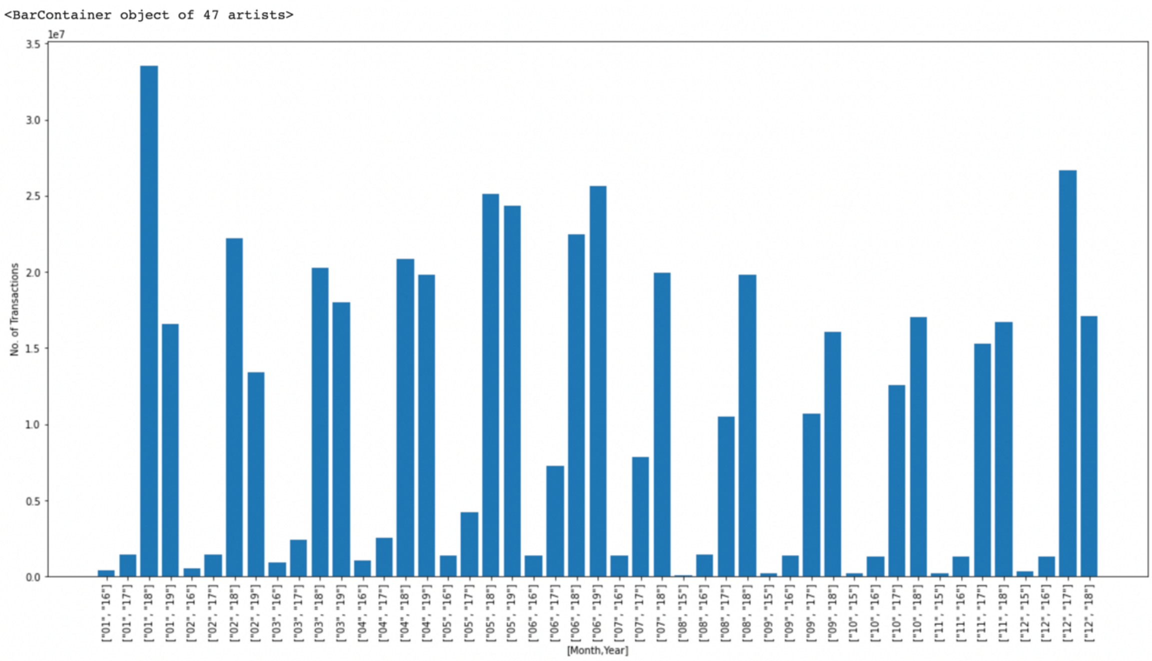
["02", "16"]   520040

["02", "18"]   22231978

["03", "17"]   2426471

Plotting : matplotlib Python library is used to visualise the output.

This program combines the two output files, reads it as pandas dataframe and creates a bar plot with X axis as month-year and Y axis as number of transactions



Observation : Year 2018 seem to have the greatest number of transactions followed by 2019. Year 2015 has the least number of transactions throughout. The highest number of transactions is seen in January 2018.

Job 2:

Problem Statement : Find the average of the value for transactions occurring in each month of different years of the given dataset and make a bar plot for the same.

JOB ID :

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_5305/

Code file : partA2.py

Code file for plotting : partAplot.ipynb

Type of program : Hadoop MapReduce

Command : python partA2.py -r hadoop --output-dir PartA2_out --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions

Explanation :

Input : data/ethereum/transactions

Output : part-00000.txt, part-00001.txt

Mapper : Extracts the 7th column from transactions, which is *block_timestamp.* This is in epoch format. *time.strftime()* is used to get the month and year from the epoch time. Column 3 from transactions which is *value* is extracted.

Key: month, year

Value: value, 1

Hence, for each transaction, its month-year, value and a count 1 is yielded

Combiner : Receives all items with same key(month-year) and sums the value of all transactions with the same month-year as well as the count of number of transactions of the same. This yields the following intermediate key-value pair, which gets printed on the output file.

Key: *month, year*

Value: *total-value, total-count*

Reducer : Receives all items with same key(month-year) and sums the value of all transactions with the same month-year as well as the count of number of transactions of the same. The average value is found dividing the total value of transactions happened in a month-year by the total number of transactions happened in the same. This yields the following key and value, which gets printed on the output file.

Key: *month, year*

Value: *average value of transaction*

Sample Output :

["01", "17"] 5.620285956535426e+19

["01", "19"] 4.4548138889025393e+18

["02", "16"] 6.5547608759905436e+19

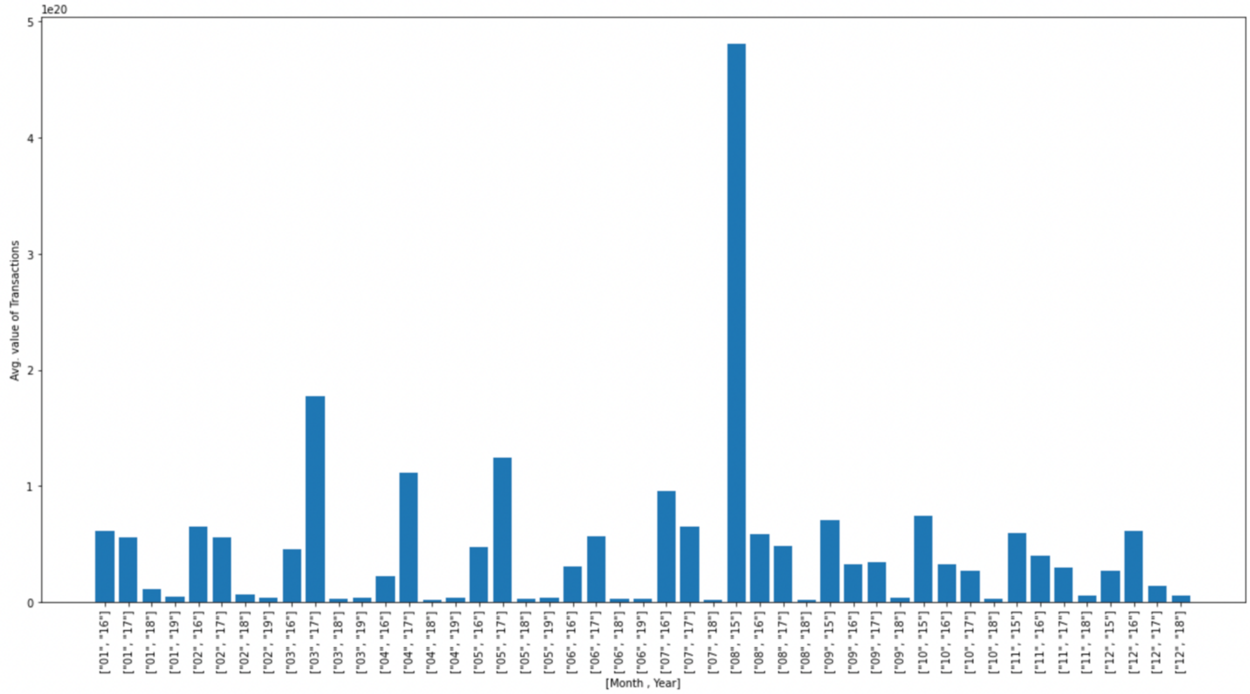
["02", "18"] 6.23036279509048e+18

["03", "17"] 1.77021580942219e+20

Plotting : matplotlib Python library is used to visualise the output.

This program combines the two output files, reads it as pandas dataframe and creates a bar plot with X axis as month-year and Y axis as average value of transactions.

<BarContainer object of 47 artists>



Observation : The average value of transaction seems to decrease over years. The least value of transaction occurs in 2019. August 2015 has the highest value of transaction, which seem to surpass all the other averages.

PART B – TOP TEN MOST POPULAR SERVICES

Question:

Evaluate the top 10 smart contracts by total Ether received.

Problem Statement :

1) Aggregate **transactions** to see how much each address within the user space has been involved in.
2) Perform a **repartition join** between this aggregate and **contracts** and filter top 10 results.
3) In the reducer, if the address for a given aggregate from Job 1 was not present within **contracts** this should be filtered out as it is a user address and not a smart contract.

JOB ID :

Job 1 :

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_6961/

Job 2 :

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_7009/

Code file : partB.py

Type of program : Hadoop MapReduce

Command : python partB.py -r hadoop --output-dir PartB_Out hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions -r hadoop hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/contracts

Explanation:

Input : data/ethereum/transactions, data/ethereum/contracts

Output : part-00000.txt

Mapper1 :

For Transactions Table:

For any line from Transactions table, its corresponding *to_address* and *value* is extracted. *flag_value* is set to 'T'

Key: *to_address*

Value: *flag_value, value*

For Contracts Table:

Its corresponding *address* is extracted. *flag_value* is set to ‘C’

Key: *address*

Value: *flag_value, 1*

Reducer1 : Here is where the repartition join is performed. The primary key for the join is that address should exist in both transaction and contracts table. Hence, all items with same address are received and the *flag_value* is used for comparison. *flag_value* ‘C’ indicates that the address is present in contracts table and a *flag_value* ‘T’ indicates that the address is present in transaction table. Hence, if both the values match, then the *transactionValue* extracted from the transaction table for that address is appended to a list and all values belonging to same address are summed up and yielded.

Key: *address*

Value: *sum(transactionValues)*

Mapper2 : Receives the *address* and the sum of *values* for that address and yields both as values for sorting.

Key: *None*

Value: *address, values*

Reducer2 : Receives all *address* and *value* , sorts the address-values in descending order, and gets the top ten address of contracts with higher value of transaction.

MRStep : A single MapReduce program is created to run both the jobs and MRStep is used to schedule the order for mappers and reducers to execute.

Sample Output:

"0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444"	84155100809965865822726776
"0xfa52274dd61e1643d2205169732f29114bc240b3"	45787484483189352986478805
"0x7727e5113d1d161373623e5f49fd568b4f543a9e"	45620624001350712557268573
"0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef"	43170356092262468919298969

"0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8"   27068921582019542499882877

PART C – TOP TEN MOST ACTIVE MINERS

Question:

Evaluate the top 10 miners by the size of the blocks mined. This is simpler as it does not require a join. You will first have to aggregate **blocks** to see how much each miner has been involved in. You will want to aggregate **size** for addresses in the **miner** field.

Problem Statement :

1) Get the size of the block that each miner has mined.
2) Calculate the total size of block mined for each miner
3) Find the top miners with the highest block size mined.

JOB ID :

Job 1 :

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_7050/

Job 2 :

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_7051/

Code file : partC.py

Type of program : Hadoop MapReduce

Command : python partC.py -r hadoop --output-dir PartC_Out
hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/blocks

Explanation:

Input : data/ethereum/blocks

Output : part-00000.txt

Mapper1 : Takes the blocks table as the input, extracts *miner* and integer value of *size* of block mined by each miner and yields the same

Key: *miner*

Value: *size*

Reducer1 : Receives all the items with the same *miner* ID and sums the *size* of block mined for each unique *miner* ID.

Key: *miner*

Value: *sum(size)*

Mapper2 : Receives the *miner* ID and the sum of *size* of block mined for that minerID and yields both as values for sorting.

Key: *None*

Value: *miner, totalSize*

Reducer2 : Receives all *miner* and *size* , sorts the miner-size in descending order, and gets the top ten IDs of miner with higher value of block size mined.

MRStep : A single MapReduce program is created to run both the jobs and MRStep is used to schedule the order for mappers and reducers to execute.

Sample Output:

"0xea674fdde714fd979de3edf0f56aa9716b898ec8" 23989401188

"0x829bd824b016326a401d083b33d092293333a830" 15010222714

"0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c" 13978859941

"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5"	10998145387
"0xb2930b35844a230f00e51431acae96fe543a0347"	7842595276

PART D – DATA EXPLORATION

SCAM ANALYSIS

1) Popular Scams

Question:

Utilising the provided scam dataset, what is the most lucrative form of scam? Does this correlate with certainly known scams going offline/inactive? For the correlation, you could produce the count of how many scams for each category are active/inactive/offline/online/etc and try to correlate it with volume (value) to make conclusions on whether state plays a factor in making some scams more lucrative. Therefore, getting the volume and state of each scam, you can make a conclusion whether the most lucrative ones are ones that are online or offline or active or inactive. So, for that purpose, you need to just produce a table with SCAM TYPE, STATE, VOLUME which would be enough

Part 1:

Problem Statement : Produce the volume for each category of scam

JOB ID :

Job 1:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_7729/

Job 2:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_7807/

Code file : popularScams1.py

Type of program : Hadoop MapReduce

Command : python popularScams1.py -r hadoop

hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions -r hadoop

hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/scams.json > popularScams1.txt

Explanation :

Input : data/ethereum/transactions, data/ethereum/scams.json

Output : popularScams1.txt

Mapper 1 :

For Transactions Table:

For any line from Transactions table, its corresponding *to_address* and *value* is extracted. *flag_value* is set to 'T'

Key: *to_address*

Value: *value,flag_value*

For Scams file:

category and *address* of each scam is extracted and the *flag_value* is set to 'S'

Key: *address*

Value: *category, flag_value*

Reducer 1 : Repartition join is performed here. Values with same *address* are received. If the *flag_value* is 'T', the *value* is totalled. Else if the *flag_value* is 'S', the *category* is extracted. If the *address* is present in both transactions and scams, the *value* and *category* is yielded.

Key: *category*

Value: *totalValue*

Mapper 2 : Gets all the *values* with the same *category* and yields the same.

Key: *category*

Value: *value*

Reducer 2 : Receives the *values* with the same *category* and sums the *value* up.

Key: *category*

Value: *sum(value)*

MRStep : A single MapReduce program is created to run both the jobs and MRStep is used to schedule the order for mappers and reducers to execute.

Sample Output :

"Scamming" 3.833616286244436e+22

"Fake ICO" 1.35645756688963e+21

"Phishing" 2.699937579408742e+22

"Scam" 0

Part 2:

Problem Statement : Produce the state of each category of scam.

JOB ID :

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_8291/

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_7009/

Code file : popularScams2.py

Type of program : Hadoop MapReduce

Command : python popularScams2.py -r hadoop hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions -r hadoop hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/scams.json > popularScams2.txt

Explanation :

Input : data/ethereum/transactions, data/ethereum/scams.json

Output : popularScams2.txt

Mapper 1 :

For Transactions Table:

For any line from Transactions table, its corresponding *to_address* is extracted.

flag_value is set to 'T'

Key: *to_address*

Value: *flag_value,* 0

For Scams file:

category , *status* and *address* of each scam is extracted and the *flag_value* is set to 'S'

Key: *address*

Value: *flag_value, category, status*

Reducer 1 : Repartition join is performed here. Values with same *address* are received. If the *flag_value* is 'T', the *value* is totalled. Else if the *flag_value* is 'S', the *category* and *status* is extracted. If the *address* is present in both transactions and scams, the *value* and *category* is yielded.

Key: *category*

Value: *totalValue*

Mapper 2 : Gets all the *values* with the same *category* and yields the same.

Key: *status, category*

Value: *value*

Reducer 2 : Receives the *values* with the same *status* and *category* and sums the *value* up.

Key: *status,category*

Value: *sum(value)*

MRStep : A single MapReduce program is created to run both the jobs and MRStep is used to schedule the order for mappers and reducers to execute.

Sample Output :

["Active", "Scamming"]	88444
["Inactive", "Phishing"]	22
["Offline", "Fake ICO"]	121
["Offline", "Phishing"]	7022

["Offline", "Scam"] 0

["Suspended", "Phishing"] 11

["Active", "Phishing"] 1584

["Offline", "Scamming"] 24692

["Suspended", "Scamming"] 56

Conclusion:

From the output, it is clearly seen that **Scamming** is the most lucrative form of scam with the highest value of **active** scams. All the categories of scam seem to be having a part in **offline** scam. **Active** and **Offline** scams contribute the most to scamming.

MISCELLANEOUS ANALYSIS

1) Fork the Chain

Question:

There have been several forks of Ethereum in the past. Identify one or more of these and see what effect it had on price and general usage. For example, did a price surge/plummet occur, and who profited most from this?

Problem Statement : Identify a fork and get the count and price involved in the fork.

JOB ID :

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_0721/

Code file : fork.py

Code file for plotting : forkPlot.ipynb

Type of program : Hadoop MapReduce

Command : python fork.py -r hadoop --output-dir forkOut --no-cat-output hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions

Explanation :

Input : data/ethereum/transactions

Output : part-00000.txt, part-00001.txt

Fork used:

2020

Beacon Chain genesis

Dec-01-2020 12:00:35 PM +UTC

Beacon Chain block number: 1

ETH price: $586.23 USD

ethereum.org on waybackmachine

Mapper : Take the transaction folder as input. Extract gas price and block timestamp. Convert the epoch time to human readable time and extract the month and year. Check if the month is 12 and Year is 2020 and yield the date, count and gas_price

Key: *block_timestamp.tm_mday*

Value: 1*, gas_price*

Combiner :

Get all the prices with the same date, sum the total count and price

Key: *block_timestamp*

Value: (*totalCount,totalPrice)*

Reducer : Get all the prices with the same date, sum the total count and price

Key: *block_timestamp*

Value: (*totalCount,totalPrice)*

Sample Output :

29 [40011, 25000000000.0]

3 [47672, 35484297556.0]

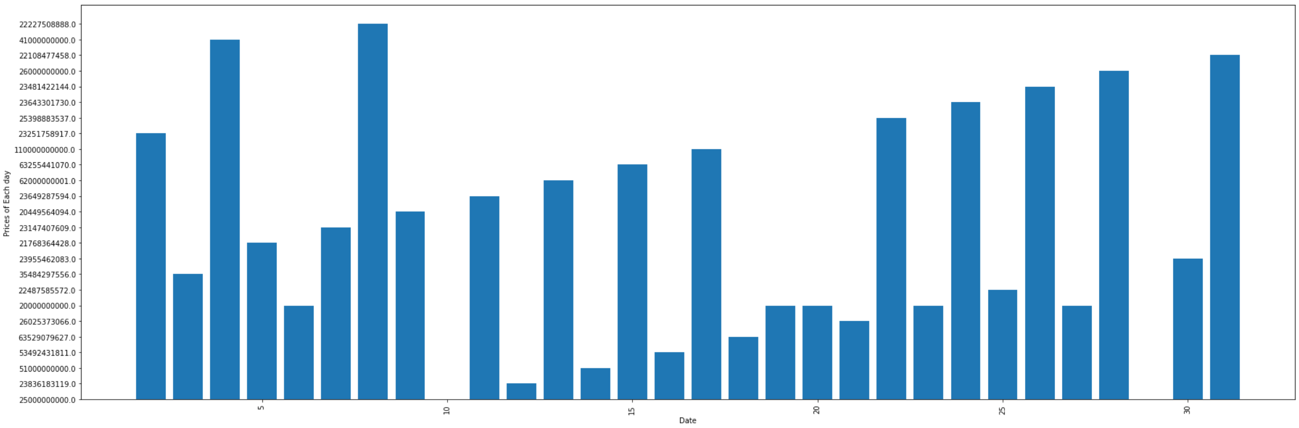
30 [39435, 23955462083.0]

5 [41719, 21768364428.0]

7 [43708, 23147407609.0]

9 [43940, 20449564094.0]

Plotting : matplotlib Python library is used to visualise the output.



Observation : The price of gas lowers dramatically on forking days, and the number of transactions follows a similar pattern. Both of these drops, however, were brief and quickly reversed.

2) Gas Guzzlers

Question:

For any transaction on Ethereum a user must supply gas. How has gas price changed over time? Have contracts become more complicated, requiring more gas, or less so? Also, could you correlate the complexity for some of the top-10 contracts found in Part-B by observing the change over their transactions

Problem Statement : Identify the change in gas price over time

JOB ID :

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_0825/

Code file : gasGuzzlers.py

Code file for plotting : gasGuzzlersPlot.ipynb

Type of program : Spark

Command : spark-submit gasGuzzlers.py

Explanation :

Input : data/ethereum/transactions, data/ethereum/contracts, data/ethereum/blocks

Output : AverageGas.txt, TimeDifference.txt

Code:

Gets the input. Checks if each line is from transactions, contracts or blocks.

If the line comes from transactions, it extracts the block_timestamp and gas_price, reduces the items based on same month-year and sums up the gas price and the total count of items with the same month-year. Calculates the average of gas price by dividing total gas price by total count and yields it to the output text file **AverageGas.txt**

If the line is from contracts, it extracts the address of the contracts.

If the line is from blocks, it extracts the block_number, difficulty, gas_used and timestamp. Finds all the addresses available in contracts, and yields the month-year, difficulty and gas used for the same.

Sample Output :

AverageGas:

('15.08', 159744029578.0333)

('15.09', 56511301521.03311)

('15.10', 53901692120.53661)

('15.11', 53607614201.79755)

('15.12', 55899526672.35286)

TimeDifference:

('15.08', (4030960805570.0, 360218))

('15.09', (6577868584193.0, 540131))

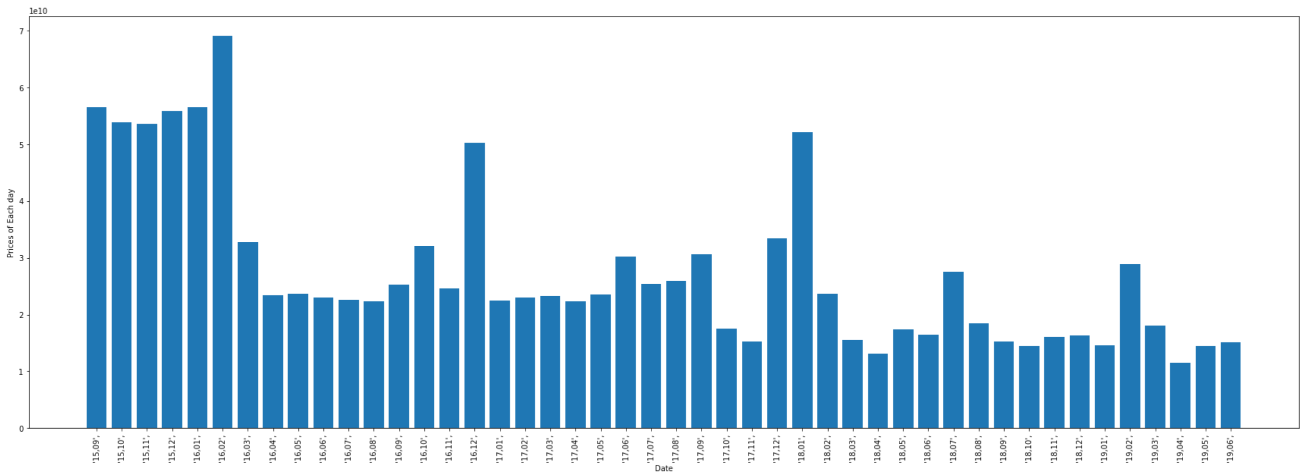
('15.10', (6352827787297.0, 641355))

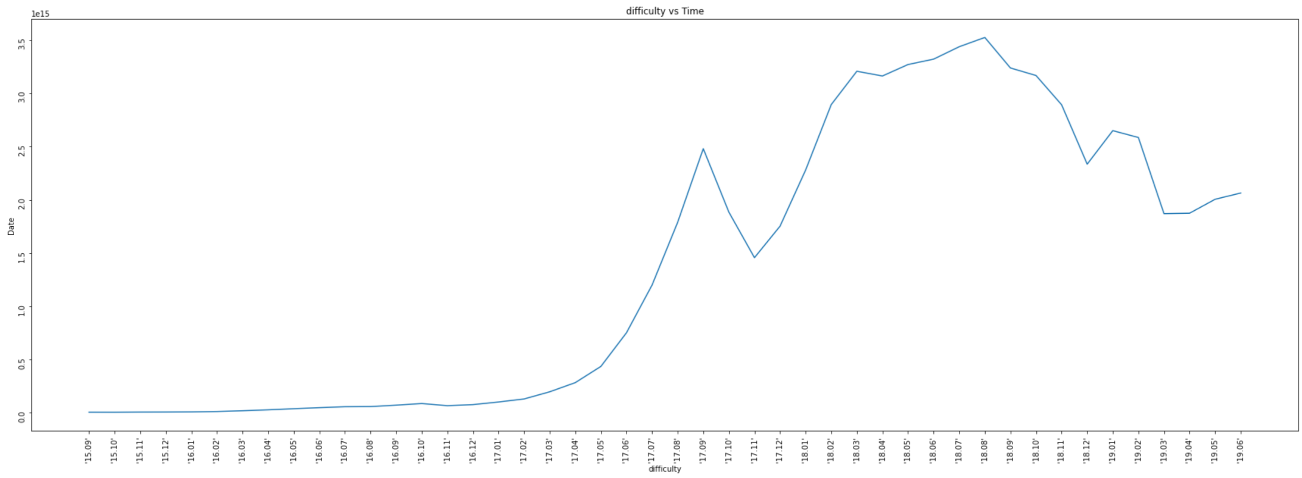
('15.11', (7772752046929.0, 609518))

('15.12', (8279271173937.0, 696716))

('16.01', (9509622515878.0, 714548))

Plotting : matplotlib Python library is used to visualise the output.





Observation : The average price of gas has decreased over years and in each year, prices seem to spike during the beginning and end of the year.

The amount of gas used increases over the years and maintains a constant state after 2015.

3) Comparative Evaluation

Question:

Reimplement Part B in Spark (if your original was MRJob, or vice versa). How does it run in comparison? Keep in mind that to get representative results you will have to run the job multiple times, and report median/average results. Can you explain the reason for these results? What framework seems more appropriate for this task?

Solution:

Hadoop Map/Reduce:

The MapReduce code to evaluate top ten smart contracts is stored in comparativeEvaluationHadoop.py. This code is run thrice, and the time of each run is noted down and finally the average time is calculated.

Command: python comparativeEvaluationHadoop.py -r hadoop --output-dir PartD_Hadoop_Compare_3 hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/transactions -r hadoop hdfs://andromeda.eecs.qmul.ac.uk/data/ethereum/contracts

Run 1:

Job 1:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_1715/

Start Time: Thu Apr 14 17:46:51 +0100 2022

Launch Time: Thu Apr 14 17:46:52 +0100 2022

Finish Time: Thu Apr 14 18:35:28 +0100 2022

Job 2:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_1854/

Start Time: Thu Apr 14 18:35:32 +0100 2022

Launch Time: Thu Apr 14 18:35:32 +0100 2022

Finish Time: Thu Apr 14 18:40:45 +0100 2022

Total Time: 53minutes

Run2:

Job1:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_1751/

Start Time: Thu Apr 14 17:56:31 +0100 2022

Launch Time: Thu Apr 14 17:56:32 +0100 2022

Finish Time: Thu Apr 14 18:42:35 +0100 2022

Job 2:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_1878/

Start Time: Thu Apr 14 18:42:39 +0100 2022

Launch Time: Thu Apr 14 18:42:42 +0100 2022

Finish Time: Thu Apr 14 18:45:49 +0100 2022

Total Time: 49minutes

Run 3:

Job1:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_1808/

Start Time: Thu Apr 14 18:15:43 +0100 2022

Launch Time: Thu Apr 14 18:15:43 +0100 2022

Finish Time: Thu Apr 14 18:51:41 +0100 2022

Job 2:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_1904/

Start Time: Thu Apr 14 18:51:45 +0100 2022

Launch Time: Thu Apr 14 18:51:46 +0100 2022

Finish Time: Thu Apr 14 18:55:18 +0100 2022

Total Time: 39minutes

Average Time: 47 minutes

Spark:

The same logic is replicated in Spark to find the top smart contracts and the code is run thrice to calculated the average time.

Command:

spark-submit comparativeEvaluationSpark.py

Run 1 :

http://andromeda.student.eecs.qmul.ac.uk:8088/cluster/app/application_1649894236110_1516

Start Time:Thu Apr 14 16:58:48 +0100 2022

Launch Time:Thu Apr 14 16:58:48 +0100 2022

Finish Time: Thu Apr 14 17:04:03 +0100 2022

Total Time: 5minutes

Run 2:

http://andromeda.student.eecs.qmul.ac.uk:8088/cluster/app/application_1649894236110_1559

Start Time: Thu Apr 14 17:09:50 +0100 2022

Launch Time: Thu Apr 14 17:09:50 +0100 2022

Finish Time: Thu Apr 14 17:15:58 +0100 2022

Total Time: 6 minutes

Run 3:

http://andromeda.student.eecs.qmul.ac.uk:8088/cluster/app/application_1649894236110_1595

Start Time: Thu Apr 14 17:19:14 +0100 2022

Launch Time: Thu Apr 14 17:19:14 +0100 2022

Finish Time: Thu Apr 14 17:25:46 +0100 2022

Total time: 6 minutes 32 seconds

Average Time: 5 minutes 40 seconds

Evaluation:

In comparison, Spark (Average Time: 5 min 40 sec) seems to execute the code at an extremely faster pace when compared to Hadoop MapReduce (Average Time: 47 min). MapReduce processes data on the disc, whereas Spark processes and keeps data in memory (Resilient Distributed Dataset (RDD)) for further processing.