ANALYSIS OF ETHEREUM TRANSACTIONS AND SMART CONTRACTS

Shyam Babu Valmiki
Department of electronics and communications
Queen Mary University of Londonline

London, United Kingdom
s.b.valmiki@se22.qmul.ac.uk

***Abstract*— Here we analyze the Ethereum data collected from August 2015 till January 2019. We perform some basic operations to apply the knowledge gained to get some insights from the data.**

I. INTRODUCTION

We have datasets that are available for processing. The files available are blocks, transactions, contracts, and the scams dataset.

We perform some basic operations such as aggregation, join and other operations to answer the tasks. For our convenience we have small subsets of the original data available for us. We first test the code on these small datasets and then test the code on the larger dataset. The path for the datasets is /DATA-REPOSITORY-BKT/ECS765/ETHEREUM-PARVULUS.

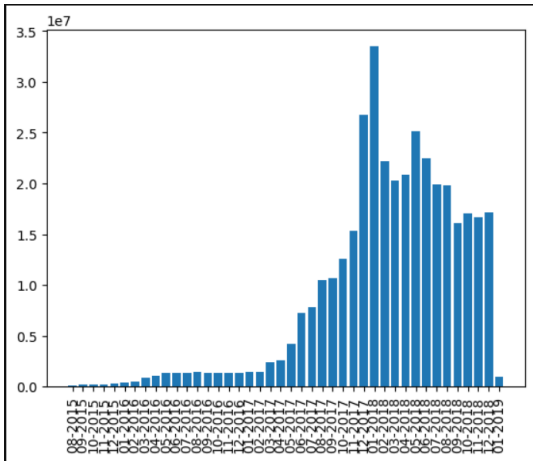

Figure – 1. Plot showing the bar graph time vs number of transactions in a month.

II. SOLUTIONS

A. *Task - A*

Plotting graph showing the number of transactions in every month.

For this task we first read the csv file and form a rdd consisting of month and year as the key, 1 as the value. Finally, we use the reduceByKey() to group and count the number of transactions using the key to group the values.

The code for the task is as follows.

The code below reads the csv file and stores the lines.

```
lines = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket + "/ECS765/ethereum-parvulus/transactions.csv")

clean_lines = lines.filter(good_line)
```

The code below extracts the month and year from the time stamp.

```
dates = clean_lines.map(lambda b: (datetime.fromtimestamp(int(b.split(',')[11])).strftime("%m-%Y"), 1))


# Use reducebykey get the counts per each month of the year

dates = dates.reduceByKey(operator.add)
```

Plotting graph showing the number of transactions in every month.

We start by reading the csv files and storing the lines. We then filter the malformed lines. We then start extracting the features the month and year from the time stamp and the value from each line.

Again, we continue to group the rdds using the key i.e. month and year of the transaction. We then divide the sum of values by the length of values to get the average value of the transactions. We then print and write the average values to a file.

The code for the above is as following.

```
#reading the lines from csv

lines = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket + "/ECS765/ethereum-parvulus/transactions.csv")

clean_lines = lines.filter(good_line)


# Extracting the month date from the timestamp

dates = clean_lines.map(lambda b: (datetime.fromtimestamp(int(b.split(',')[11])).strftime("%m-%Y"), (int(b.split(',')[7]))))


grouped_rdd = dates.groupByKey()

averages_rdd = grouped_rdd.mapValues(lambda values: sum(values) / len(values))
```

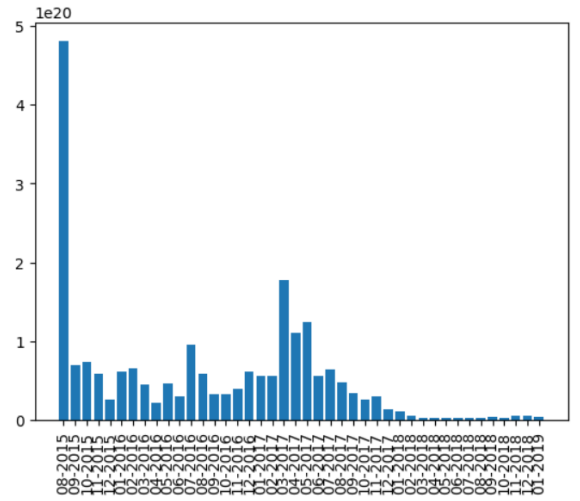

Figure – 2. Plot showing the bar graph time vs average value of transactions in a month.

B. Task – B

Top 10 smart contracts by total Ether received.

For this task we must read two csv files namely the transactions and the contracts file. We define two functions namely **good_lines_1** and **good_lines_2** to filter the malformed lines and obtain the required data type from the csv files. We store the address field from the contracts csv file and to_address and the value from the transactions csv onto rdd's.
We then join the two rdd's when **address** field and **to_address** field match and save it as tran_cont rdd. We sort this rdd by the **value** field. We then continue to store the top 10 rdd's into a variable then print and write to it a file.
The code for the above task is as follows.

```
    # Reading the csv and storing the values
    # Reading transactions csv
    lines_1 = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket + "/ECS765/ethereum-parvulus/transactions.csv")
    clean_lines_1=lines_1.filter(good_line_1)
    transactions_features=clean_lines_1.map(lambda l: (l.split(',')[6],(l.split(',')[7])))
    print(transactions_features.take(1))
    sum_by_key = transactions_features.reduceByKey(lambda a, b: float(a) + float(b))
    print(sum_by_key.take(10))

# Reading the contracts csv
    lines_2 = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket + "/ECS765/ethereum-parvulus/contracts.csv")
    clean_lines_2=lines_2.filter(good_line_2)
    contracts_features=clean_lines_2.map(lambda l: (l.split(',')[0],1))
    print(contracts_features.take(1))

    # Join on the address and to_address from the contracts and transactions values respectively
    tran_cont=contracts_features.join(sum_by_key)
    print(tran_cont.take(20))

    # Extracting the second element of the second element of the tuple
    second_elements = tran_cont.map(lambda x: float(x[1][1]))

    # Sorting the RDD by the second element of the tuple in descending order
    sorted_rdd = tran_cont.sortBy(lambda x: -float(x[1][1]))

    # Getting the top 10 tuples in the sorted RDD
    top_10_rdds = sorted_rdd.take(10)
```

The top 10 contracts are as following.
[["0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444", [1, 8.415536369994149e+25]],
["0x7727e5113d1d161373623e5f49fd568b4f543a9e", [1, 4.56271285129153e+25]],
["0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef", [1, 4.2552989136413066e+25]],
["0xbfc39b6f805a9e40e77291aff27aee3c96915bdd", [1, 2.110419513809366e+25]],
["0xe94b04a0fed112f3664e45adb2b8915693dd5ff3", [1, 1.554307763526369e+25]],
["0xabbb6bebfa05aa13e908eaa492bd7a8343760477", [1, 1.0719485945628904e+25]],
["0x341e790174e3a4d35b65fdc067b6b5634a61caea", [1, 8.379000751917755e+24]],
["0x58ae42a38d6b33a1e31492b60465fa80da595755", [1, 2.902709187105735e+24]],
["0xc7c7f6660102e9a1fee1390df5c76ea5a5572ed3", [1, 1.238086114520043e+24]],
["0xe28e72fcf78647adce1f1252f240bbfaebd63bcc", [1, 1.1724264325158214e+24]]]

C. Task – C

Top 10 Most active miners.

We use the same approach here. We first read csv files and then remove malformed lines. We then store the miner and size in rdd.
We proceed by grouping the rdds by the miner, then find the sum of size that was mined by the respective miner.
We then print and store the values onto a file.
The code is as following.

```
    # Read the CSV files
    lines = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket + "/ECS765/ethereum-parvulus/blocks.csv")
    clean_lines = lines.filter(good_line)

    # Storing miner and the size
    size_1 = clean_lines.map(lambda b: (b.split(',')[9], float(b.split(',')[12])))

    # Grouping and finding the sum
    size_1 = size_1.reduceByKey(operator.add)
```

```
print(size_1.take(5))

# sort the values
top10=size_1.takeOrdered(10, key=lambda x: -x[1])

# Printing and storing values to a file
for c,v in top10:
    print("{}: {}".format(c,v))
```

The top 10 miners are as following.

```
[["0xea674fdde714fd979de3edf0f56aa9716b898ec8", 17453393724.0],
["0x829bd824b016326a401d083b33d092293333a830", 12310472526.0],
["0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c", 8825710065.0],
["0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5", 8451574409.0],
["0xb2930b35844a230f00e51431acae96fe543a0347", 6614130661.0],
["0x2a65aca4d5fc5b5c859090a6c34d164135398226", 3173096011.0],
["0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb", 1152847020.0],
["0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01", 1134151226.0],
["0x1e9939daaad6924ad004c2560e90804164900341", 1080436358.0],
["0x61c808d82a3ac53231750dadc13c777b59310bd9", 692942577.0]]
```

D. Task - D

GAS GUZZLERS

Plotting graph showing the average gas price per transaction in a month.

We extract the month and year from the time stamp and the price reading the csv file transactions and storing in rdd. We proceed by aggregation; we find the total price and the number of transactions in a month. Then divide the sum by the count of transactions to get the average price per month.

The code is as follows.

```
#finding the avg gas_price per month.
gas_price = clean_lines.map(lambda b: (datetime.fromtimestamp(int(b.split(',')[11])).strftime("%m-%Y"),float(b.split(',')[9])))

sum_count = gas_price.mapValues(lambda x: (x, 1)).reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))

# compute the average by dividing the sum by the count
average_gas_price = sum_count.mapValues(lambda x: x[0] / x[1])
print(average_gas_price.take(5))
```

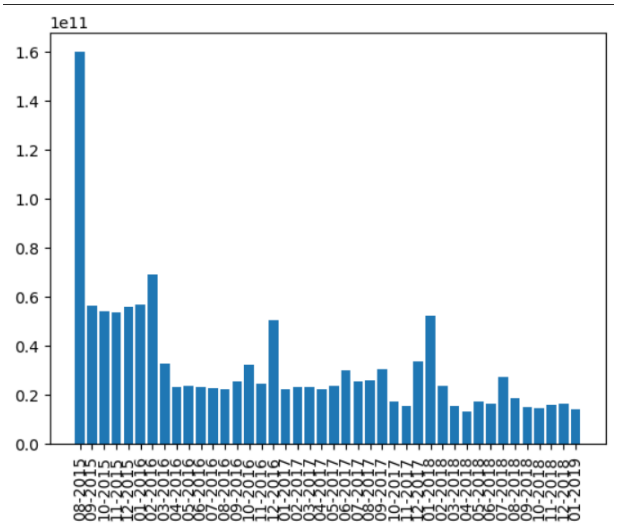

Figure – 3. Plot showing the bar graph time vs average gas price per transaction in a month.

Plotting graph showing the average gas used per transaction in a month.

We extract the month and year from the time stamp and the price reading the csv file transactions and storing in rdd. We proceed by aggregation; we find the total gas and the number of transactions in a month. Then divide the sum by the count of transactions to get the average gas used per month.

The code is as follows.

```
gas_used = clean_lines.map(lambda b: (datetime.fromtimestamp(int(b.split(',')[11])).strftime("%m-%Y"),float(b.split(',')[8])))

sum_count = gas_used.mapValues(lambda x: (x, 1)).reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))

# compute the average by dividing the sum by the count
average_gas_used = sum_count.mapValues(lambda x: x[0] / x[1])
```

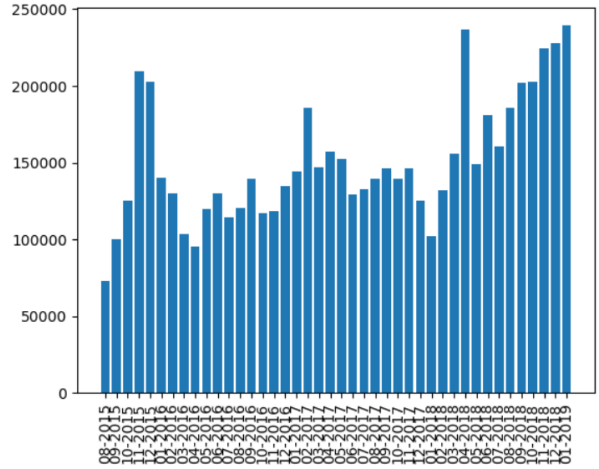

Figure – 4. Plot showing the bar graph time vs average gas used per transaction in a month.

Comparing if the popular contracts use the same amount of average gas.

a. *Finding the gas used by popular contracts.*

We first continue to read the lines from csv as usual, We filter lines to remove the malformed lines and store the to_address and the gas values in the rdd.

Then from the result of task B, let us use the top 10 contracts and store in a list. We proceed by creating a set for faster lookup and filter the first rdd to extract the id's. Then we proceed by grouping the values by the key. We then calculate the average per group.

The code is as following.

```
# converting top_10_contracts to a set for faster lookup
top_10_contracts_set = set([item for sublist in top_10_contracts for item in sublist])

# filter transactions_features to only include ids in top_10_contracts_set
filtered_transactions = transactions_features.filter(lambda x: x[0] in top_10_contracts_set)

# grouping values by id
grouped_transactions = filtered_transactions.groupByKey()

# calculating average value per group
average_transactions = grouped_transactions.mapValues(lambda values: sum(map(float, values))/len(values))
```

The top 10 contracts and the average gas used per transaction are as follows.

```
[["0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444", 83153.10930123563],
["0xabbb6bebfa05aa13e908eaa492bd7a8343760477", 101932.03479546004],
["0xe28e72fcf78647adce1f1252f240bbfaebd63bcc", 150945.27918781727],
["0x341e790174e3a4d35b65fdc067b6b5634a61caea", 198832.66666666666],
["0xbfc39b6f805a9e40e77291aff27aee3c96915bdd", 39999.61330880607],
["0x7727e5113d1d161373623e5f49fd568b4f543a9e", 89826.72419423984],
["0xe94b04a0fed112f3664e45adb2b8915693dd5ff3", 136704.900966096],
["0x58ae42a38d6b33a1e31492b60465fa80da595755", 50212.80846597866],
["0xc7c7f6660102e9a1fee1390df5c76ea5a5572ed3", 33973.648482614575],
["0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef", 42553.82170235774]]
```

b. Finding the average gas used per transaction.

This is simple aggregation of data. Find the sum of gas used by all the transactions and then divide by the count of the transactions.

The code is as follows.

```
lines_1 = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket + "/ECS765/ethereum-parvulus/transactions.csv")
clean_lines_1=lines_1.filter(good_line_1)

transactions_features=clean_lines_1.map(lambda l: (l.split(',')[8],1))
print(transactions_features.take(1))

# Step 1: Extracting the first elements of each tuple
first_elements = transactions_features.map(lambda x: float(x[0]))

# Step 2: Summing up all the first elements
sum_first_elements = first_elements.reduce(lambda x, y: x + y)

# Step 3: Counting the total number of tuples
count = transactions_features.count()

# Step 4: Calculating the average
average_first_elements = sum_first_elements / count

print("Average of first elements: ", average_first_elements)
```

Inference

The average gas used per transaction is 161725.7 units.

The miner 0x341e790174e3a4d35b65fdc067b6b5634a61caea used more gas per transaction than the average gas used per transaction. While the other nine miners used less gas per transaction than the average gas used per transaction.

Data Overhead

Here we assumed the following columns as overhead, they are sha3_uncles, logs_bloom, transactions_root, state_root, and receipts_root.

To calculate the total overhead, we can first calculate the individual column size. Then we can sum the size of each column and then multiply by 4 to get the final size.

The code for the same is as following.

```
# Reading lines from csv file
   lines_1 = spark.sparkContext.textFile("s3a://" + s3_data_repository_bucket + "/ECS765/ethereum-parvulus/blocks.csv")

   clean_lines_1=lines_1.filter(good_line_1)

   sha3_uncles = clean_lines_1.map(lambda l: (l.split(',')[4],len(l.split(',')[4])))
   print(sha3_uncles.take(3))

   sum1 = sha3_uncles.map(lambda x: x[1]).reduce(lambda a, b: a + b)
   print(sum1)

   logs_bloom = clean_lines_1.map(lambda l: (l.split(',')[5],len(l.split(',')[5])))
   print(logs_bloom.take(3))

   sum2 = logs_bloom.map(lambda x: x[1]).reduce(lambda a, b: a + b)
   print(sum2)

   transactions_root = clean_lines_1.map(lambda l: (l.split(',')[6],len(l.split(',')[6])))
   print(transactions_root.take(3))

   sum3 = transactions_root.map(lambda x: x[1]).reduce(lambda a, b: a + b)
   print(sum3)

   state_root = clean_lines_1.map(lambda l: (l.split(',')[7],len(l.split(',')[7])))
   print(state_root.take(3))

   sum4 = state_root.map(lambda x: x[1]).reduce(lambda a, b: a + b)
   print(sum4)
   receipts_root = clean_lines_1.map(lambda l: (l.split(',')[8],len(l.split(',')[8])))
   print(receipts_root.take(3))

   sum5 = receipts_root.map(lambda x: x[1]).reduce(lambda a, b: a + b)
   print(sum5)

   sum = sum1+sum2+sum3+sum4+sum5
   print('The size is {} '.format(sum))
```

The sum of size is 5446000839.