Big Data Processing ECECS765P Shabnam Khodadadi 210936704

Github repository: https://github.com/shabnam-kh/Ethereum-Analysis

PART A. TIME ANALYSIS (20%)

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.

MRJob ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_4832/
Tools: Hadoop Map-Reduce
Method definition:

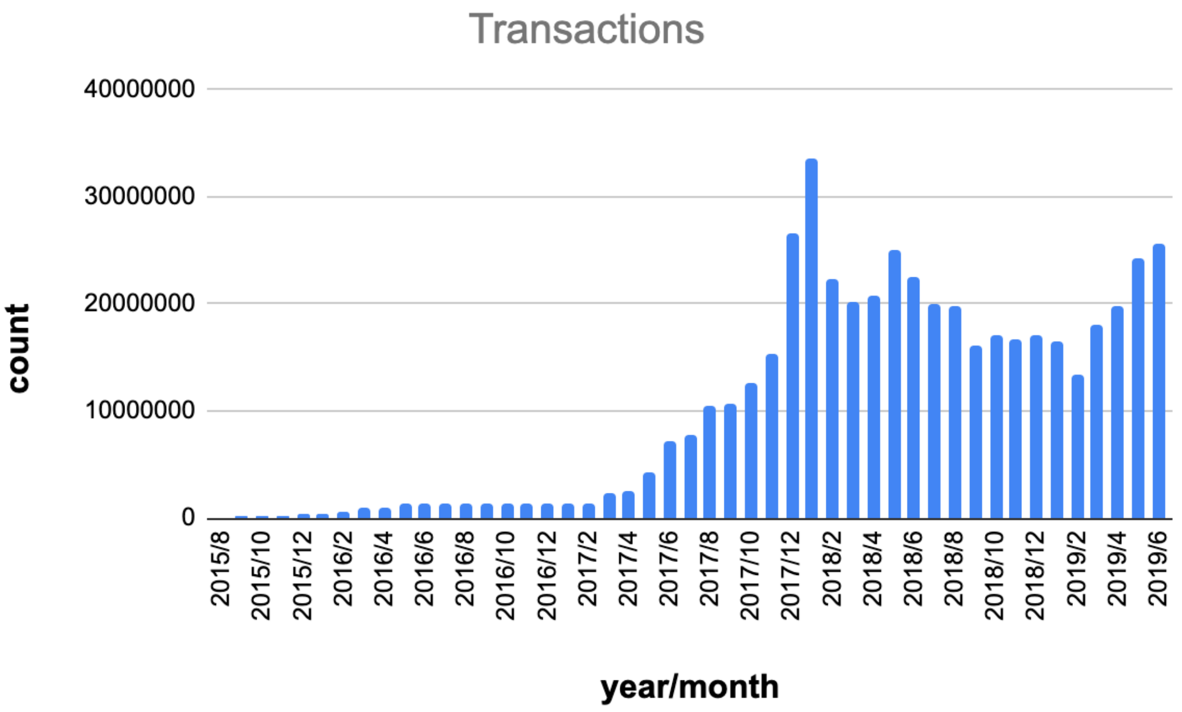
In the map, read the transactions lines, split the lines with comma, check the number of fields to figure out that data is correct, just skip the header line, fields[6] is the transaction time in timestamp format, we should convert it to the datetime object to extract the year and month. return the key:(str(dt_object.month),str(dt_object.year)) and the value is 1.In the reducer, return the input key and sum of values

Sample Output:

```
["1", "2017"]   1409664
["1", "2019"]   16569597
["10", "2015"]  205213
["10", "2017"]  12579172
["11", "2016"]  1301586
["11", "2018"]  16713911
```

Result:

The bar chart shows the counts of transactions in different months(for different years) from 8/2015 to 6/2019. Overall, the number of transactions increased during that time. The most transaction during the period occurs in the December of 2017.



PART B. TOP TEN MOST POPULAR SERVICES (25%)

Evaluate the top 10 smart contracts by total Ether received. An outline of the subtasks required to extract this information is provided below, focusing on a MRJob based approach. This is, however, is not the only way to complete the task, as there are several other viable ways of completing this assignment.

MRJob ID:
application_1648683650522_2331 top10_addresses.py SPARK sk018 root.users.sk018 RUNNING UNDEFINED 10%
http://itl331.student.eecs.qmul.ac.uk:4040
Tools: Spark
Method definition:
Read the contracts file, filter the correct lines with specific fields number(for contract it should be 5 field), the map output is (to_address,1), also read the transactions file, filter the correct lines, return the to_address filed and value(ether recieved),the map output is (to_address, value), join the transactions and contracts on the address field the output is (address, (value,1)). In the reduce sum the transactions value and counts, the output is (sum(values),sum(counts)), then we should sort the results based on values descending and get the top10. The output is (address, value)
Result:

Rank	Transaction Address	Total Ether recieved
1	0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444	8.42E+25
2	0xfa52274dd61e1643d2205169732f29114bc240b3	4.58E+25
3	0x7727e5113d1d161373623e5f49fd568b4f543a9e	4.56E+25
4	0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef	4.32E+25
5	0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8	2.71E+25
6	0xbfc39b6f805a9e40e77291aff27aee3c96915bdd	2.11E+25
7	0xe94b04a0fed112f3664e45adb2b8915693dd5ff3	1.56E+25
8	0xbb9bc244d798123fde783fcc1c72d3bb8c189413	1.20E+25
9	0xabbb6bebfa05aa13e908eaa492bd7a8343760477	1.17E+25
10	0x341e790174e3a4d35b65fdc067b6b5634a61caea	8.38E+24

PART C. TOP TEN MOST ACTIVE MINERS (15%)

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.
MRJob ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_2368/
Tools: Hadoop Map-Reduce
Method definition:
We have 2 reducer/mapper, we use steps to handle this. In the first map check the blocks lines if correct(have specific fields, 9). Then we extract the miner address and the size which is field 2 and 4, then reducer return the (miner , sum(size)) in the second mapper we aggregate the results and return (miner, sum(size)) finally in the second reducer we sort the values and return the top10, (miner, size)
Result:

Rank	Miner Address	Size of blocks
1	0xea674fdde714fd979de3edf0f56aa9716b898ec8	23989401188
2	0x829bd824b016326a401d083b33d092293333a830	15010222714
3	0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c	13978859941
4	0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5	10998145387
5	0xb2930b35844a230f00e51431acae96fe543a0347	7842595276
6	0x2a65aca4d5fc5b5c859090a6c34d164135398226	3628875680
7	0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01	1221833144
8	0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb	1152472379
9	0x1e9939daaad6924ad004c2560e90804164900341	1080301927
10	0x61c808d82a3ac53231750dadc13c777b59310bd9	692942577

PART D. DATA EXPLORATION (40+%)

SCAM ANALYSIS

Popular Scams

Utilising the provided scam dataset, what is the most lucrative form of scam?
MRJob ID:

```
application_1649894236110_5320        scams.py          SPARK       sk018
root.users.sk018            RUNNING         UNDEFINED          10%
http://itl331.student.eecs.qmul.ac.uk:4040
```

Tools: Spark

Method definition:

For this task we use spark SQL which is a module to work with dataframes and datasets. First, read the data in scams.json file (our target data is under the “results” field) with spark sql and save it in the dataframe to work with. Fetch the fields of (list of addresses, category, status), the fields that we need for our process and drop the rest. In the next step there is a map that return each address of the list along with category and status.the output is (scam_adress, (category, status, gas_provided)). Then we read the transactions file, check if the line are correct and finally extract the (to_address, (value, time)). Now we can join the scams with transaction on the address field, the output is (to add, ((val, time), (cat, status))), then use the map to have the fields that we need, the output is (scam_category, transactions_value), then we apply the reduce by scam_category and return the sum of values for each scam category. Sort it and print it.

Sample Output:

```
(u'Scamming', 3.90533915076866e+22)
(u'Phishing', 3.127925794719535e+22)
(u'Fake ICO', 1.35645756688963e+21)
```

Result:

Related to the results the most lucrative form of scam is scamming category which has the highest transaction value. In addition to transaction value, results show that the scamming category has the more number of scams than two other category.

Does this correlate with certainly known scams going offline/inactive?

Scamming

Method definition:

Use the output RDD of join from previous part (transactions join scams on address), filter that the type of category is scamming,

For time/value: use map to return (transactions_value, time), apply reduce to sum up the values, sort by time in ascending order and save the result in text file named scamming

For time/status: use map to return ((status, time), 1), apply reduce to sum up the counts, sort by time in ascending order and save the result in text file named scamming_status

For time/gas: use map to return (gas, time), apply reduce to sum up the gases, sort by time in ascending order and save the result in text file named scamming_gas

Sample Output:

For time/value:

```
('17.06', 9.87841012e+18)
('17.07', 2.45268175362857e+21)
('17.08', 2.984374287e+19)
```

```
('17.09', 1.815640348962187e+20)
('17.10', 1.8390126786632517e+21)
('17.11', 2.2864587814574449e+18)
('17.12', 2.6928221538918744e+19)
```

For time/status:

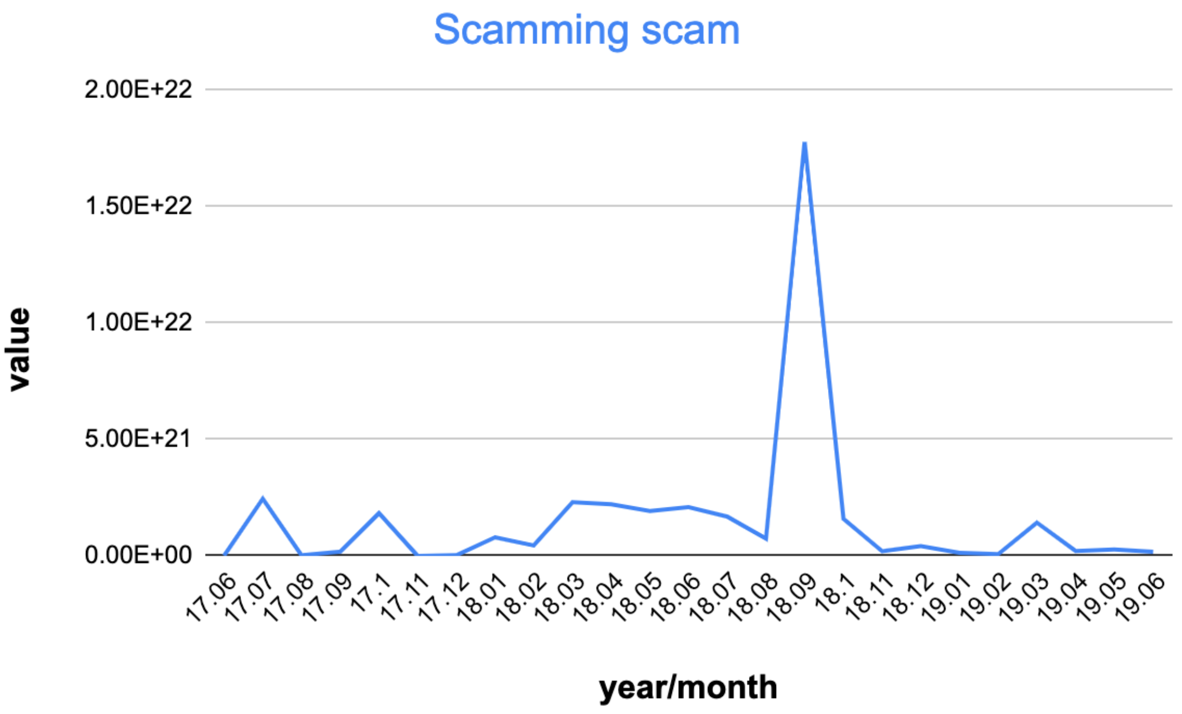
```
(('17.06', u'Active'), 2)
(('17.06', u'Offline'), 5)
(('17.07', u'Active'), 8)
(('17.07', u'Offline'), 1389)
(('17.08', u'Active'), 2)
(('17.08', u'Offline'), 19)
(('17.09', u'Active'), 2)
(('17.09', u'Offline'), 62)
```

For time/gas:

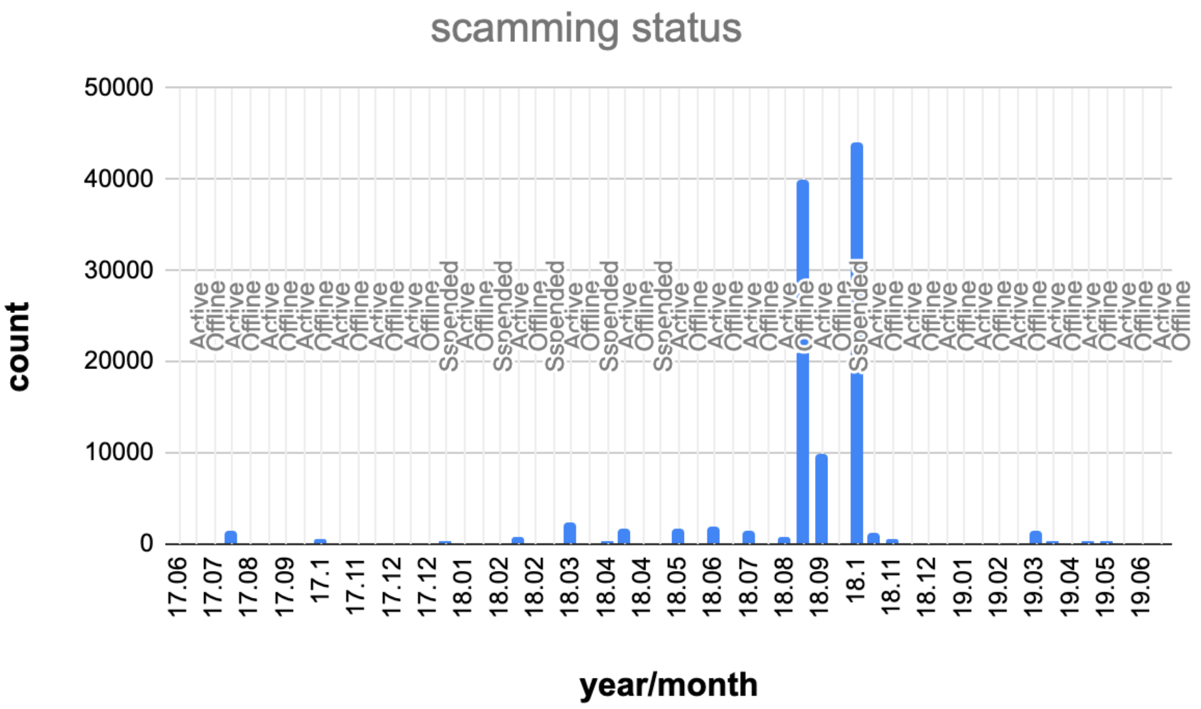
```
('17.06', 950998.0)
('17.07', 285077267.0)
('17.08', 3169557.0)
('17.09', 22516938.0)
('17.10', 88803856.0)
('17.11', 11847000.0)
('17.12', 7233833.0)
('18.01', 14399172.0)
```

Result:

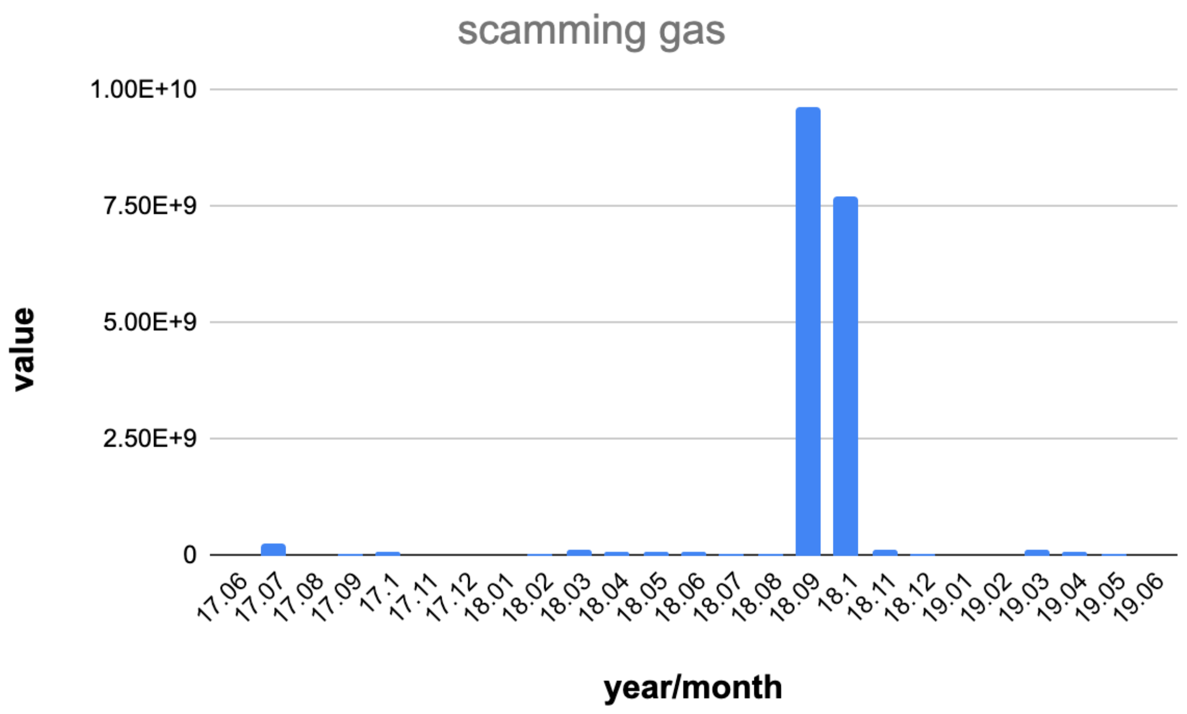
As we can see in the diagram, the scamming category reach its highest value in the month 09 of year 2018. Related to the status chart, in this time scamming has the most highest active number of scams approximately 10,000 and zero number of offline scams in comparison to other time of the period, also the peak of the gas chart is occurred in this exact time means users spend the most value of gas in this time. In conclusion, in the time that there are more ether, the most lucrative scam works hardly to hunt.



Scamming scam
value
2.00E+22
1.50E+22
1.00E+22
5.00E+21
0.00E+00
17.06
17.07
17.08
17.09
17.1
17.11
17.12
18.01
18.02
18.03
18.04
18.05
18.06
18.07
18.08
18.09
18.1
18.11
18.12
19.01
19.02
19.03
19.04
19.05
19.06
year/month



scamming status
count
50000
40000
30000
20000
10000
0
Active
Offline
Ssspended
17.06
17.07
17.08
17.09
17.1
17.11
17.12
18.01
18.02
18.03
18.04
18.05
18.06
18.07
18.08
18.09
18.1
18.11
18.12
19.01
19.02
19.03
19.04
19.05
19.06
year/month



phishing

Method definition:

Use the output RDD of join from previous part (transactions join scams on address), filter that the type of category is phishing,

For time/value: use map to return (transactions_value, time), apply reduce to sum up the values, sort by time in ascending order and save the result in text file named phishing

For time/status: use map to return ((status, time), 1), apply reduce to sum up the counts, sort by time in ascending order and save the result in text file named phishing_status

For time/gas: use map to return (gas, time), apply reduce to sum up the gases, sort by time in ascending order and save the result in text file named phishing_gas

Sample Output:

For time/value:

```
('17.05', 9e+16)
('17.06', 1e+18)
('17.07', 1.0600967321322426e+22)
('17.08', 6.456006316562121e+21)
('17.09', 3.6021275371606943e+21)
```

For time/status:

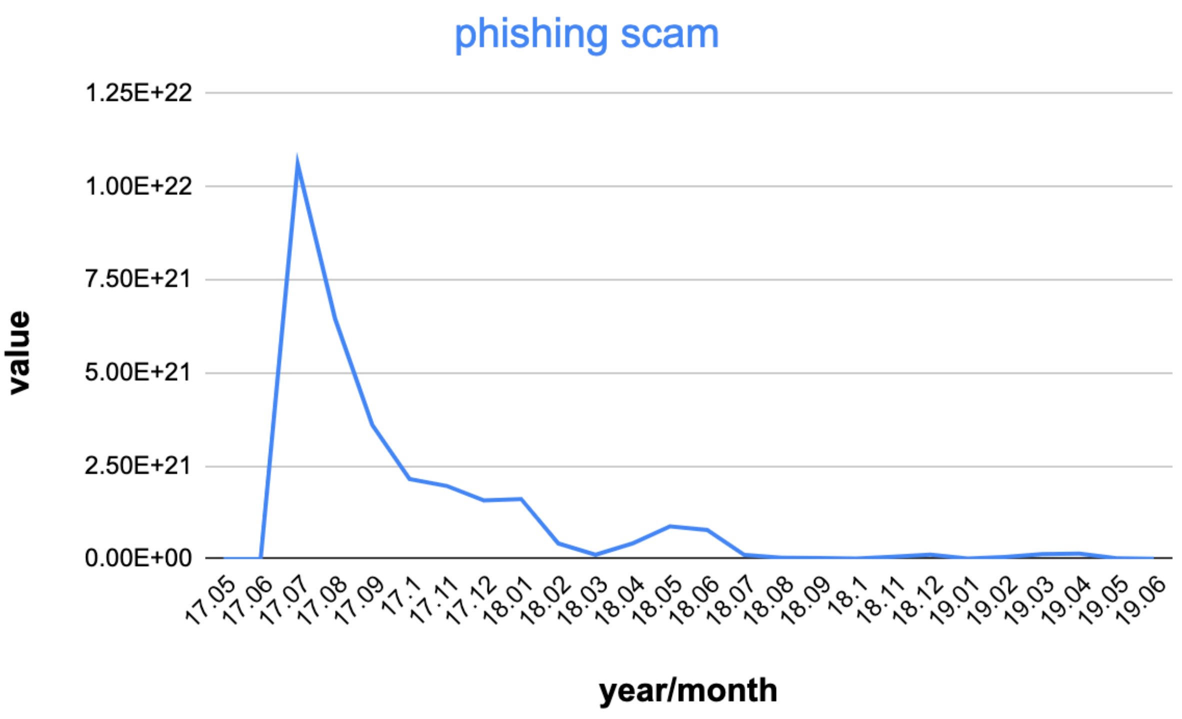
```
(('17.05', u'Offline'), 1)
(('17.06', u'Offline'), 1)
(('17.07', u'Active'), 150)
(('17.07', u'Offline'), 477)
(('17.08', u'Active'), 13)
(('17.08', u'Offline'), 748)
```

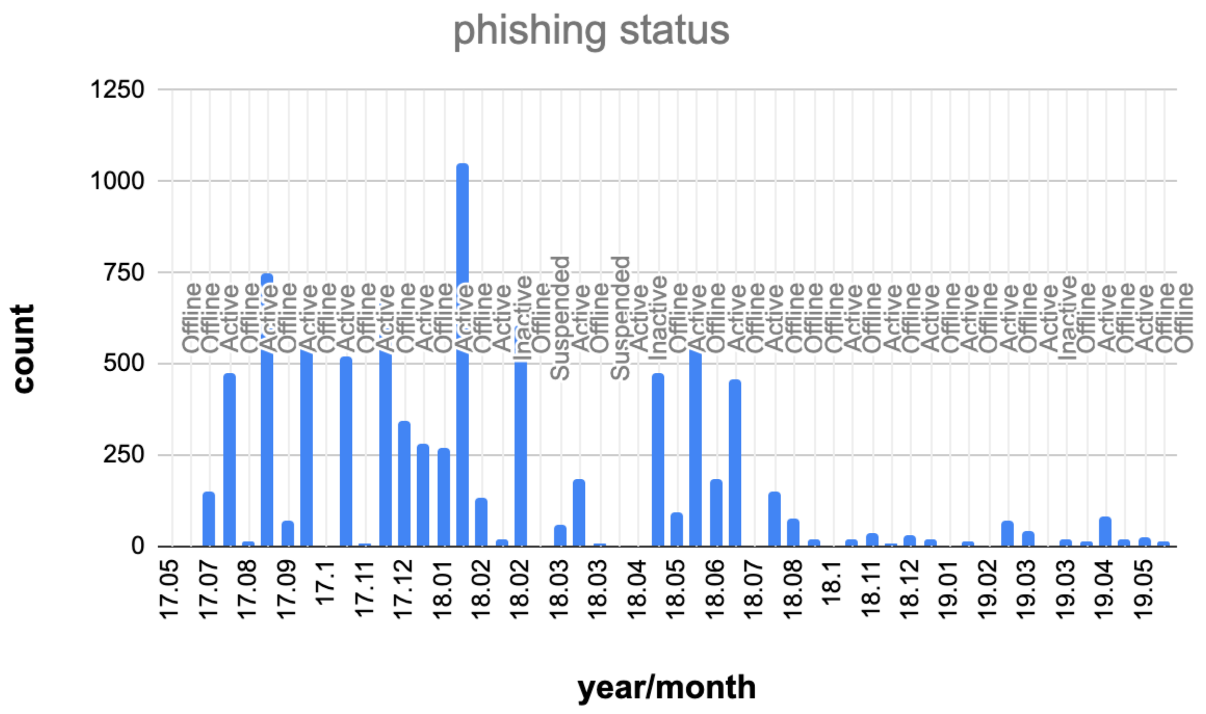
For time/gas:

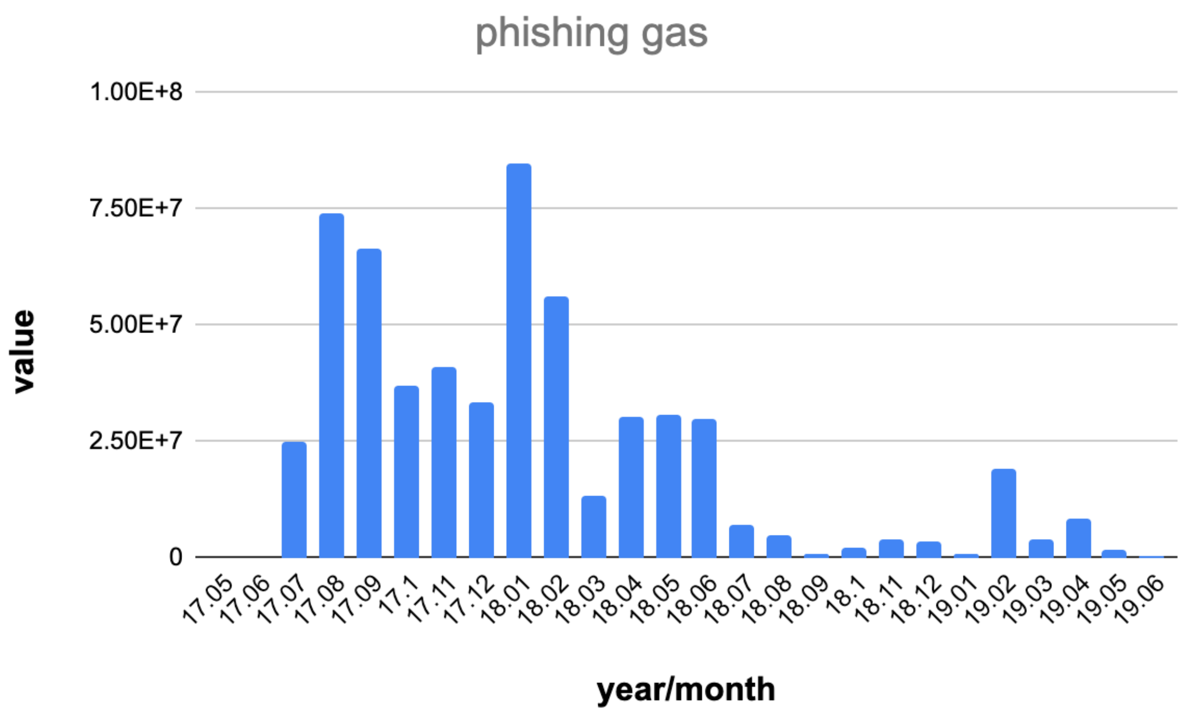
```
('17.05', 21000.0)
('17.06', 90000.0)
('17.07', 24979617.0)
('17.08', 74237767.0)
('17.09', 66396031.0)
('17.10', 36916610.0)
('17.11', 40876917.0)
```

Result:

As we can see in the diagram, the phishing category reach its highest value in the month 07 of year 2017.




phishing status
count
year/month
1250
1000
750
500
250
0
17.05
17.07
17.08
17.09
17.1
17.11
17.12
18.01
18.02
18.02
18.03
18.03
18.04
18.05
18.06
18.07
18.08
18.1
18.11
18.12
19.01
19.02
19.03
19.03
19.04
19.05
Offline Offline Active Offline Active Offline Active Offline Active Offline Active Offline Active Offline Active Offline Active Inactive Offline Suspended Active Offline Suspended Active Inactive Offline Active Offline Active Offline Active Offline Offline Offline Active Offline Active Offline Active Offline Active Offline Active Offline Active Inactive Offline Active Offline Active Offline Offline


phishing gas
value
year/month
1.00E+8
7.50E+7
5.00E+7
2.50E+7
0
17.05
17.06
17.07
17.08
17.09
17.1
17.11
17.12
18.01
18.02
18.03
18.04
18.05
18.06
18.07
18.08
18.09
18.1
18.11
18.12
19.01
19.02
19.03
19.04
19.05
19.06

fICO

Method definition:

Use the output RDD of join from previous part (transactions join scams on address), filter that the type of category is fICO,
For time/value: use map to return (transactions_value, time), apply reduce to sum up the values, sort by time in ascending order and save the result in text file named fICO
For time/status: use map to return ((status, time), 1), apply reduce to sum up the counts, sort by time in ascending order and save the result in text file named fICO_status
For time/gas: use map to return (gas, time), apply reduce to sum up the gases, sort by time in ascending order and save the result in text file named fICO_gas

Sample Output:

For time/value:

```
('17.06', 1.8267402332376325e+20)
('17.07', 1.6242199484949187e+19)
('17.08', 1.8116466237713616e+20)
('17.09', 9.751383634137814e+20)
('18.06', 1.23831829e+18)
```

For time/status:

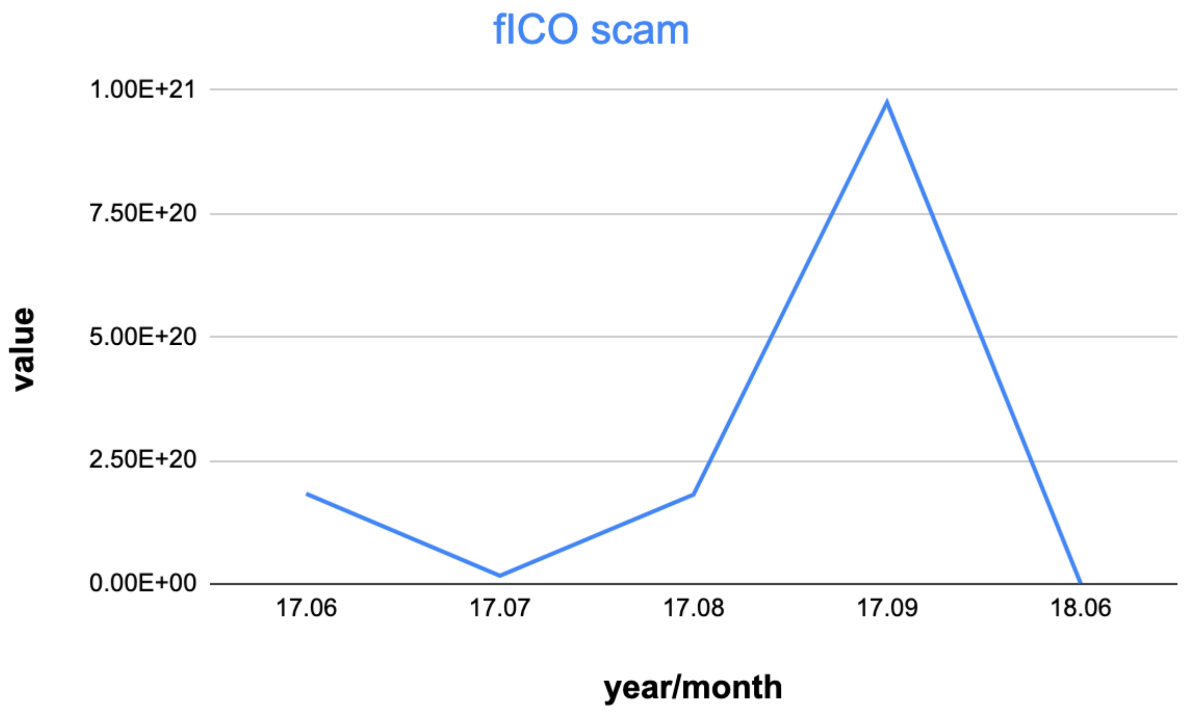
```
(('17.06', u'Offline'), 30)
(('17.07', u'Offline'), 11)
(('17.08', u'Offline'), 35)
(('17.09', u'Offline'), 42)
(('18.06', u'Offline'), 3)
```

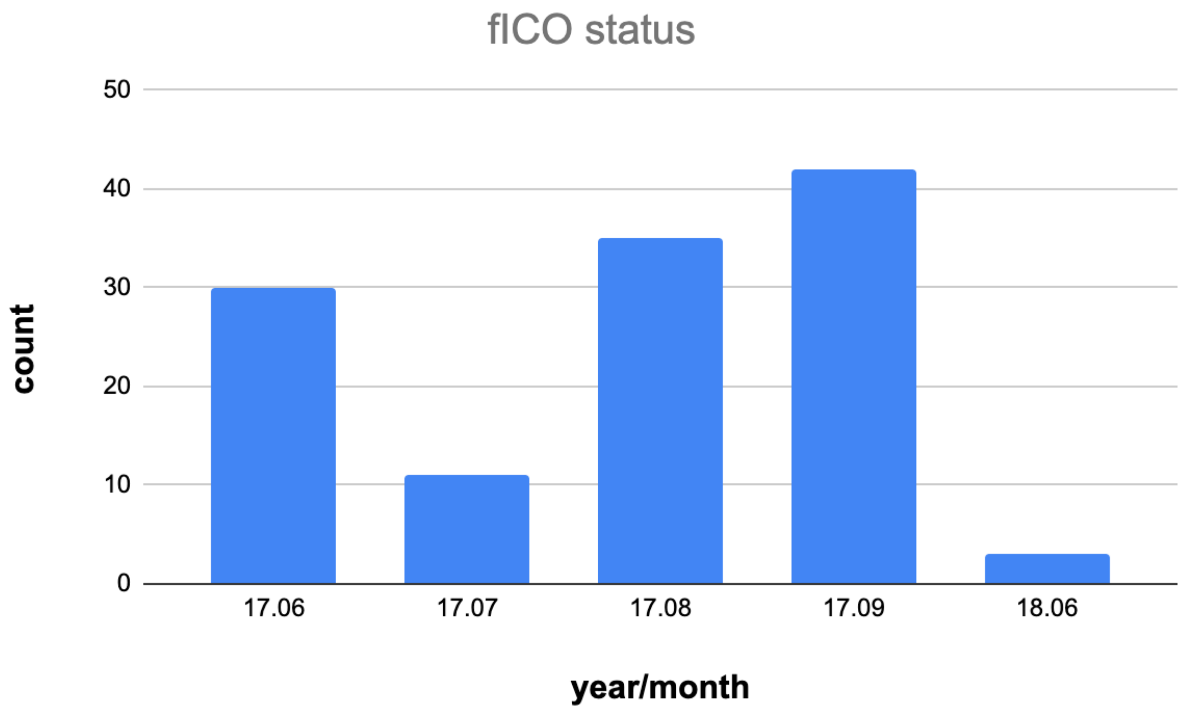
For time/gas:

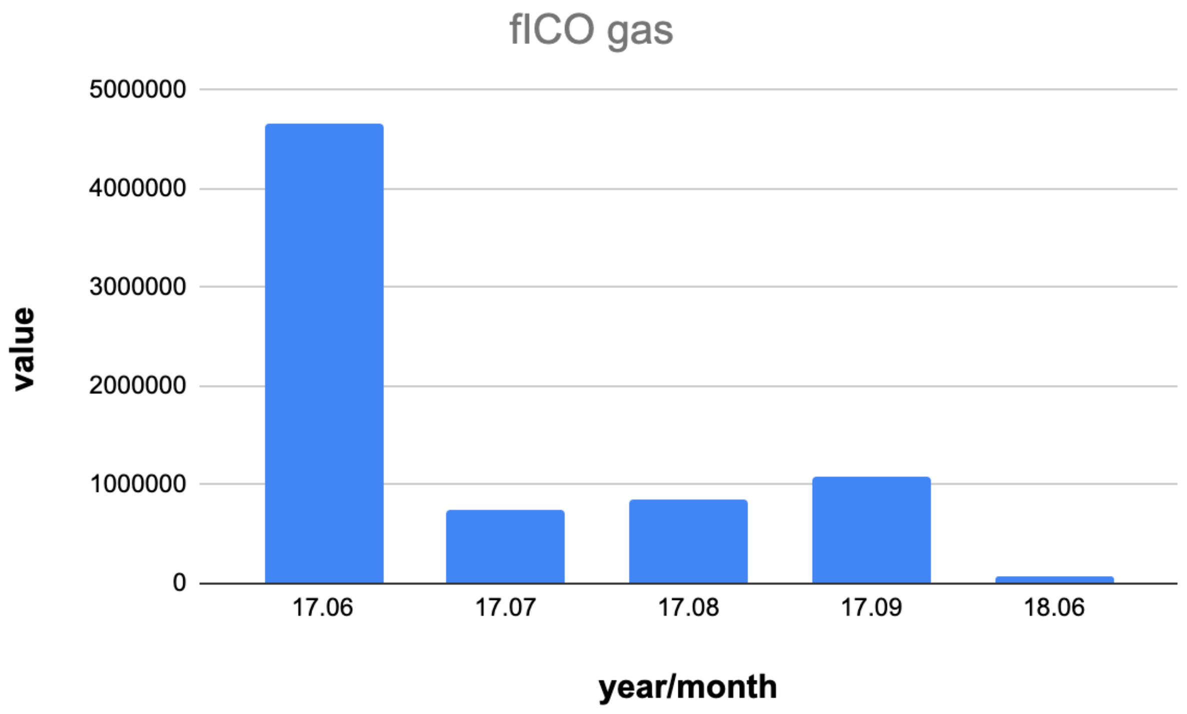
```
('17.06', 4655000.0)
('17.07', 756000.0)
('17.08', 843202.0)
('17.09', 1083000.0)
('18.06', 65100.0)
```

Result:

As we can see in the diagram, the fICO category reach its highest value in the month 09 of year 2017.


fICO scam
1.00E+21
7.50E+20
5.00E+20
2.50E+20
0.00E+00
value
17.06
17.07
17.08
17.09
18.06
year/month


fICO status
count
50
40
30
20
10
0
17.06
17.07
17.08
17.09
18.06
year/month


fICO gas
value
5000000
4000000
3000000
2000000
1000000
0
17.06
17.07
17.08
17.09
18.06
year/month

Compare the peak time of scam categories to each other

We can infer from the below table that, scamming is the most lucrative category in September of 2018, in this exact time the phishing category doesnt have any active scams and no data of fICO scam, so the only active scam (scamming) becomes the most lucrative one.
In the July of 2017, phishing is the most lucrative one, we can see that at this time phishing has 50 active scams in comparison to scamming and fICO which have 8,0 active.
fICO in September of 2017 reaches its peak of transaction value but it is not still the more lucrative one as phishing has 69 active scams and 3.6021275371606943e+21 transaction values which is more than scamming and fICO.

time	scamming	phishing	fico
09/2018 (scamming peak)	Value: 1.777207356325177 7e+22 Active: 40035 Offline: 9851 Suspended:1 Gas: 9654606702.0	Value: 2.93727866355e+19 Offline: 21 Active:- Gas: 937000.0	
07/2017 (phishing peak)	Value: 2.45268175362857e +21 Active: 8 Offline: 1389 Gas: 285077267.0	Value: 1.060096732132242 6e+22 Active: 50 Offline: 477 Gas: 24979617.0	Value: 1.624219948494918 7e+19 Offline: 11 Gas: 756000.0
09/2017 (fICO peak)	Value: 1.815640348962187 e+20 Active: 2 Offline: 62 Gas: 22516938.0	Value: 3.602127537160694 3e+21 Active: 69 Offline: 561 Gas: 66396031.0	Value: 9.751383634137814 e+20 Offline:42 Gas: 1083000.0

MISCELLANEOUS ANALYSIS

Fork the Chain

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? (10%)

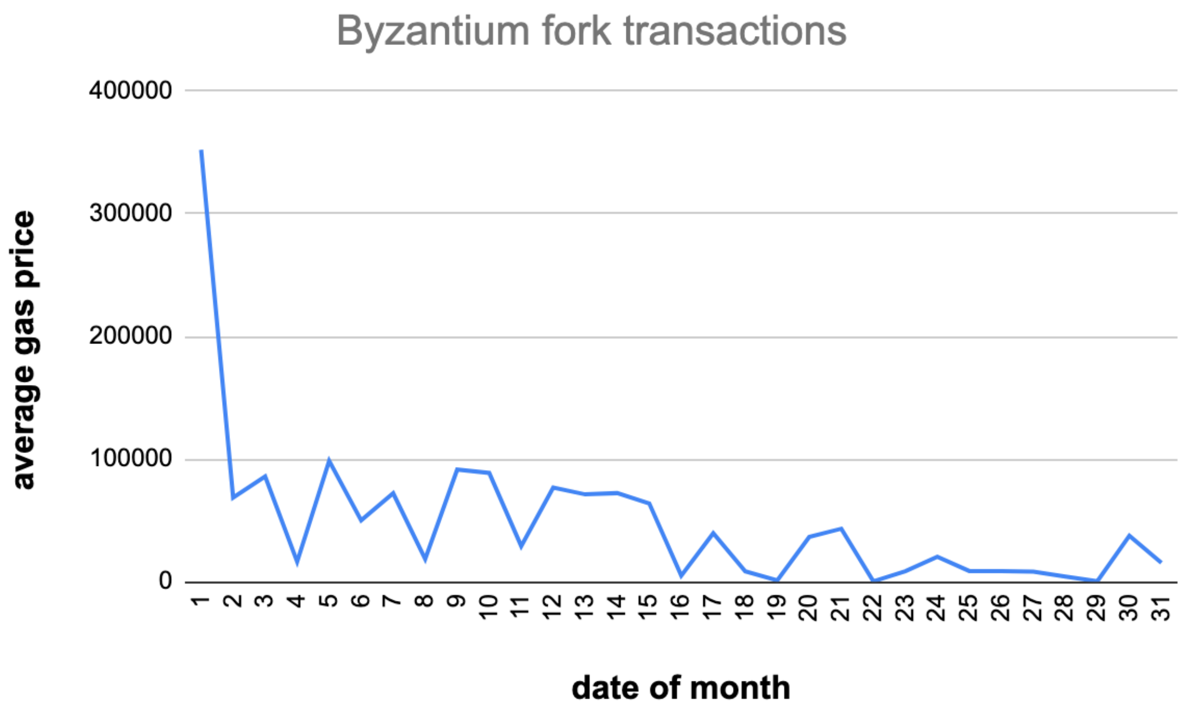
MRJob ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_5222/

Related to the time analysis of part A of our dataset, our transaction data is from 08/2015 to 06/2019, so we should choose forks which is lie on this period of time.
We decide to analyze The Byzantium fork which occurred on Oct-16-2017 05:22:11 AM +UTC
For this purpose we should extract the transaction data for October in the year 2017. By this, we can compare the average price and transaction value on the day of Byzantium fork (16th of October) to the days before and after the fork in the same months.

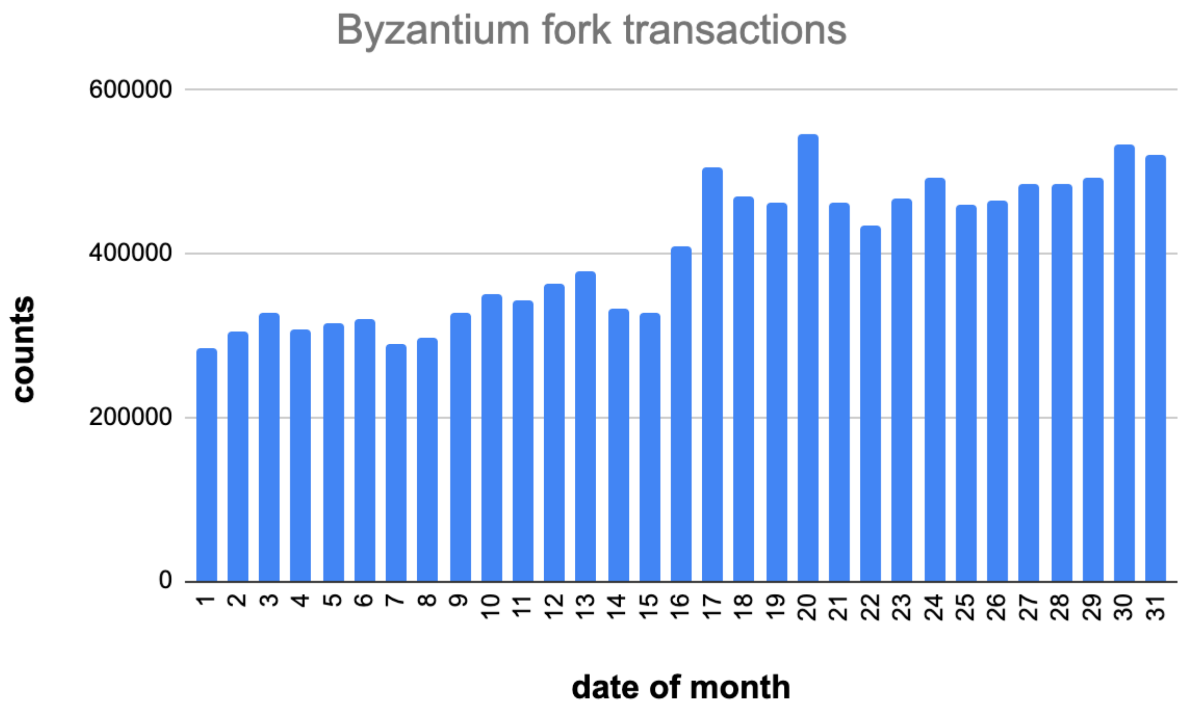
Tools: Hadoop Map-Reduce
Method definition:
Read transaction lines, check if lines are correct, extract the gas_price and transaction_time which is field[5] and field[6]. Check if the transaction year is 2017 and the month is October if it is matched then returns the (day, (gas_price, 1)). In the reducer sum up the gas prices also sum up the counts, then divide the sum(gas_price) by counts to get the gas_price average, the output is (day, (transactions_count, average_gas_price)). We also use a combiner in this task which is equal to the reducer code
Sample Output:

```
1	[283871, 352272.68724174006]
10	[349605, 88671.50069363996]
12	[363411, 76857.4543120599]
14	[332169, 72252.3775547989]
16	[408297, 4898.395040864861]
18	[470449, 8502.51568182736]
21	[462902, 43205.68932517034]
23	[468168, 8543.941493224655]
25	[460419, 8687.738777070452]
27	[486147, 8227.963969745777]
```

Result:
Related to the price diagram, the average gas price in the 16th October is in the lowest amount of it compare to the days before and after. So the price plummet occurred on the fork day. In my opinion, users will have profited most from this because they can have more transactions by paying less.



Based on the transaction count diagram for the month that the fork occurred, we can see that the number of transactions increased on the day of the fork, maybe it is because of the low gas price. The transactions gradually grow from the day after the fork.



Gas Guzzlers

For any transaction on Ethereum a user must supply gas. How has gas price changed over time?

MRJob ID:

```
application_1648683650522_6493          gas.py              SPARK           sk018
root.users.sk018              RUNNING           UNDEFINED           10%
```

http://itl264.student.eecs.qmul.ac.uk:4040

Tools: Spark

Method definition:

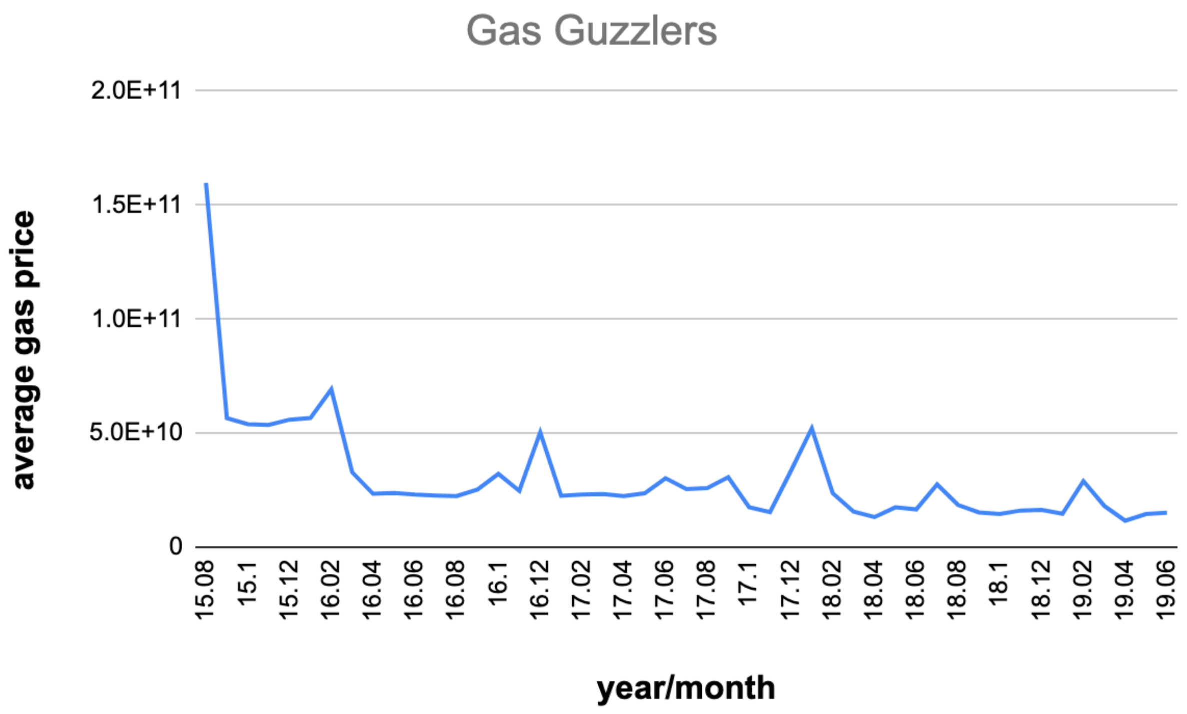
Read the transaction lines, check if lines are correct, fetch the transaction_time and gas price, use map for count transactions, the output is (time, (gas_price,1)), apply the reduce, sum up the prices and the counts and calculate the average price, the output is (time, avg_price), sort by time in ascending order, save the result in AveGas text file.

Sample Output:

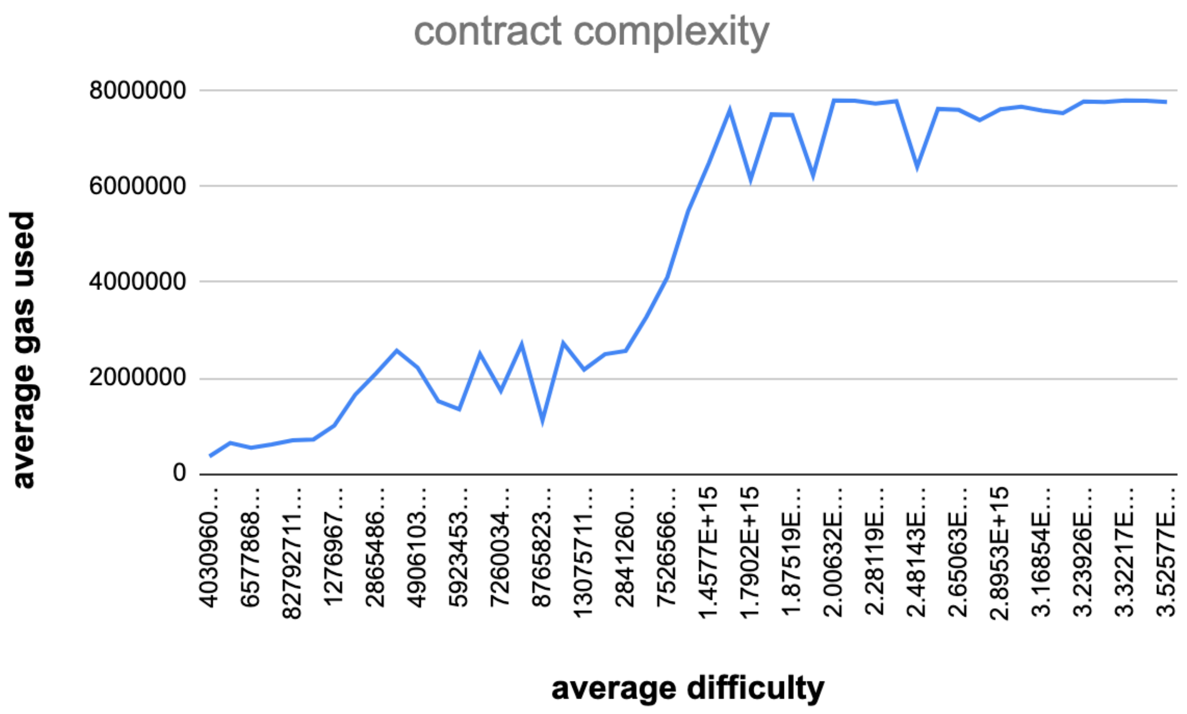
```
('15.08', 159744029578.0333)
('15.09', 56511301521.03311)
('15.10', 53901692120.53661)
('15.11', 53607614201.79755)
('15.12', 55899526672.35286)
('16.01', 56596270931.31675)
('16.02', 69180681134.38782)
```

('16.03', 32797039087.356033)

Result:
Gas price decreased during the period of 2015 to 2019



Have contracts become more complicated, requiring more gas, or less so?
Method definition:
Read contract lines, check if line is correct, extract the block_address and count, the map out put is (block_address, 1). Then read the blocks line, check if correct, fetch the (block_number, (difficulty, gas_used, timestamp)) fields. Join the contracts with blocks on the block_number and the output is (time, (avg_difficulty, avg_gas)), save the result in Diff_Time file.
Sample Output:
```
('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))
```
Result:
Results show that the more complex contract is the more gas it used, On the other word by increasing the average difficulty of contract, the average gas used will grow up.



Could you correlate the complexity for some of the top-10 contracts found in Part-B by observing the change over their transactions
We choose this contract from PartB top10 to observe

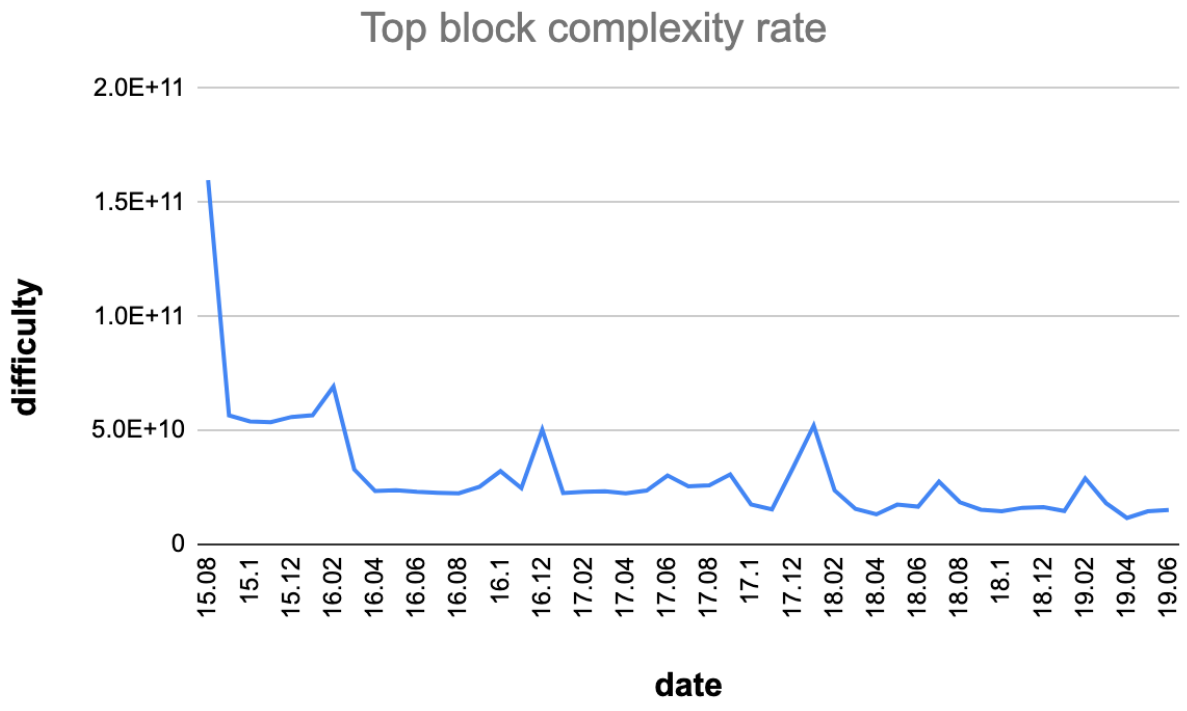
target_contract_address = '0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444

MRJob ID:
application_1649894236110_0283 gas.py SPARK sk018 root.users.sk018 RUNNING UNDEFINED 10%
http://itl264.student.eecs.qmul.ac.uk:4040
Tools: Spark
Method definition:
Do the same things in the previous session, but first we filter the contracts with the address to just have the data of the target contract.
Sample Output:

```
('15.08', 159744029578.0333)
('15.09', 56511301521.03311)
('15.10', 53901692120.53661)
```

```
('15.11', 53607614201.79755)
('15.12', 55899526672.35285)
('16.01', 56596270931.31675)
('16.02', 69180681134.38783)
('16.03', 32797039087.356033)
```

Result:
Related to the line chart of top block complexity, the difficulty of the targeted contract decreased during the period time.



Comparative Evaluation

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? (10%)

Map-reduce

First run:
step1
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_1864/
CPU time spent (ms)=8587200
GC time elapsed (ms)=112248

Step2
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_1983/
CPU time spent (ms)=625190
GC time elapsed (ms)=1672

9212390
113920

Second run:
Step1
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_3014/
CPU time spent (ms)=8296160
GC time elapsed (ms)=171179
Step2
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_3036/
CPU time spent (ms)=502310
GC time elapsed (ms)=914

8798470
172093

Third run:
Step1
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_4771/
CPU time spent (ms)=8065480
GC time elapsed (ms)=139378
Step2
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_4791/
CPU time spent (ms)=638140
GC time elapsed (ms)=1557

8703620
140935

Average CPU time spent (ms)=8904826 = 8904.826 seconds
Average GC time elapsed (ms)=142316= 142.316 seconds

Spark

First run:
```
application_1649894236110_4967    top10_addresses.py         SPARK          sk018
root.users.sk018                  RUNNING            UNDEFINED           10%
http://itl264.student.eecs.qmul.ac.uk:4040
```
Duration : 135.649525881

Second run:

```
application_1649894236110_4984      top10_addresses.py              SPARK           sk018
root.users.sk018                    RUNNING                 UNDEFINED               0%
http://itl264.student.eecs.qmul.ac.uk:4040
Duration:193.468039989
```

Third run:

```
application_1649894236110_4990      top10_addresses.py              SPARK           sk018
root.users.sk018                    ACCEPTED                UNDEFINED               0%
N/A
Duration:142.394073963
```

Average = 157 seconds

Conclusion:
Map-reduce job take more time than spark job for this task.

The reason is because of approach to processing: Spark can do it in-memory, while Hadoop MapReduce has to read from and write to a disk. As a result, the speed of processing differs significantly – Spark is 56 times faster than hadoop in this case.

Hadoop_run_time (8904) = Spark_run_time (157) * 56

Spark is more faster for join datasets than hadoop