Coursework 2022: Ethereum Analysis

Big Data Processing ECS765P

Tushara Govinda Reddy EC211256

MSc Big Data Science with Machine Learning Systems
2021/22-Sem B

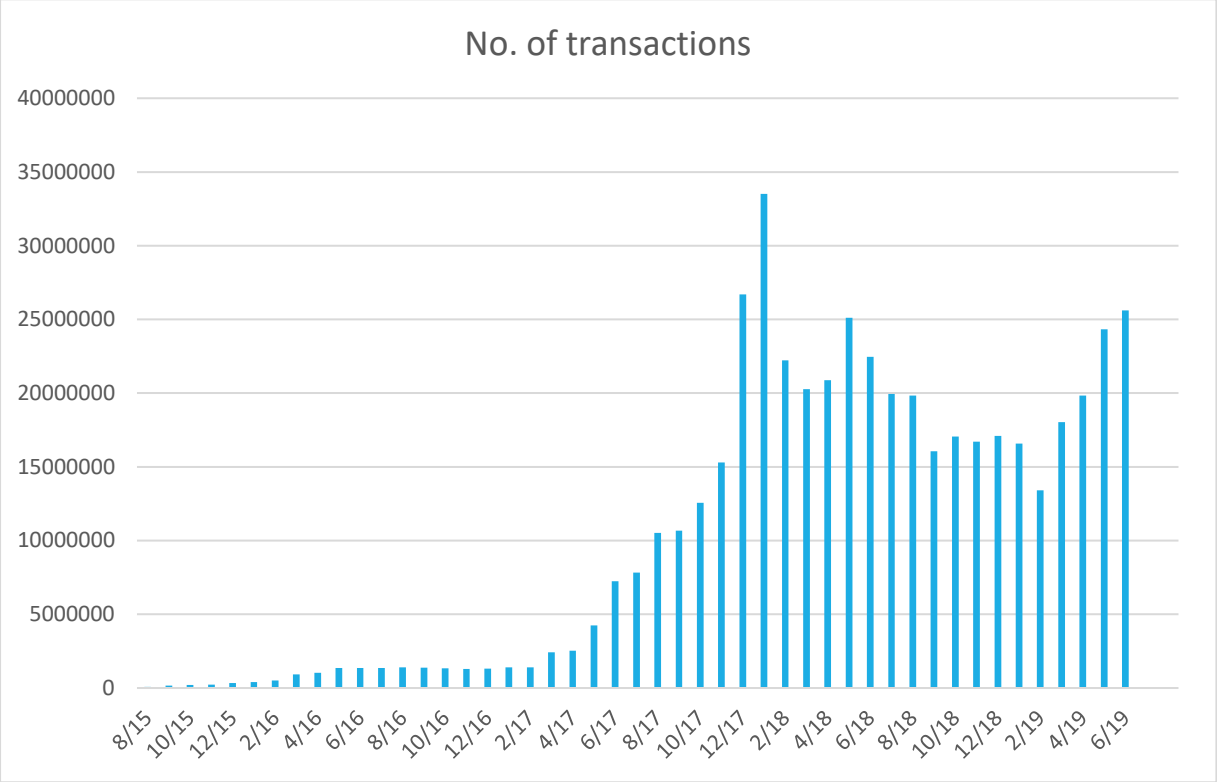
Table of Contents

PART A. TIME ANALYSIS (20%)

Job1

JobURL:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1637317090236_22456/



Key:

Y- Axis: Number of transactions aggregated per month

X-Axis: Chronologically sorted months, years

Method:

Mapper:

1. Import MRJobs
2. Read input 'data/ethereum/transactions' file line by line and split them at every ',' .
3. Using the field 'block_timestamp' as the reference for month, year and its occurrence to calculate the no. of transaction per month per year.

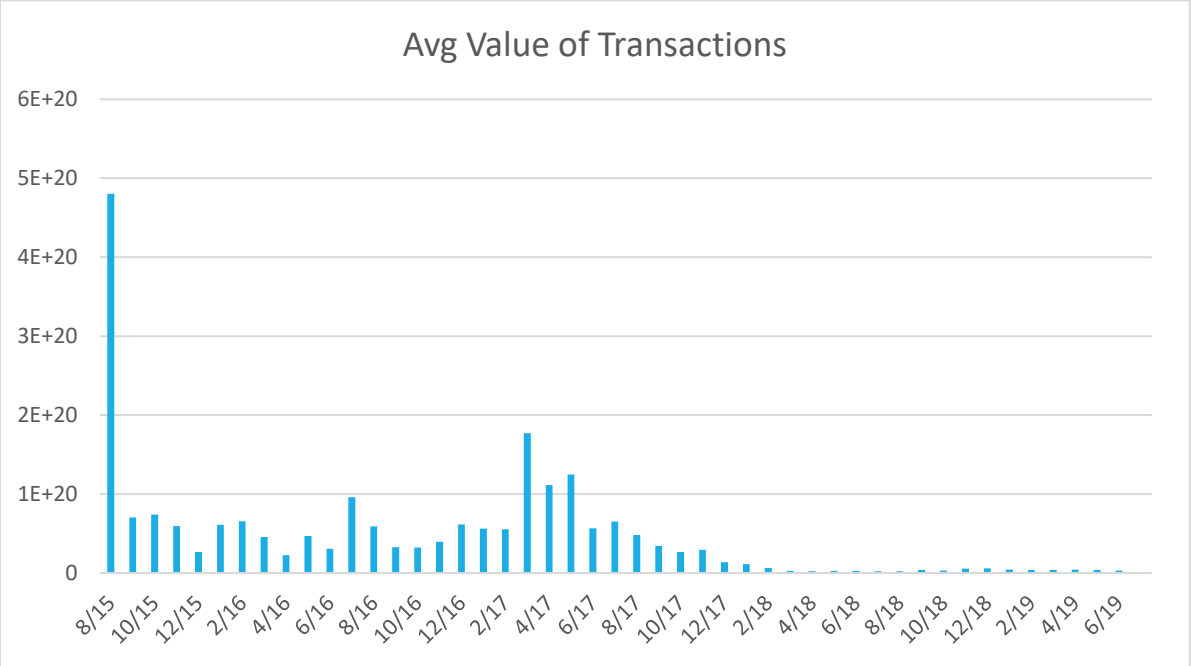
Combiner:

1. Receives the key=date.tm_mon,date.tm_year and val= count of the transactions.
2. Combiner performs local aggregation of the key-val pair from each mapper node and writes to the local disk.

Reducer:
1. Receives the key=date.tm_mon,date.tm_year and val= count of the transactions from all mapper nodes and preforms the aggregation at reducer node.

Job2:

Job URL:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1637317090236_22663/



Key:
Y- Axis: Average Value of Transactions
X-Axis: Chronologically sorted months, years

Method:
Mapper:
1. Import MRJobs
2. Read input 'data/ethereum/transactions' file line by line and split them at every ',' .
3. Using the field 'block_timestamp' as the reference for month, year and the 'value' field for calculation of the average value of transactions.

Combiner:
1. Receives the key=date.tm_mon,date.tm_year month,year and val= (1,val) value aggregated per month.
2. Combiner performs local aggregation of the key-val pair from each mapper node and writes to the local disk.

Reducer:
1. Receives the key=date.tm_mon,date.tm_year and val= average value of transactions from all mapper nodes and preforms the aggregation at reducer node.

PART B. TOP TEN MOST POPULAR SERVICES (25%).
Job URL1:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1637317090236_32496/

Job URL2:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1637317090236_32613/

Output:

"0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444"
8.415510080995852e+25

"0xfa52274dd61e1643d2205169732f29114bc240b3"
4.578748448318587e+25

"0x7727e5113d1d161373623e5f49fd568b4f543a9e"
4.562062400135223e+25

"0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef"4.317035609226448e+25

"0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8"
2.706892158201811e+25

"0xbfc39b6f805a9e40e77291aff27aee3c96915bdd"
2.110419513809458e+25

"0xe94b04a0fed112f3664e45adb2b8915693dd5ff3"
1.5562398956801569e+25

"0xbb9bc244d798123fde783fcc1c72d3bb8c189413"
1.198360872920345e+25

"0xabbb6bebfa05aa13e908eaa492bd7a8343760477"
1.1706457177940784e+25

"0x341e790174e3a4d35b65fdc067b6b5634a61caea"
8.379000751917756e+24

Key:
Contract Numbers and Aggregated Value of Contract- sorted in decreasing order of aggregated value of contracts.

Method:
Mapper1:
1. Import MRJobs , MRSteps
2. Read two input files 'data/ethereum/transactions' and file 'data/ethereum/contracts' line by line and split them at every ',' .
3. 'If elif' condition passed to identify the to_address fields which acts as the unique identifier in both the two data sets for mapping.

'If' the dataset has totally 8 fields with 7 indices, then it is identified as 'transactions' data set and the value of transactions is aggregated with the key being the to_address and the value being the tuple (1,value) which gives the count of values for one unique transaction number and identifies as the 'transactions' dataset to the reducers in the subsequent steps.

'elif' the dataset is checked for a total field count of 6 with 5 indices, then it is identified as 'contracts' dataset and the value of transactions is aggregated with the key being the to_address and the value being 2 for the key-value pair and 1 as the counter for the reducers in the subsequent steps.

Reducer1:
1. Receives the key= to_address and val= (1,val) to check if the vsalue are valid, concludes as they exist.
2. Then the reducer sums all the values and key being the unique identifier, passes this value to the cascaded Mapper2.

Mapper2:
1. Receives the key-value pair from Reducer1 key= key= to_address and value= average value of transactions from Mapper2, combines all the keys and values, passed 'none' as the key for the reducer to find the Top 10 contracts by their value.

Reducer2: The value is sorted by decreasing order using lambda function then using the lambda function then using a 'for' loop the reducer extracts the Top 10 values of the contracts with their aggregated values.

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

Job URL1:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1637317090236_66099/

Job URL2:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1637317090236_66102/

Output:

"0x0000000000000000000000f56aa9716b898ec4"	23989401188
"0x829bd824b016326a401d083b33d092293333a830"	15010222714
"0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c"	13978859941

"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5"	10998145387
"0xb2930b35844a230f00e51431acae96fe543a0347"	7842595276
"0x2a65aca4d5fc5b5c859090a6c34d164135398226"	3628875680
"0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01"	1221833144
"0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb"	1152472379
"0x1e9939daaad6924ad004c2560e90804164900341"	1080301927
"0x61c808d82a3ac53231750dadc13c777b59310bd9"	692942577

Key:
miner: The address of the beneficiary to whom the mining rewards were given
size: The size of this block in bytes

Method:
Mapper1:
1. Import MRJobs , MRSteps
2. Read input file 'data/ethereum/blocks' line by line and split them at every ',' .
3. Checks if the total number of fields are 9 with 8 indices. The field indexed as 2 'miner' is passed as the key, with the integer value of the corresponding aggregated field indexed as 4 'size'.

Reducer1:
1. Receives the key= miner and val= size.
2. Aggregates the size in bytes each unique miner identification number.

Mapper2:
1. Read input files line by line, if split along '\t' and has 2 fields. We yield key(None) and values(miner, size)

Reducer2:
1. Using sort function all the pairs are sorted in descending order. Next, we iterate through the sorted values 10 times yielding them and then stop.

PART D. DATA EXPLORATION (40+%)
SCAM ANALYSIS

1.Popular Scams

Job URL:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_0230/

Output1:

"Scamming" 3.833616286244429e+22

"Fake ICO" 1.3564575668896302e+21

"Phishing" 2.6999375794087423e+22

"Scam" 0

Key:
category: Category of scam - Phishing, Ransomware, Trust Trade, etc
value: value of scam

Method:
Mapper1:
1. Import MRJobs , MRSteps
2. Read input file 'data/ethereum/scams.json' and 'data/ethereum/transactions' line by line and split them at every ',' .
3. Checks if the total number of fields are 7, then it is considered as transactions dataset, else scams dataset then Key= 'to_address' value= value of scam, 0 to identify dataset as transactions file.
4. Else we consider dataset as key= 'to_address' value= value of scam, 1 to identify dataset as scams.json file

Reducer1:
Checks if key-value pair is from transactions or scams dataset. Passes it on to Mapper2
Mapper2:
Passes the key-value pair from the reducer1 to reducer2 which is the category of scam and value.

Reducer2:
Aggregates the total value of for each category of scam

Observation: From the above statistics it is evident that "Scamming" is the most lucrative form of scams, with a value of 3.8 e^22 with "Phishing" being the second most lucrative form of scams at 2.69 e^22.

Job URL:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_0261/

Output2:

["Fake ICO", "Offline"] 1.35645756688963e+21

["Phishing", "Inactive"] 1.4886777707995027e+19

["Phishing", "Offline"] 2.245125123675149e+22

["Phishing", "Suspended"] 1.63990813e+18

["Scam", "Offline"] 0

["Scamming", "Active"] 2.20969523566792e+22

["Phishing", "Active"] 4.5315978714979385e+21

["Scamming", "Offline"] 1.6235500337815095e+22

["Scamming", "Suspended"] 3.71016795e+18

Key:
category: Category of scam - Phishing, Ransomware, Trust Trade, etc
status: If the scam is currently active, inactive or has been taken offline
value: volume of scam
Method:
Mapper1:
1. Import MRJobs , MRSteps
2. Read input file 'data/ethereum/scams.json' and 'data/ethereum/transactions' line by line and split them at every ',' .

3. Checks if the total number of fields are 7, then it is considered as transactions dataset, else scams dataset then Key= 'to_address' value='value' of scam, 1 to identify dataset as transactions file.
4. Else we consider dataset as key= 'to_address' value='category', 'status', 'volume' of scam, 2 to identify dataset as scams.json file

Reducer1:
Checks if key-value pair is from transactions or scams dataset. Passes it on to Mapper2
Mapper2:
Passes the key-value pair from the reducer1 to reducer2 which is the category of scam, status and value.

Reducer2:
Aggregates the total value of for each category of scam and its status.

Observation: From the aggregated value per scam category per status, it is evident that offline scamming is more lucrative due to lack of non-fungible tokens that can trace the malicious contracts. **The offline scams account to 4.0 e^22 units i.e. 59.9% of total scam value of 6.67 e^22 units including "phishing", "fakeICO", "scamming" and "scam".** Whereas the active, inactive and suspended status of scams contribute to 39.1%, 0.2% and 0.8% adding up to 40.1% equal to **2.66e^22 units.**

Statistically, "Offline Phishing" can be considered as most lucrative form of scamming with 2.25 e^22 units out of total 6.67 e^22 gas consumption i.e. 33.6% of the total value of scam, closely followed by "Active Scamming" at 33.1% and "Offline Scamming" at 24.3%.

MISCELLANEOUS ANALYSIS

1. Fork the Chain

URL:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1637317090236_24024/

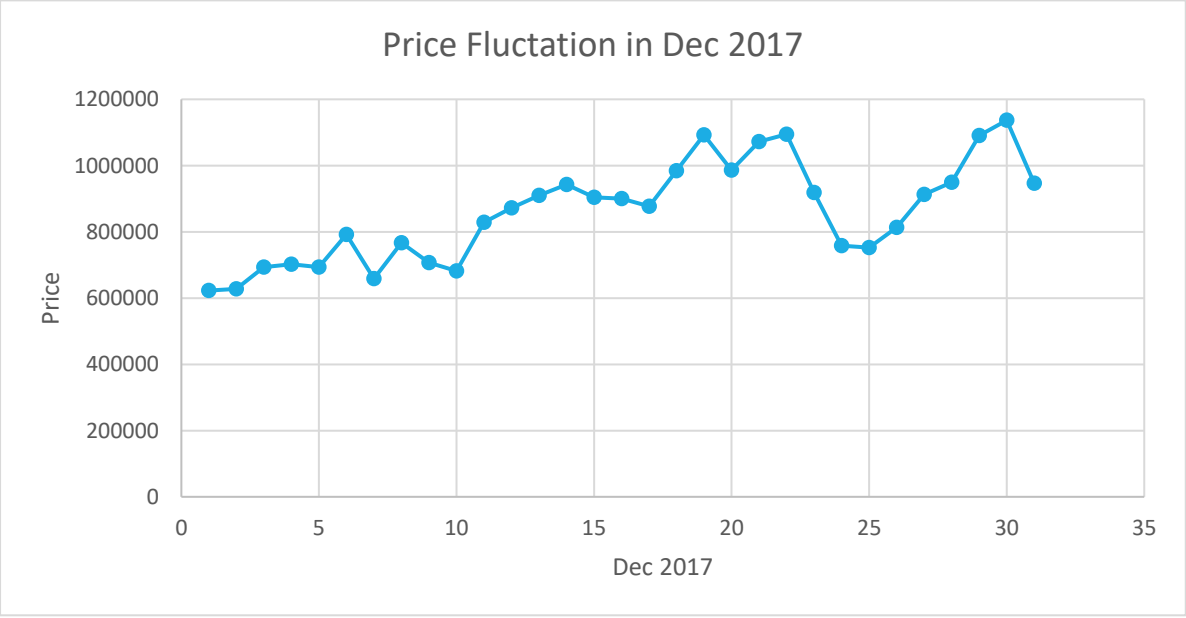
Fork: (date.tm_year== 2017 and date.tm_mon== 12)

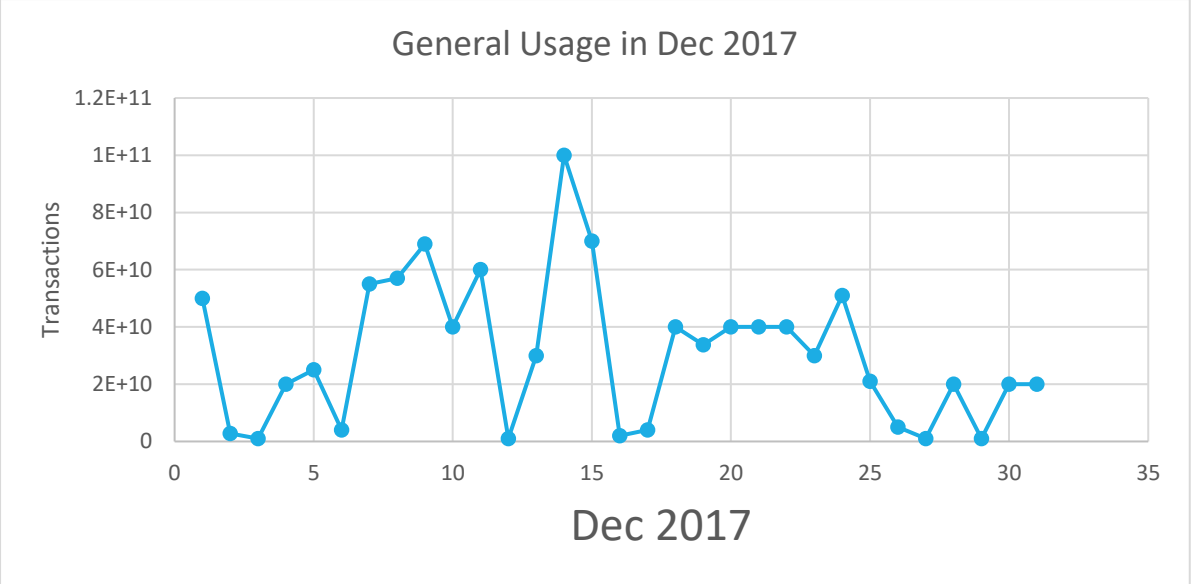
Method: We can identify two forks on Dec 14,15 2017, we chose this fork to show the combined impact of the forks on the price and gas usage.

Using MRjobs and input data as transactions dataset we obtain the price fluctuation and general consumption in Dec 2017 for the entire month to identify the changes before and after the fork.

Mapper: Reads transactions dataset, line by line and splits the data by ‘,’. Timestamp converted to time. Yield the key as day_of_month and value as price if the (date.tm_year== 2017 and date.tm_mon== 12).

Reducer: Aggregates the count of transactions and value of transactions during one month where a fork occurs.





Output:

1 [622720, 50000000000.0]

10 [681677, 40000000000.0]

12 [872340, 1000000000.0]

14 [942559, 100000000000.0]

16 [899857, 2000000000.0]

18 [984021, 40000000000.0]

21 [1072665, 40000000000.0]

23 [918881, 30000000000.0]

25 [752361, 21000000000.0]

27 [912752, 1000010001.0]

29 [1090191, 1000000000.0]

3 [693808, 1000000000.0]

30 [1136659, 20000000000.0]

5 [693090, 25000000000.0]

7 [659068, 55000000000.0]

9 [707223, 69000000000.0]

11 [828899, 60000000000.0]

13 [909631, 30000000000.0]

15 [904346, 70000000000.0]

17 [876574, 4000000000.0]

19 [1092234, 33800000000.0]

2 [627785, 2800008456.0]

20 [986104, 40000000000.0]

22 [1094028, 40000000000.0]

24 [758234, 51000000000.0]

26 [813510, 5000000000.0]

28 [949503, 20000000000.0]

31 [946981, 20000000000.0]

4 [701834, 20000000000.0]

6 [791746, 4000000000.0]

8 [766411, 57000000000.0]

Observation:

In the above graph, we can observe the price and count in transactions before, during and after a fork in Dec 2017. We can clearly observe a surge in the number of transactions on the fork days that is 14th , 15th Dec 2017 in comparison to the previous days and sharply falls on the next day. We see the fall in prices on the 14th and 15th Dec with quick recovery on the subsequent days. Thus, we can conclude that forking does impact the transaction average price directly and general usage inversely.

2. Gas Guzzlers:

Method: We need to analyse the trends in gas price, complexity of contracts, correlation between gas consumption and transactions for the top 10 contracts.

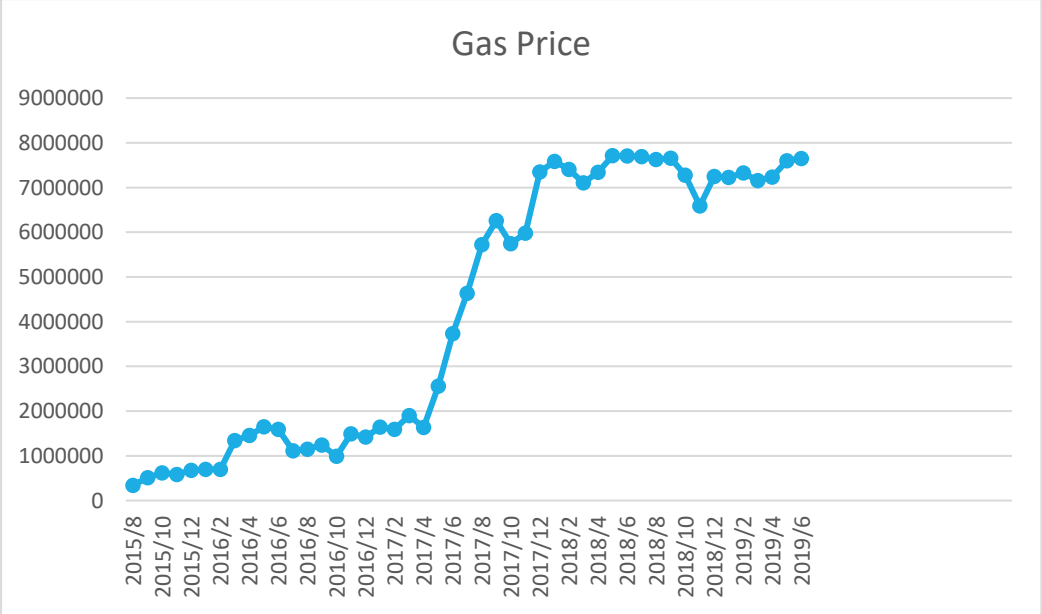
First we use MapReduce for the analysis of change in gas price and complexity over time. We perform a check on the number of fields the dataset has if its 7 it is considered as 'transactions', if 5 then ' contracts' else it is taken as 'blocks'. We calculate the average 'gas_price' field= [5] per month over the years 'timestamp'=field[6] as one part of the value, the other aggregates the 'difficulty' field =[3] of the contracts, key being the 'timestamp' field =[7] in blocks dataset.

The subsequent mappers and reducers aggregate the total gas price and complexity for the month per year.

We then perform write a pyspark code and run the three datasets to procure the gas price and transactions for the top 10 contracts.

Gas1: Change of Gas Price over time

JobURL:http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_6288/



Observation: Gas Price has risen from 2015 to 2019 from 0.5 to 8 million units, with major surge in price during April to August 2017,

with 4.3 times increase in the price during a period of 5 months. Overall, the gas prices have gradually increased with time.

Output:

["2015", "09"] [511130.0238095238, 6575007525031.816]

["2015", "10"] [617120.9283707865, 6351912456529.025]

["2015", "12"] [676801.2943100065, 8280026429333.4795]

["2016", "02"] [699891.4884663342, 12805679682947.633]

["2016", "04"] [1453811.85509839, 28151713317813.887]

["2016", "06"] [1590509.3207848598, 49493950288416.99]

["2016", "08"] [1145195.6850787767, 59338119960680.43]

["2016", "11"] [1488459.4490766649, 68392929882483.93]

["2017", "01"] [1645004.5666060764, 101441916721288.06]

["2017", "03"] [1900062.4476502456, 198132058062115.0]

["2017", "05"] [2555896.5429241275, 427943193614743.8]

["2017", "07"] [4635260.841524708, 1188209445191042.5]

["2017", "09"] [6259570.235479398, 2472615368389359.0]

["2017", "10"] [5744839.77941408, 1982205861334579.8]

["2017", "12"] [7348227.023722457, 1715527999392166.0]

["2018", "02"] [7408453.419146736, 2904709176436539.0]

["2018", "04"] [7339641.553098808, 3152081258617400.0]

["2018", "06"] [7704112.409546432, 3312525280779383.5]

["2018", "08"] [7626441.449449848, 3524781455793420.5]

["2018", "11"] [6591821.474788537, 2867483299389692.0]

["2019", "01"] [7227507.102983546, 2647277315508496.5]

["2019", "03"] [7153174.262677149, 1861662833720996.0]

["2019", "05"] [7600090.315332367, 2006025542803913.8]

["2015", "08"] [342633.8565400844, 4001527822780.391]

["2015", "11"] [583187.0693153001, 7770939637944.217]

["2016", "01"] [695822.0702730031, 9504433076575.49]

["2016", "03"] [1344195.8812997348, 19503022691379.637]

["2016", "05"] [1649703.1031405525, 38924813929437.58]

["2016", "07"] [1109384.0947795825, 57858382394762.78]

["2016", "09"] [1239754.4080812854, 72708751309632.16]

["2016", "10"] [989291.0377268059, 88084306677558.62]

["2016", "12"] [1420966.420770878, 81620432517653.64]

["2017", "02"] [1594969.164809835, 130438449525581.77]

["2017", "04"] [1636672.635781005, 278467790990791.1]

["2017", "06"] [3730226.7347838604, 748864389603084.8]

["2017", "08"] [5724477.568567611, 1815109248848101.5]

["2017", "11"] [5984028.924169609, 1465338813397301.2]

["2018", "01"] [7583611.642400637, 2250010920771103.5]

["2018", "03"] [7107875.709015946, 3201791840962772.0]

["2018", "05"] [7710297.813332786, 3260816559034257.5]

["2018", "07"] [7691812.627910354, 3454264351592171.5]

["2018", "09"] [7652196.720304907, 3255259820702298.5]

["2018", "10"] [7274765.667196118, 3187277567051765.0]

["2018", "12"] [7246771.347853503, 2354652049889461.5]

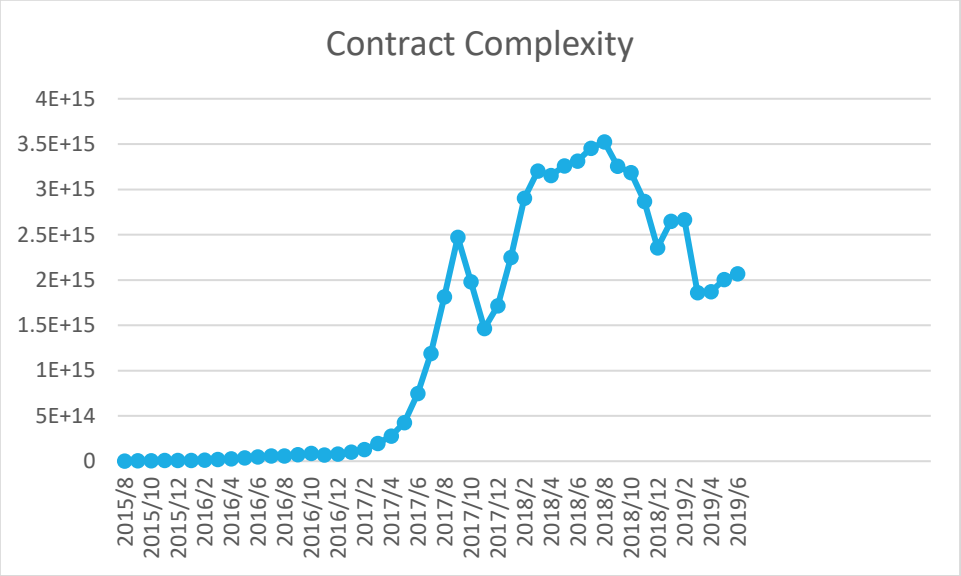
["2019", "02"] [7325267.441476677, 2666442177822670.0]

["2019", "04"] [7231753.62817928, 1870018219496794.0]

["2019", "06"] [7646949.620029971, 2068085609513129.2]

Gas2: Contract Complexity Fluctuation over time

Job URL:http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_1535/

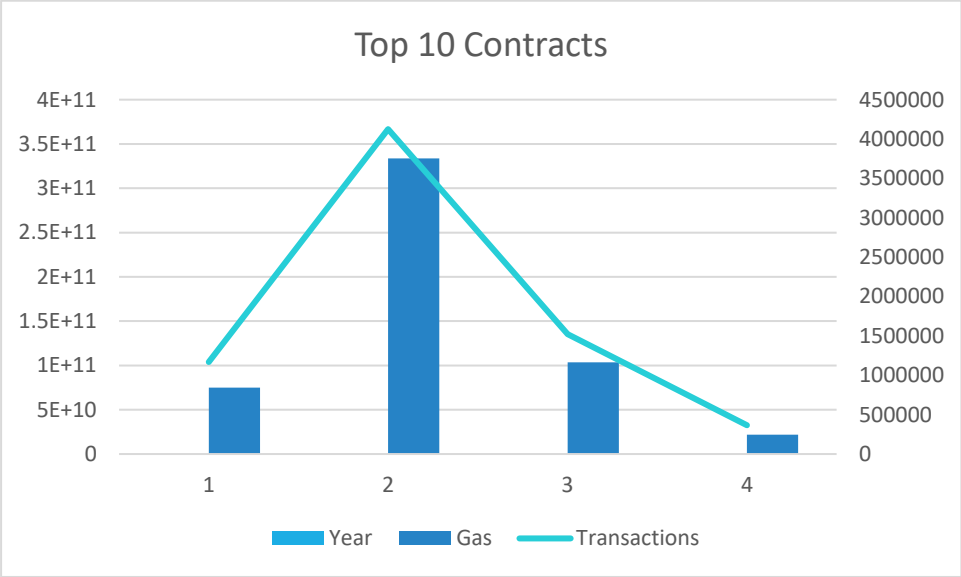


Observation:

The difficulty or complexity of contracts has drastically increased between April 2017 to August 2017, almost 80% surge in complexity within 5 months. There is a sudden dip in complexity during December 2017, and then a gradual increase until October 2018 and then fluctuations. **We can say that overall, the trend suggests that contracts are becoming more complex with time.**

Gas3: Trends of Gas and Transactions in Top 10 contracts over (2016-2019)

Job URL:http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_1168



Observation: The overall gas consumption is correlated to the rate of transactions. With increase in transactions per year, the average gas consumed increases, the contracts tend to be more complex, hence there is a direct correlation to the transactions, gas consumption and complexity.

Output:

('Address', 'Year', 'Total Gas', 'Total number')

0xfa52274dd61e1643d2205169732f29114bc240b3,2019,6339130000,181118

0xfa52274dd61e1643d2205169732f29114bc240b3,2016,4302626266,122931

0xe94b04a0fed112f3664e45adb2b8915693dd5ff3,2017,153063053121,1135600

0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444,2018,4117528937,68618

0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8,2017,5290233312,112539

0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef,2017,53467167552,1272151

0xabbb6bebfa05aa13e908eaa492bd7a8343760477,2016,642102467,6413

0xe94b04a0fed112f3664e45adb2b8915693dd5ff3,2018,47504180297,328935

0xabbb6bebfa05aa13e908eaa492bd7a8343760477,2018,15993613432,234160

0xfa52274dd61e1643d2205169732f29114bc240b3,2017,21801690000,622905

0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444,2019,580212,8

t,e,266905745,266905745

0xabbb6bebfa05aa13e908eaa492bd7a8343760477,2017,50257842481,415655

0x341e790174e3a4d35b65fdc067b6b5634a61caea,2016,8350972,42

0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8,2018,6359956049,135296

0xbb9bc244d798123fde783fcc1c72d3bb8c189413,2017,17871078,180

0xe94b04a0fed112f3664e45adb2b8915693dd5ff3,2019,6876600000,45844

0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444,2016,18230724414,183724

0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef,2018,9614776129,192717

0xabbb6bebfa05aa13e908eaa492bd7a8343760477,2019,4392548000,54230

0x7727e5113d1d161373623e5f49fd568b4f543a9e,2017,45264887181,497064

0xbb9bc244d798123fde783fcc1c72d3bb8c189413,2019,50000,1

0xfa52274dd61e1643d2205169732f29114bc240b3,2018,19750197130,564291

0xbfc39b6f805a9e40e77291aff27aee3c96915bdd,2016,14481700000,362046

0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8,2019,1561370720,33215

0xbb9bc244d798123fde783fcc1c72d3bb8c189413,2016,9976458418,65778

0xe94b04a0fed112f3664e45adb2b8915693dd5ff3,2016,14153448629,106212

0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444,2017,4409845975,69523

0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8,2016,412342127,8769

0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef,2019,2572995022,51455

0x7727e5113d1d161373623e5f49fd568b4f543a9e,2016,814811141,15938

0xbb9bc244d798123fde783fcc1c72d3bb8c189413,2018,1191274,12

0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef,2016,11800348083,294997

3. Comparative Evaluation:

Method:

- Implementing the top 10 contracts using spark code. We make use of 'transactions' and 'contracts' dataset similar to what was done in PartB using MapReduce jobs.
- Import pyspark module.
- Check condition if length of fields is not equal to 7 then return false for 'transactions' dataset, if length is not equal to 5 then return false for 'contracts' dataset.
- Load the datasets and check the fields with the above conditions.
- Map Stage: Key: 'to address', Value= value of the transaction/contract.
- Reduce Stage: with key being the common the values are aggregated.
- 'Contracts' dataset the address is mapped as key.
- Join the address and aggregate values.
- Sort the in descending order the top 10 contracts and their values.

Output:

0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444:84155100809965865822726776

0xfa52274dd61e1643d2205169732f29114bc240b3:45787484483189352986478805

0x7727e5113d1d161373623e5f49fd568b4f543a9e:45620624001350712557268573

0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef:4317035609226246891929896 9

0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8:270689215820195424 99882877

0xbfc39b6f805a9e40e77291aff27aee3c96915bdd:211041951380936600 50000000

0xe94b04a0fed112f3664e45adb2b8915693dd5ff3:15562398956802112 254719409

0xbb9bc244d798123fde783fcc1c72d3bb8c189413:11983608729202893846818681

0xabbb6bebfa05aa13e908eaa492bd7a8343760477:11706457177940895521770404

0x341e790174e3a4d35b65fdc067b6b5634a61caea:8379000751917755624057500

Hadoop:

Iteration1:

Time Recorded: 48 mins 02 sec

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_0326/

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_0401/

Iteration2:

Time Recorded: 49 mins 19sec

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_0332/

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_0415/

Iteration3:

Time Recorded: 50 mins 43sec

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_0232/

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_0319/

Average time: 49.35 Mins

Median time: 47.30 Mins

Spark:

Iteration1: http://itl111.student.eecs.qmul.ac.uk:4040/

Time: 4 min 03 sec

Iteration2: http://itl111.student.eecs.qmul.ac.uk:4040/

Time: 3 min 48 sec

Iteration3: http://itl111.student.eecs.qmul.ac.uk:4040/

Time: 3 min 35 sec

Average Time: 3.78 Mins

Median Time: 3.75 Mins

Observation:

The distinguishing factors between Spark and MapReduce is that Spark processes as well as retains data in memory for subsequent steps, in case of MapReduce processes data on disk. Thus, for smaller workloads, the data processing speeds are up to 100 times faster for Spark than MapReduce. Since we are dealing with considerable amount of data, we can consider it as BigData. We can clearly see the Average and Median time to run the same code in MapReduce takes **49.35 Mins** and Spark takes about **3.78 Mins. We could conclude that Spark is more suitable for joining multiple Big datasets and processing them at a rate atleast 10-15 times faster than MapReduce.**