Part A. Time Analysis:

Number of transactions (total for each month):

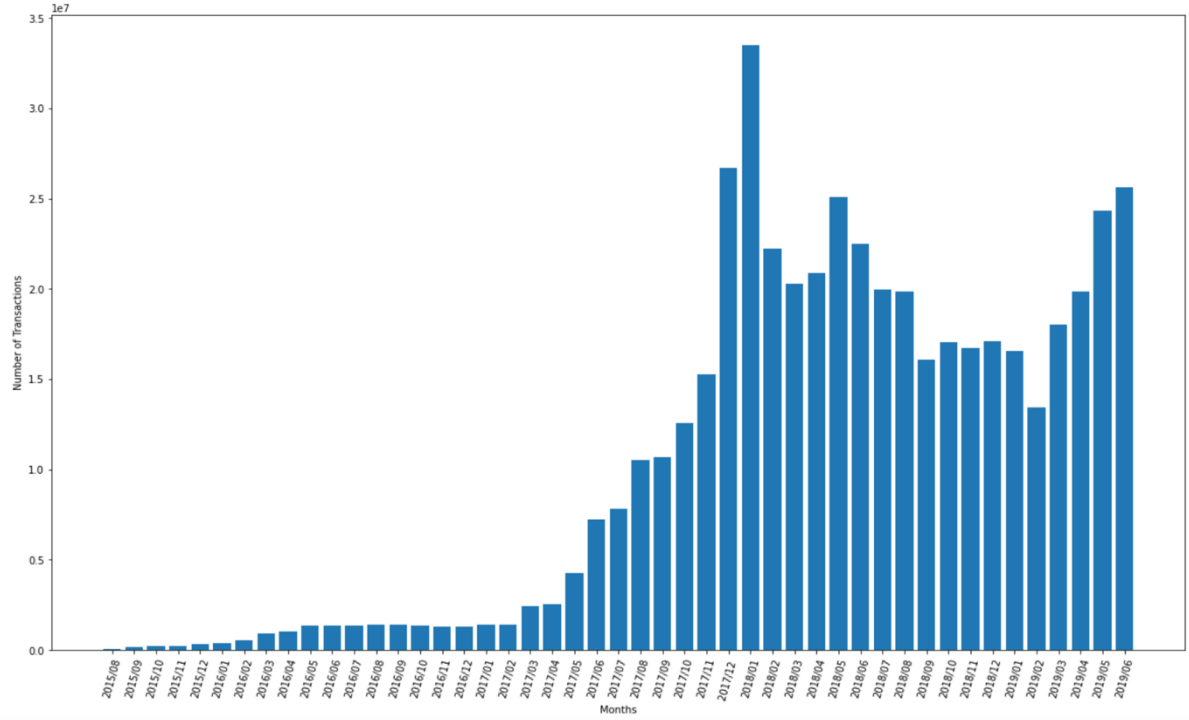


Figure1: Total Number of transactions for each month (from Aug/2015 to June 2019)

In this task, we need to know the total of transactions in each month (from Aug/2015 to June 2019).

Spark has been used. Firstly, good_line() function is used to filter the line, which just needs the line has seven lengths. Then, sc.textFile() from pyspark.SparkContext() has been used to read transactions data. The filter() operation is applied to each line independently. Thus this operation will use good_line() function to check if each line has seven lengths. Furthermore, map()operation like filter() that work independently, but it will used time.strftime() to change time in block timestamp (in index 6) to formal date (month, year) after get timestamp by using time.gmtime(). In the same step, one will be put for each transaction, which will help to collect the sum of transactions for each month in the next step. The reduceByKey() operation deals with multi partitions, so it is used to collect the total number of transactions by using lambda(). Moreover, sortBy() will order data by date. Finally, For-loop is used to print data which is the date in record[0] and the total in record[1].

The output has been plotted by a bar chart, as shown in figure 1. According to this task result (see figure1), the peak was in January 2018, whereas the lowest point happened in Aug/2015. In general, from the first period until the peak point (January 2018), using transactions increased, but there are some points that slightly decreased from narrow perspective. After that, the number of transactions fluctuated.

Average value of transactions:

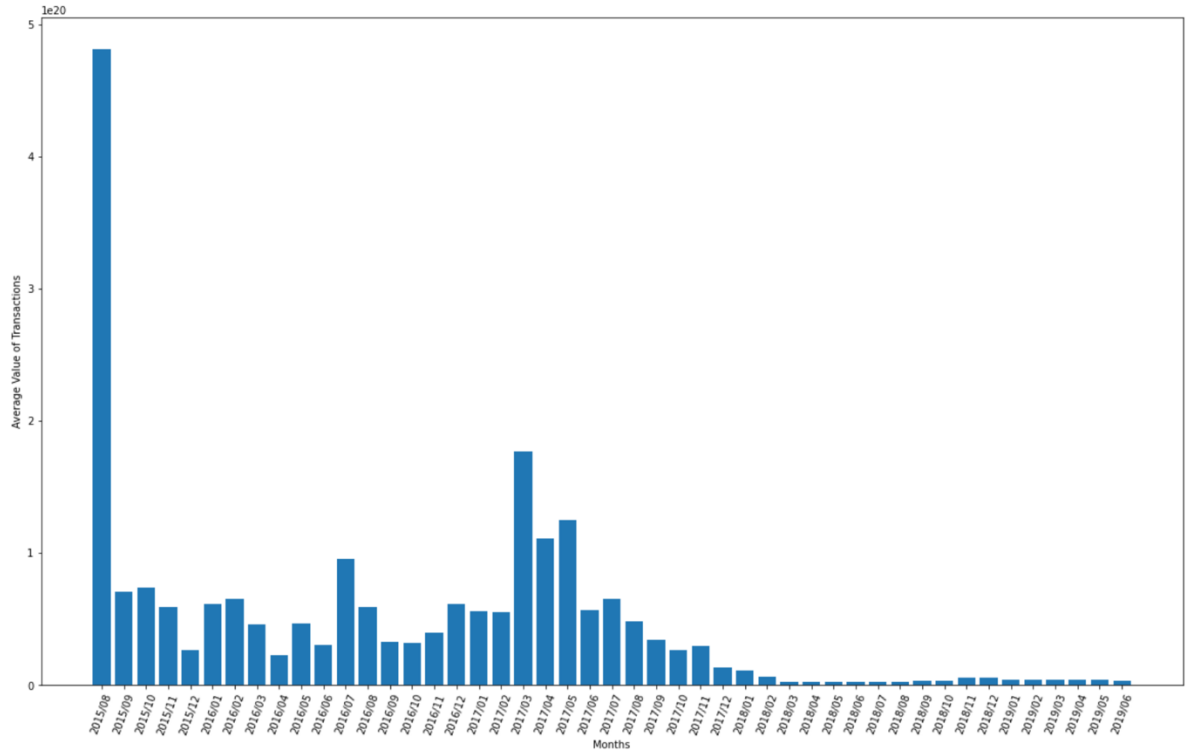


Figure 2: Average value of transactions

Spark has been used. Firstly, good_line() function used to filter the line, which just needs the line has 7 length. Second, sc.textFile() from pyspark.SparkContext() has been used to read transactions data. The filter() operation is applied to each line independently. Thus this operation will use good_line() function to check if each line has seven lengths. Furthermore, map()operation like filter() that work independently, but it will used time.strftime() to change time in block timestamp (in index 6) to formal date (month, year) after get timestamp by using time.gmtime(). In the same step, the value column, that takes place in the index [3], will be used to find the average for each month in the next step. The groupByKey() operation is used to shuffle all the value key. Then, mapValues() operation will use lambda to calculate the average for each month, which is the sum of the values divided by the length of the values for each month. Moreover, sortBy() will order data by date. Finally, For-loop is used to print data which is the date in record[0] and the total in record[1].

The output has been plotted by bar chart as shown in figure 2. According to this task result (see figure 2), the peak was in Aug/2015, which is totally different from figure 1, while the lower points were from March/2018 to Aug/2018. In general, the values fluctuated in the whole period, and the last half period is significantly decreasing compared with the first period.

Part B. Top Ten Most Popular Services:

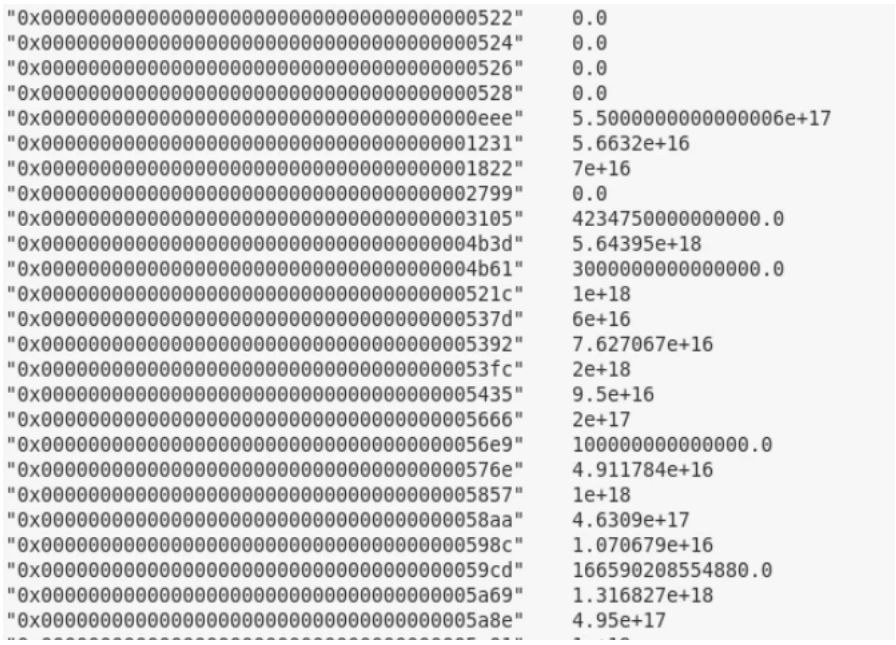
This part is solved by three jobs independently, and the output for each job will be input for the next job.

Job 1 - Initial Aggregation:

This step will aggregate two fields in the transactions dataset, which is the value and to_address field. The mapper deals with each line independently. Inside the mapper, split() has been used to divide each line. Then, If-conditional is used to check the length of each line, that if the length of the line equals 7, the mapper will yield the to_address and value data, which will be in index 2 and 3**,** respectively.

The reducer will work on the mapper's output data, which will sum the values for each key (to_address) by using a for-loop to collect values in the array and then sum it. Finally, this step will yield each address with aggregation of its values, that is output for this job.

The output of this job is 53296857 records; therefore, it is difficult to attach it; see figure 3, which is a sample of Job 1 output.



```
"0x0000000000000000000000000000000000000522"    0.0
"0x0000000000000000000000000000000000000524"    0.0
"0x0000000000000000000000000000000000000526"    0.0
"0x0000000000000000000000000000000000000528"    0.0
"0x000000000000000000000000000000000000eee"     5.5000000000000006e+17
"0x0000000000000000000000000000000000001231"    5.6632e+16
"0x0000000000000000000000000000000000001822"    7e+16
"0x0000000000000000000000000000000000002799"    0.0
"0x0000000000000000000000000000000000003105"    4234750000000000.0
"0x0000000000000000000000000000000000004b3d"    5.64395e+18
"0x0000000000000000000000000000000000004b61"    3000000000000000.0
"0x000000000000000000000000000000000000521c"    1e+18
"0x000000000000000000000000000000000000537d"    6e+16
"0x0000000000000000000000000000000000005392"    7.627067e+16
"0x000000000000000000000000000000000000053fc"    2e+18
"0x0000000000000000000000000000000000005435"    9.5e+16
"0x0000000000000000000000000000000000005666"    2e+17
"0x00000000000000000000000000000000000056e9"    100000000000000.0
"0x000000000000000000000000000000000000576e"    4.911784e+16
"0x0000000000000000000000000000000000005857"    1e+18
"0x00000000000000000000000000000000000058aa"    4.6309e+17
"0x000000000000000000000000000000000000598c"    1.070679e+16
"0x00000000000000000000000000000000000059cd"    166590208554880.0
"0x0000000000000000000000000000000000005a69"    1.316827e+18
"0x0000000000000000000000000000000000005a8e"    4.95e+17
```

Figure 3: Aggregation sample output for Job 1

Job 2 - Joining transactions/contracts and filtering:
The url to track the job:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_3387/

This job will join the job one output with the contracts dataset. In mapper, if condition will select whether the line from contracts or job one output dataset. If the length is equal five, that means the line comes from contracts, so the address (index 0) and block_number (index 3) are yield and number 2. Otherwise, the length equals two, which means the line comes from job one output; thus, address (index 0), values (index 1), and number 1 are yield.

In reducer, for-loop has been used to collect values for each key (address). Inside for-loop, if statement will be used to determine whether value comes from which data; if index 1 has number 1, that means from job one output, but if it has number 2, that means from contracts. The contracts values will append in an empty array, and Job 1 output will assign to a new variable. Finally, after finishing the for-loop, the if statement will check the new variable for Job 1 output and array length for contracts. Therefore, both don't equal None or 0, respectively. The reducer will yield the key (address) and job one value. The output of this job is 2243685; therefore, it's difficult to attach it. See figure 4, which is a sample from Job2 output.

```
"0x7539e64961c2a95a6d83592d50fea99e2413d983"    "3.21245018e+18"
"0x7539ede4251a0b448a7a339e721fae03cfcc75b9"    "1.25360255e+17"
"0x7539f484c6bf1c367b1f9a0907d0da96d18fb0df"    "3.70112608e+18"
"0x7539ff2d40f8bdf90e500cea746f079df652463b"    "0.0"
"0x753a0da9740fd4cfe9e1e6da6061e9922c429caa"    "2.91937478e+18"
"0x753a23561cc125b1a726b33c1a2721064b1dbefb"    "0.0"
"0x753a27e5a1cecf845ae3ec3e85cf1f80fe302455"    "0.0"
"0x753a29a0a0d47258ffd2557f9c9b66969c668d7d"    "7.1877952e+17"
"0x753a2aad86a2262f32d4bccfbd959ddd4b2f17ae"    "0.0"
"0x753a2e86d4028503807bf8f262bf65339894c429"    "0.0"
"0x753a3773c93867cb43d9ea7cc3aefedaea1366f0"    "0.0"
"0x753a3bd6153b9d44b47617552e51963dc2e88260"    "0.0"
"0x753a495ec77ba04c140117fbce8165329672fa07"    "0.0"
"0x753a5471b4669e777ae3ab9f552a4a937ae8e136"    "8.085163687368808e+18"
"0x753a8c1a7d761f7460d910f738a69add95ccbab8"    "0.0"
"0x753a9298475bf17f2723d8401ae8bc29f6cd73f3"    "9.3927181e+17"
"0x753a98e2c1ec5c172a21989cea7d8a4f34474125"    "9.5e+17"
"0x753a9cbecf5102ab11d46234e174b3732b61dde1"    "3.58e+18"
"0x753aaa2069a54edff6619a884ecdd9c4f60820ae"    "0.0"
"0x753ac1def1060ba64984d045cca1ea86c05caf35"    "0.0"
"0x753ad7e5582efb5746ed33d7c7795f5d275887ef"    "8.188835140783558e+18"
"0x753aee9e2899d38b5c31e8679890666502519d0d"    "3.725700895223818e+19"
"0x753af0d74a4bcfacec3d6176c39f91e7ec1df3c9"    "5.70969463e+18"
"0x753b2eaa2e2a8e4ba90bd4edf11ea933135d171d"    "0.0"
"0x753b3ad5c79a9d69d75e955993077949 8eea8751"    "0.0"
"0x753b41ca3a89a0540971b13351e700a99143f0a8"    "0.0"
"0x753b45be8e5d913c8f259bba36fc780ce45f4126"    "0.0"
```

Figure 4: Joining sample output for Job 2

Job 3 - Top Ten:

The final job will be getting the top ten addresses with aggregation values. In mapper, if statement will check whether the length of the line equals two, that line will come from the output of job 2. Thus, the mapper will yield address and aggregation, which take place in index 0 and 1, respectively, after strip double quotes and slash symbols.

The combiner() has been used to make processing faster. Inside the combiner, sorted() function is used to order values by using the lambda function. Moreover, a new variable has been assigned, named i and assigned by 0. Finally, for-loop has been used to yield value, which content address and aggregation. So, i variable will increase by 1 (inside for-loop) until reaching 10 (using the if statement to check), which will help to count and print the top 10.

The reducer has the same combiner steps. Figure 5 displays the top ten addresses and aggregation values for the most popular services.

```
"0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444"    8.415510080996048e+25
"0xfa52274dd61e1643d2205169732f29114bc240b3"    4.5787484483186015e+25
"0x7727e5113d1d161373623e5f49fd568b4f543a9e"    4.562062400135001e+25
"0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef"    4.3170356092262535e+25
"0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8"    2.706892158201942e+25
"0xbfc39b6f805a9e40e77291aff27aee3c96915bdd"    2.1104195138094395e+25
"0xe94b04a0fed112f3664e45adb2b8915693dd5ff3"    1.5562398956803886e+25
"0xbb9bc244d798123fde783fcc1c72d3bb8c189413"    1.1983608729202686e+25
"0xabbb6bebfa05aa13e908eaa492bd7a8343760477"    1.1706457177940745e+25
"0x341e790174e3a4d35b65fdc067b6b5634a61caea"    8.379000751917755e+24
```

Figure 5: Top ten most popular services

Part C. Top Ten Most Active Miners:

This part is split into two jobs. The first job will aggregate the size of the miner in the blocks dataset.

In mapper, if statement will check about line length, which each line equals nine will be true. Thus, miner (index 2) and size (index 4) will yield. The combiner() has been used to make processing faster. The combiner and reducer have the same code, which will yield each key with the sum of its values.

The output of this job is 4826; therefore, it's difficult to attach it; see figure 6, which is a sample from job1 production.

```
"0x000083ceb2317f5755be7a745e3c4be7ba396877"    2362.0
"0x0000d91c20a5e7539eab3e52f73b41f450361aba"    545.0
"0x000354c8e89af5a4ea0be126176a14848678a391"    2197.0
"0x000a4e0c3fcfd7254a32f07d6b4e633e4f17c02d"    600113.0
"0x000a8076834dbc2edca1d7692775b6eeca08ee77"    3382.0
"0x000b800ab6000bc8e5c02423723bbb168720530a"    311949.0
"0x0012243161978464e7a354b08acc21fad187a3b5"    12460.0
"0x0017cf30c39f3ca48316d4874a5cb0f2ba2ed18b"    999.0
"0x0019405887ce7cdf99075c2fa41dae0dda1ea596"    1863947.0
"0x0021c4d83c714510c0cde78e68a739a978b63a63"    2541.0
"0x0025571697c62d4fe90b06dda1ceae7c6f31df0d"    75440.0
"0x0027bcf5801c9d21801f8089a926643da06eeb3a"    1316.0
"0x002880a8c9e859e4f3f1b6e7cef007c1c4ef5a0c"    2509.0
"0x002a0f8b3d5d866e3ceafb904e7f48741c43026b"    994.0
"0x002e08000acbbae2155fab7ac01929564949070d"    197850542.0
"0x002f0c43306a07ee407859b3730e2c273434849f"    10163.0
"0x0037a6b811ffeb6e072da21179d11b1406371c63"    47057.0
"0x0037ce3d4b7f8729c8607d8d0248252be68202c0"    1236984.0
```

Figure 6: output sample of Job 1

The second job will input the output of the first job. In the mapper, if the statement will check whether the length of the line equals two, so, the mapper will yield address and aggregation.

The combiner() has been used to make processing faster. Inside combiner, sorted() function is used to order values by using the lambda function. Moreover, a new variable has been assigned, named i and assigned by 0. Finally, for-loop has been used to yield value, which contents address and aggregation. So, i variable will increase by 1 (inside for-loop) until reaching 10 (using the if statement to check), which will help to count and print the top 10.

The reducer has the same combiner steps. Figure 7 displays the top ten addresses and aggregation values.

```
"0xea674fdde714fd979de3edf0f56aa9716b898ec8"    23989401188.0
"0x829bd824b016326a401d083b33d092293333a830"    15010222714.0
"0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c"    13978859941.0
"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5"    10998145387.0
"0xb2930b35844a230f00e51431acae96fe543a0347"    7842595276.0
"0x2a65aca4d5fc5b5c859090a6c34d164135398226"    3628875680.0
"0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01"    1221833144.0
"0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb"    1152472379.0
"0x1e9939daaad6924ad004c2560e90804164900341"    1080301927.0
"0x61c808d82a3ac53231750dadc13c777b59310bd9"    692942577.0
```

Figure 7: Top ten Most Active Miners

Part D. Data exploration:

A. Scam Analysis (Popular Scams):

The scams dataset is provided in JSON format (see figure 8), so the scams file has been converted to CSV format (see figure 9), by creating josnTocvs.py file after coping to the local desktop. Inside josnTocvs.py file, reading the JSON file is the first step, then the loads() function will use to load the data. Secondly, a new CSV file will create and open to write on it. The nested loop has been used to select specific columns. Addresses, category and status have been determined and written on CSV file and separated by /.

Figure 8: Sample of scams dataset by json format

	A
1	0x00e01a648ff41346cdeb873182383333d2184dd1/Phishing/Offline
2	0x858457daa7e087ad74cdeeceab8419079bc2ca03/Phishing/Active
3	0x4cdc1cba0aeb5539f2e0ba158281e67e0e54a9b1/Phishing/Offline
4	0x11c058c3efbf53939fb6872b09a2b5cf2410a1e2c3f3c867664e43a626d878c0/Phishing/Offline
5	0x2dfe2e0522cc1f050edcc7a05213bb55bbb36884ec9468fc39eccc013c65b5e4/Phishing/Offline
6	0x1c6e3348a7ea72ffe6a384e51bd1f36ac1bcb4264f461889a318a3bb2251bf19/Phishing/Offline
7	0x11c058c3efbf53939fb6872b09a2b5cf2410a1e2c3f3c867664e43a626d878c0/Phishing/Offline
8	0x2dfe2e0522cc1f050edcc7a05213bb55bbb36884ec9468fc39eccc013c65b5e4/Phishing/Offline
9	0x1c6e3348a7ea72ffe6a384e51bd1f36ac1bcb4264f461889a318a3bb2251bf19/Phishing/Offline
10	0x11c058c3efbf53939fb6872b09a2b5cf2410a1e2c3f3c867664e43a626d878c0/Phishing/Offline
11	0x2dfe2e0522cc1f050edcc7a05213bb55bbb36884ec9468fc39eccc013c65b5e4/Phishing/Offline
12	0x1c6e3348a7ea72ffe6a384e51bd1f36ac1bcb4264f461889a318a3bb2251bf19/Phishing/Offline
13	0x2268751eafc860781074d25f4bd10ded480310b9/Fake ICO/Offline
14	0xd0cc2b24980cbcca47ef755da88b220a82291407/Phishing/Offline
15	0x379ce20c018fb6301c1872c429ec7270ffa4dc5b/Phishing/Offline
16	0xaa1886de3f70a3ef502ea1379a311c5b4e05f3fd/Phishing/Offline
17	0xba83e9ce38b10522e3d6061a12779b7526839eda/Phishing/Offline
18	0xf73ef0650415656b51be39bd5d37cf5ce03f4033/Phishing/Offline
19	0xedf202629bb7e9f72d4c62c325d198513fa7a3d3/Phishing/Offline
20	0x42c5459911ae51d1d005cbe39749bd8d8e533c22/Phishing/Offline
21	0xcdd1c19dcc3f473eaef6edb0a28e1e796d6e1767/Phishing/Offline

Figure 9: Sample of scams dataset by CSV format

- What is the most lucrative form of scam?
  To answer this question, three MapReduce have been used with MYStep to order each step.
  The first mapper and reducer (step 1) will join the datasets. It uses if statement to determine whether the line comes from the scams CSV file or transactions dataset in mapper. If the line comes from scams CSV, that means the line's length is equal 3; thus, address (index 0), category (index 1), and number 1 will be yielded (each category and one on a tuple). Otherwise, the line's length equals 7, which means the line comes from the transactions dataset. The to_address (index 2), value (index 3), and number 2 will be yielded (each value and number 2 on a tuple).
  In reducer, two variables have been assigned, which are val (value) equal 0 and cat (category) equal None. For-loop has been used to look inside values, which might contain a category with number 1 or value with number 2. Inside the loop, if statement has been used to check about values. If index 1 in values equals one that assigns category (index 0) to cat variable. Otherwise, index one will be equal to 2, which will sum value (index 0). Finally, after the for-loop finish,
  If statement will check about cat and val have values not empty. The cat and val variables will yield if both have values.

The second mapper and reducer (step 2) will sum values. The mapper will yield the input without any change. The reducer will sum values for each key (category).
The third mapper and reducer (step 3) will rank the categories by their values. In mapper, it will yield None, cat (category), and values after putting cat and values on one tuple. In reducer, sorted() function has been used to order the values by using lambda. The for-loop will look on sorted_values, which has all values after sorted. Inside for-loop, it will yield value[0] value[1], which value[0] has category and value[1] has aggregation value.

```
"Scamming"      3.833616286244444e+22
"Phishing"      2.699937579408742e+22
"Fake ICO"      1.35645756688963e+21
```

Figure 10: most lucrative form of scam

According to output (see figure 10), it can be seen that the scamming category is most lucrative then phishing, and Fake ICO is less lucrative.

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

To answer this question, three MapReducer have been used with MYStep to order each step.
The first mapper and reducer (step 1) will join the datasets. It uses if statement to determine whether the line comes from the scams CSV file or transactions dataset in mapper. If the line comes from scams CSV, that means the line's length is equal 3; thus, the category (index 1) and state (index 2) are assigned to the cat_sta variable. Finally, address (index 0), cat_sta variable, and number 1 will be yielded (each cat_sta variable and 1 on a tuple). Otherwise, the line's length equals 7, which means the line comes from the transactions dataset. The to_address (index 2), value (index 3), and number two will be yielded (each value and twice number 2 on a tuple).

In reducer, two variables have been assigned, which empty array named val (value), new_cat_stat (category and state) equal None. The for-loop has been used to look inside values, which might contain cat_sta (category and state) and number 1 in one tuple or value with number 2. Inside the loop, if statement has been used to check about values. If index 1 in values equal 1 that assign new_cat_stat variable to index 0 in value. Otherwise, index 2 will be equal to 2, which will append index 0 in value to val array. Finally, after the for-loop finish, if statement will check about the length of val array and value of new_cat_stat variable, which requests array length doesn't equal zero and new_cat_stat doesn't equal None. The new_cat_stat and sum of val (value) will yield if they have values.

The second mapper and reducer (step 2) will sum values. The mapper will yield the input without any change. In reducer, the new_value array has been created without any values. For-loop is used to look at values. Inside the loop, new_value will append value (val). Finally, if statement sed to check about the length of new_value array, which requested to be not equal zero. If the if statement is true, the reducer will yield the category with state and sum of values (new_value).

The third mapper and reducer (step 3) will rank the categories and state by their values. In mapper, it will yield None, cat (category and state), and values, after putting cat and values on one tuple. In reducer, sorted() function has been used to order the values by using lambda. The for-loop will look on sorted_values, which has

all values after sorted. Inside for-loop, it will yield value[0] value[1], which value[0] has [category and state] and value[1] has aggregation value.
Step 1: The url to track the job:

```
"Phishing-Offline"      2.2451251236751477e+22
"Scamming-Active"       2.209695235667906e+22
"Scamming-Offline"      1.6235500337815102e+22
"Phishing-Active"       4.5315978714979374e+21
"Fake ICO-Offline"      1.35645756688963e+21
"Phishing-Inactive"     1.4886777707995027e+19
"Scamming-Suspended"    3.71016795e+18
"Phishing-Suspended"    1.63990813e+18
```

Figure 11: output of correlate with certainly known scams

According to output (see figure 10), it can be seen that phishing in offline state has the highest number, despite the scamming category being the most lucrative. Then, scamming in the active state got the second rank. Fake ICO just has offline. The suspended state got the last status, whether in scamming or phishing.

B. Miscellaneous Analysis:

1. Fork the Chain:

The Byzantine fork was one of several fork impacts on Ethereum's blockchain that did in October 2017. Thus, this task will analyze Byzantium fork before and after October 2017 in average number of transactions, an average of value in transactions, and the average of gas price.Average

- Numbers of transactions:

Generating after and before fork will be in two separates python files, that the different just will be in if statement for timestamp in mapper.
In mapper, if statement will check on line's length, which must be equal 7 in each line in transactions dataset. Then, epoch_time will assigned to index 6, which is block_timestamp. The time.strftime() function will transfer timestamp to formal date (month and year) by using time.gmtime(). The major step is choses all date before or after October 2017, which is 1508131331 in timestamp, by using if statement. Finally, if the timestamp less than 1508131331, that will generate date before October 2017. Otherwise, if the timestamp more than 1508131331 that will display after October 2017. The mapper will yield formal date with number 1, that to count how many transactions is happened.
The combiner has been used to make the process faster. Moreover, the combiner and reducer have same code, which is yield each year with sum of value.

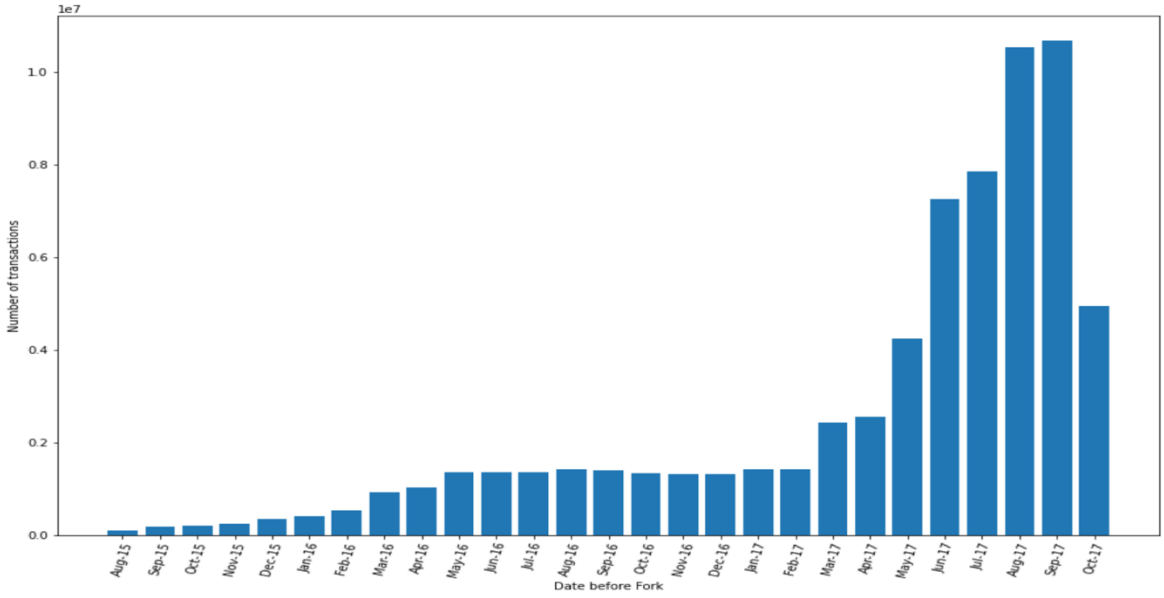


Figure 12: output of average of number of transactions before fork

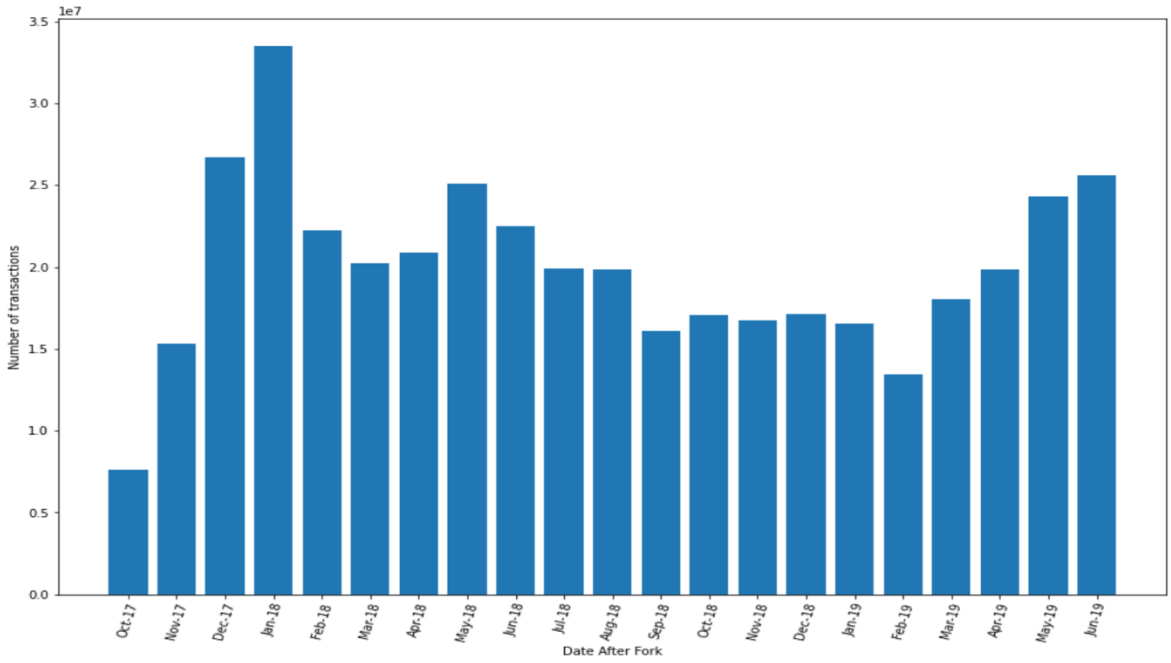


Figure 13: output of average of number of transactions after fork

According to two plots above (figure 12 and figure 13), the number of transactions significantly increased after October 2017, that means using Ethereum has been risen.

- Average of values in transactions:
  Generating after and before the fork will be in two separate python files, and the difference just will be in if statement for timestamp in mapper.
  In mapper, if statement will check on line's length, which must be equal 7 in each line in transactions dataset. Then, epoch_time will be assigned to index 6, which is block_timestamp. The time.strftime() function will transfer timestamp to formal date (month and year) by using time.gmtime(). The major step is to choose all dates before or after October 2017, which is 1508131331 in timestamp, by using if statement. Finally, if the timestamp less than

1508131331, that will generate a date before October 2017. Otherwise, if the timestamp is more than 1508131331, that will display after October 2017. The mapper will yield a formal date with the number 1, that to count how many transactions is happened.
The combiner has been used to make the process faster. Moreover, the combiner and reducer have the same code, which yields each year with a sum of values.
In the combiner, value (val) and count variables have been assigned to two empty arrays. For-loop has been used to look inside values, which contains a tuple of values and the number 1.
Inside the for-loop, the array of values will append index 0, which is values in a tuple, whereas the array of the count will append index 1, which is number 1 in a tuple. When the for-loop finish, the combiner will yield the year (date) with the sum of the array of values and sum of the array of count. The sum of the array of values and sum of the array of the count will be in the tuple. As mentioned above, the reducer has the same code, but it will yield the year (date) with average, which is the sum of the array of values divided by the sum of the array of count.

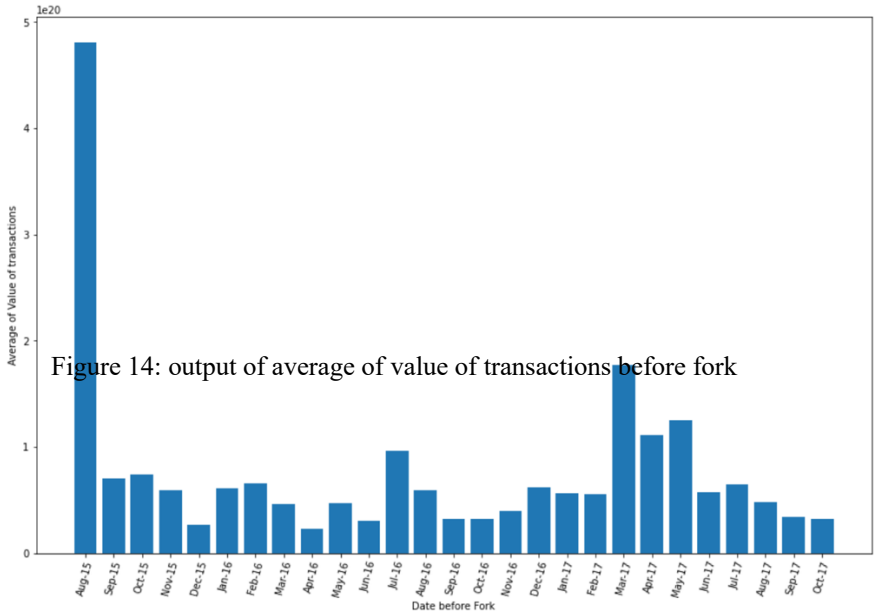


Figure 14: output of average of value of transactions before fork

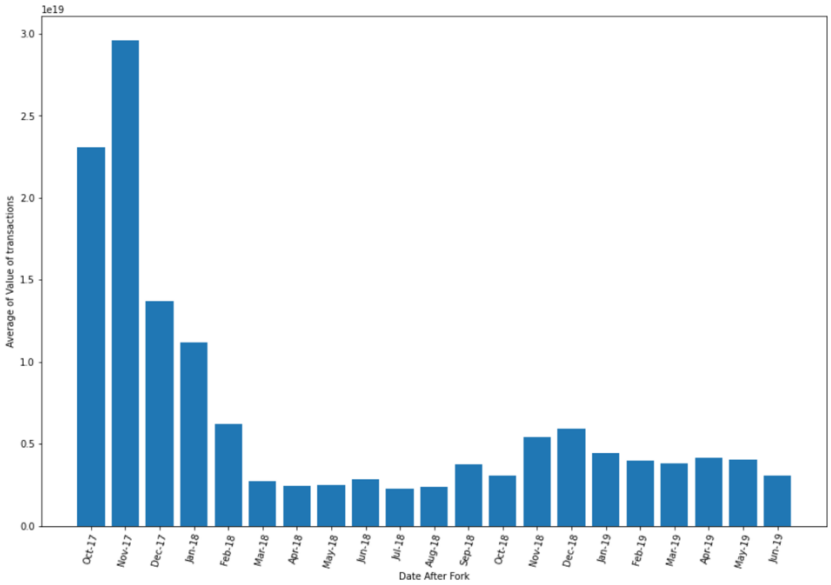


Figure 15: output of average of value of transactions after fork

According to the two plots above (figure 14 and figure 15), it can be seen that both charts fluctuated, but the value after the fork has been decreased, which means the price dropping of Ethereum.

- Average of gas price:

  Generating after and before fork will be in two separates python files, that the different just will be in if statement for timestamp in mapper.
  In mapper, if statement will check on line's length, which must be equal 7 in each line in transactions dataset. Then, epoch_time will assign to index 6, which is block_timestamp. Furthermore, values will assign to index 5. The time.strftime() function will transfer timestamp to formal date (month and year) by using time.gmtime(). The major step is to choose all dates before or after October 2017, which is 1508131331 in timestamp, by using if statement. Finally, if the timestamp is less than or equal to 1508131331, that will generate a date before October 2017. Otherwise, if the timestamp is more than or equals 1508131331 that will display after October 2017. The mapper will yield formal date, values, and number 1, which values, and number 1 will be on a tuple.
  The combiner has been used to make the process faster. Moreover, the combiner and reducer have the same code, but the difference will be in yield. In the combiner, value (val) and count variables have been assigned to two empty arrays. For-loop has been used to look inside values, which contains a tuple of values and the number 1.
  Inside the for-loop, the array of values will append index 0, which is values in a tuple, whereas the array of the count will append index 1, which is number 1 in tuple. When the for-loop finish, the combiner will yield the year (date) with the sum of the array of values and the sum of the array of the count. The sum of the array of values and sum of the array of the count will be in the tuple.
  As mentioned above, the reducer has the same code of combiner, but it will yield the year (date) with average, which is the sum of the array of values

divided by the sum of the array of the count.

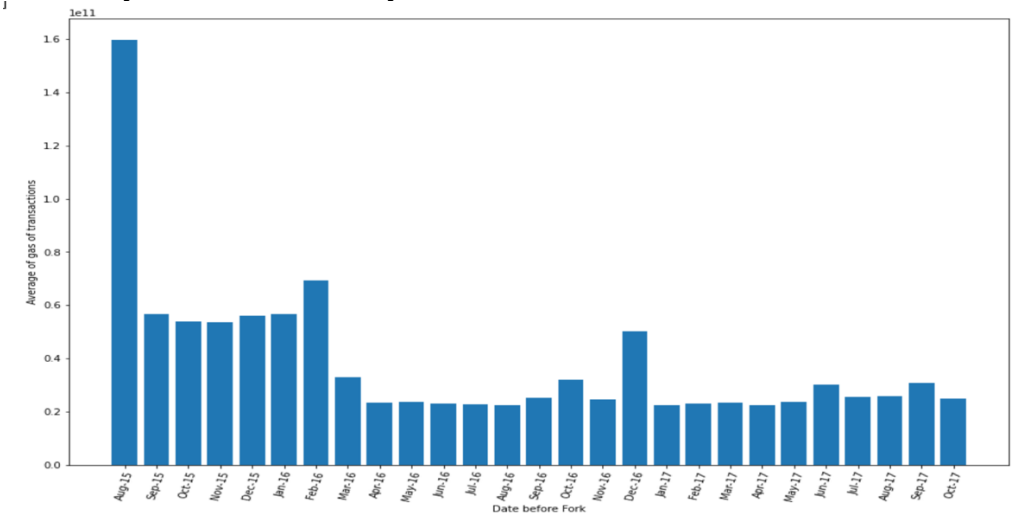


Figure 16: output of average of gas price of transactions after fork

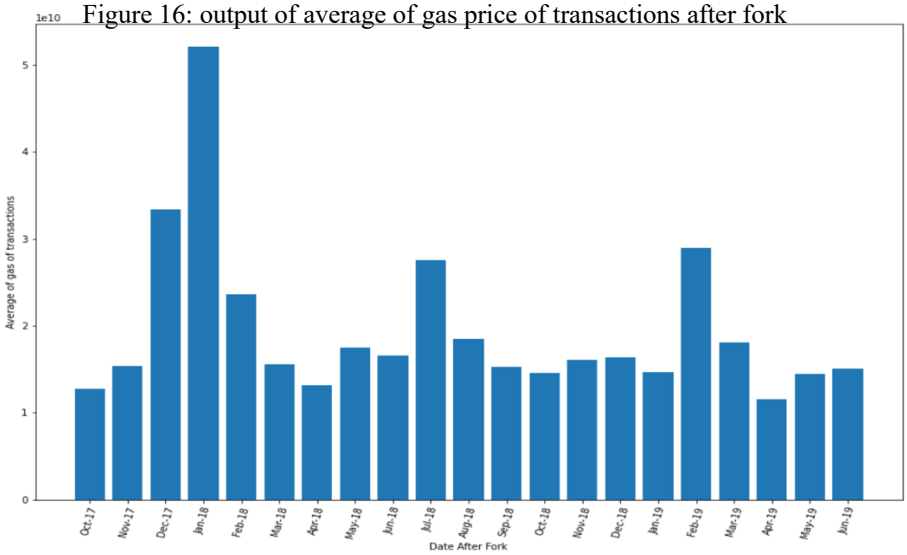


Figure 17: output of average of gas price of transactions after fork

According to two plots above (figure 16 and figure 17), the gas price significantly grown after fork, when compeering both plots. It means gas price increased.

Who profited most from this:
To answer this question, two MapReduce have been used and used MRStep to order the steps.
First MapReduce, In mapper, if statement will check about line length, which each line equal seven will be true. Thus, address (index 2), values (index 3), and epoch_time (index 6) have been assigned. If epoch_time is more than or equal to 1508131331, which is October 2017, the mapper will yield address and values. The combiner and reducer have the same code, which yields the address with the sum of values.
Second MapReduce, the mapper will yield None, address, and values, which address and values will be in a tuple.
The combiner() has been used to make processing faster. Inside the combiner, sorted() function is used to order values by using the lambda function. Moreover, a new variable has been assigned, named i and assigned by 0. Finally, for-loop has

been used to yield value, which content address and aggregation. So, i variable will increase by 1 (inside for-loop) until reaching 10 (using if statement to check), which will help to count and print the top 10. The reducer has the same combiner steps.

```
"0x3f5ce5fbfe3e9af3971dd833d26ba9b5c936f0be"    5741561714038678190018988 3
"0x876eabf441b2ee5b5b0554fd502a8e0600950cfa"    4015767887861936303557417 9
"0xfa52274dd61e1643d2205169732f29114bc240b3"    1746968010584345154631951 3
"0x6cc5f688a315f3dc28a7781717a9a798a59fda7b"    1732700426327196215019657 2
"0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8"    1509330929221533207799548 4
"0xd551234ae421e3bcba99a0da6d736074f22192ff"    1185142526328082764701984 9
"0x5e032243d507c743b061ef021e2ec7fcc6d3ab89"    1182916934378292313614527 7
"0x0681d8db095565fe8a346fa0277bffde9c0edbbf"    1182371643316662271268713 3
"0x564286362092d8e7936f0549571a803b203aaced"    1149269078780909041315322 2
"0xf4a2eff88a408ff4c4550148151c33c93442619e"    9256195852226654400425950
```

Figure 18: output address of most profited

According to output of this task, as shown in figure 18, it displayed the top ten addresses which those most profited. The first address is considered the most profited because it has highest value in aggregation value.

2. **Gas Guzzlers**:
   - How has gas price changed over time?
     Average of gas price has been calculated for each month, that comes from transactions dataset.
     MapReduce have been used for this task. In mapper, if statement checked for each length's line after split(','), which transactions dataset must have 7 lengths for each line. Inside if statement, gas price (index 5) and block_timestamp (index 6) will be assigned to gas and epoch_time variables. The block_timestamp has been conversed to date formal (month and year) by using time.gmtime() function, which inside time.strftime() function. The mapper will yield time after conversation, gas price, and number 1, that gas price, and number 1 will be in one tuple.
     The combiner and reducer had the same code but there is slightly different in yield part. In combiner, count and total are two variables, that created and both assigned to 0. For-loop has been used to look inside values, which contains a tuple (gas price and number 1). Inside for-loop, count variable will aggregate value in index 0, which is gas price, and total variable will aggregate value in index 1, which is number 1. After finish for-loop, the combiner will yield time, count and total variables. However, as a mentioned above the reducer has same code, but it will yield time and count variable divided to total variable.

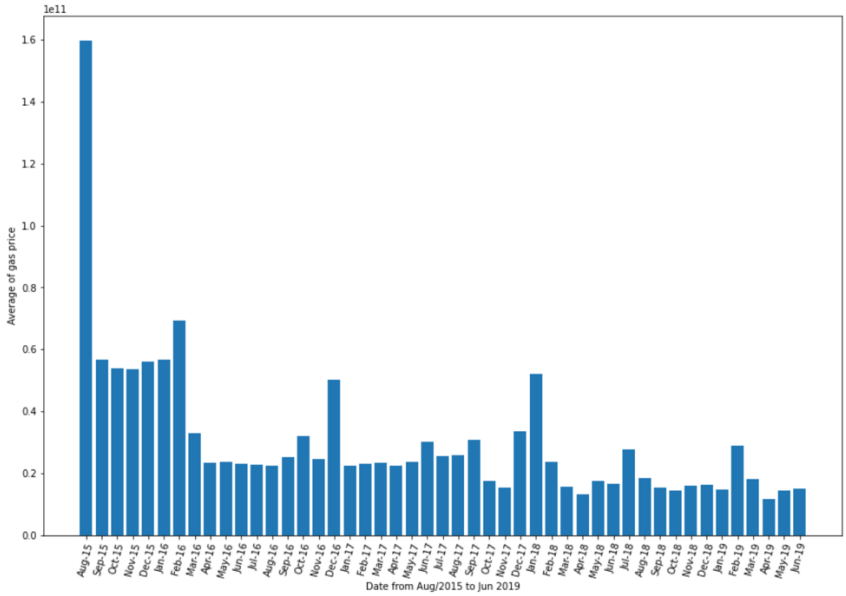


Figure 19: output of gas price changed over time

According to output in figure 19, generally the price fluctuated and near to decreased over the time, in some month there are slight increased such as Feb 2016 and Dec 2016.

- Have contracts become more complicated?

To answer this question, the blocks and contracts are important datasets. Two MapReduce have been used for this task. MRStep has been used to order steps in the python file.

For the first step: In mapper, if statement will check about length's line and determine whether the line comes from the blocks or contracts datasets. If the length's line equals 5, that means coming from contracts. The block_number column (index 3) will be assigned to block_num variable. Finally, it will yield block_num with a triple of number 1, which triple 1 will be in a tuple (used tuple 1 because blocks will yield 4 variables). In contrast, the length's line equals 9, which means coming from blocks. Thus, number (index 0), difficulty (index 3) and timestamp (index 7) are important columns in this dataset. The timestamp conversed to date formal (month and year) by using time.strftime() function, as well mention above in Fork the Chain section about this function. Here, it will yield block number, time after conversion, difficulty, and number 2. However, time after conversion, difficulty and number 2 will be in a tuple. Number 1 in contracts and number 2 in blocks have been added to separate each data in the next step.

In reducer, it will join lines. Three variables have been assigned, the first variable named complic and assigned to 0, second variable named year and assigned to empty array, and the third variable named x and assigned to None. For-loop has been used to look inside values, which contain a tuple. Inside for-loop, if condition used to check on index 2 of the tuple. If it equals 1 (comes from contracts), x variable will be index 1 of the tuple. Otherwise, it equals 2 (comes from blocks), the year array will append index 0, and complic will be index 1. Finally, after the for-loop finish, if-conditional will check about x variable and length of year array, that both must have values. If the if conditional is true, the reducer will yield year and complic values.

Step 2: this step will calculate the average of difficulty, known as a complic. The mapper will yield year (timestamp in date formal), difficulty (complic), and number 1, which difficulty and number 1 will be in the tuple. In reducer, compli_value and count have been created, which are empty arrays. For-loop will be used to look inside values, which contains a tuple of difficulty and number 1. The compli_value array will append index 0 in the tuple (difficulty), and the count array will append index 1 in the tuple (number 1). After the for-loop finish, two arrays will use the sum function separately. Finally, the reducer will yield the year and total value divided by the total of the count.

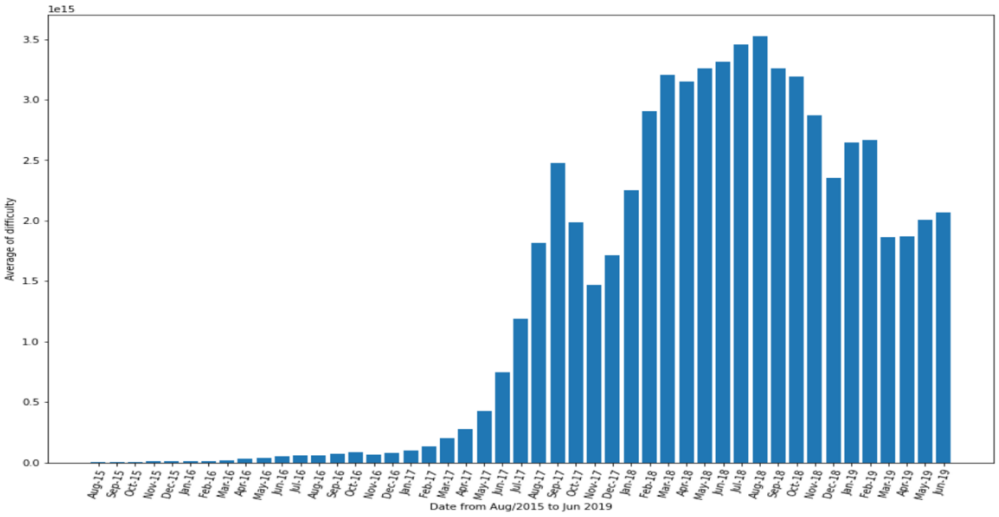


Figure 20: output of average of difficulty over time

According to output of average of difficulty over time in figure 20, the difficulty has been increased over the time, which means the contracts become more complicated over the period.

- Have contracts requiring more gas, or less?
  To answer this question, the blocks and contracts are important datasets. Two MapReduce have been used for this task. MRStep has been used to order steps in the python file.
  For the first step: In mapper, if statement will check about length's line and determine whether the line comes from the blocks or contracts datasets. If the length's line equals 5, that means coming from contracts. The block_number column (index 3) will be assigned to the block_num variable. Finally, it will yield block_num with a triple of number 1, which triple one will be in a tuple (used tuple 1 because in blocks will yield 4 variables). In contrast, the length's line equals 9, which means coming from blocks. Thus, number (index 0), gas used (index 6) and timestamp (index 7) are important columns in this dataset. The timestamp conversed to date formal (month and year) by using time.strftime() function, as well mentioned above in Fork the Chain section about this function. Here, it will yield block number, time after conversion, gas_used, and number 2. However, time after conversion, gas_used and number 2 will be in tuple. Number 1 in contracts and number 2 in blocks have been added to separate each data in the next step.

  In reducer, it will join lines. Three variables have been assigned, the first variable named Gas_used and assigned to 0, the second variable named year and assigned to

empty array, and the third variable named x and assigned to None. For-loop has been used to look inside values, which contain a tuple. Inside for-loop, if condition used to check on index 2 of the tuple. If it equals 1 (comes from contracts), x variable will be index 1 of a tuple. Otherwise, it equals 2 (comes from blocks), year array will append index 0, and Gas_used will be index 1. Finally, after the for-loop finish, if-conditional will check about x variable and length of year array, that both must have values. If the if-conditional is true, the reducer will yield year and Gas_used values.

Step 2: this will calculate the average gas used, known as a Gas_used. The mapper will yield year (timestamp in date formal), Gas_used, and number 1, which Gas_used and number 1 will be in the tuple. In reducer, Gas_used and count have been created, which are empty arrays. For-loop will be used to look inside values, which contains a tuple of Gas_used and number 1. The Gas_used array will append index 0 in the tuple (Gas_used), and the count array will append index 1 in the tuple (number 1). After the for-loop finish, two arrays will use the sum function separately. Finally, the reducer will yield the year and total of Gas_used divided by the total of the count.

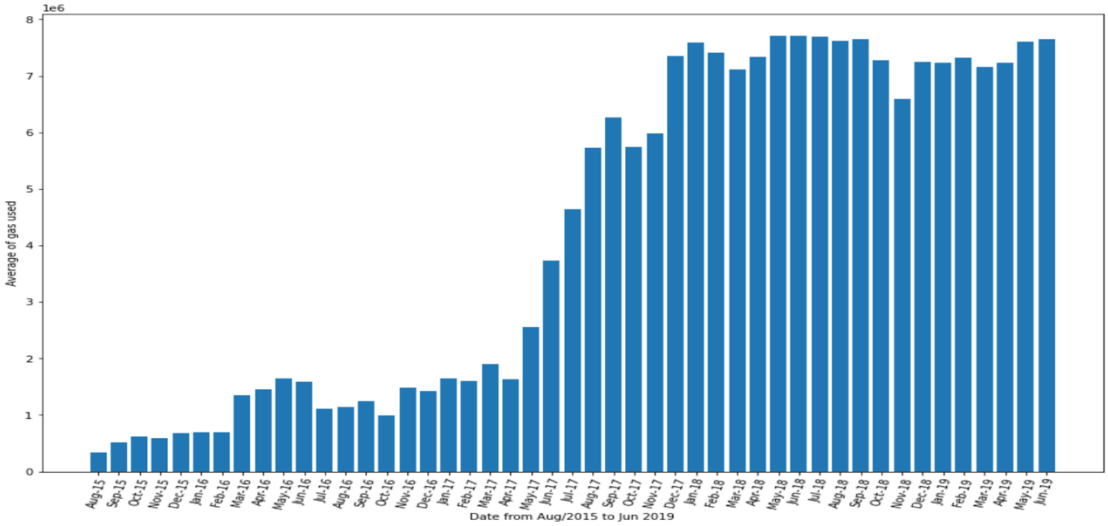


Figure 21: output of average of gas used over time

According to output of gas used over time in figure 21, the gas used in blocks increased over the time, that means is requested more gas.

- Correlate the complexity for some of the top-10 contracts:
  To correlate the complexity with part B output, finding block_number for some of the top-10 contracts' addresses separately will be the first step. Then, using the output of the previous step will be the second step to correlate the specific address with difficulty and timestamp in the blocks dataset. Those two steps will work separately, which means each top-ten address will dealing the individual.
  MapReduce has been used at this point. In the first step, if conditional used to check the line's length, which each line must be equal to 7. block_number (index 0) and to_address (index 2) have been assigned to block_num and adrr variables, respectively. Inside if condition, other if conditional has been used to pick just one address, that address is one of the top-ten contracts in part B output. Moreover, it will change with each observation. Finally, the mapper will yield the specific address with its block_number. In reducer, for-loop is used to yield each address with its block_number.

The second step will use two MapReduce. This step used the same code of (contracts become more complicated) part but changed using the transactions dataset to the previous step's output data.

- First contract in top-ten is 0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444:
  First step:

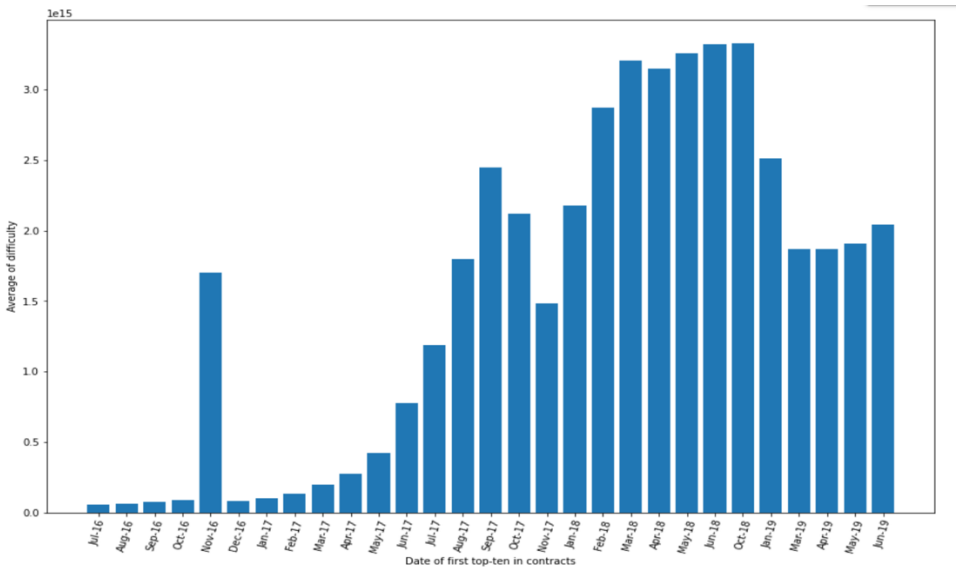


Figure 22: the complexity output of the first top-ten contracts.

According to the complexity output of the first top-ten contracts (see figure 22), generally, the difficulty increased over time. However, the difficulty significantly grew up in Nov/2016, then decreased in the next month. After that, it gradually increased, but in Sep/2017, it dropped until Nov/ 2-17. Furthermore, the difficulty back to decrees in Jan/2019.

- Second contract in top-ten is 0xfa52274dd61e1643d2205169732f29114bc240b3:
  First step:

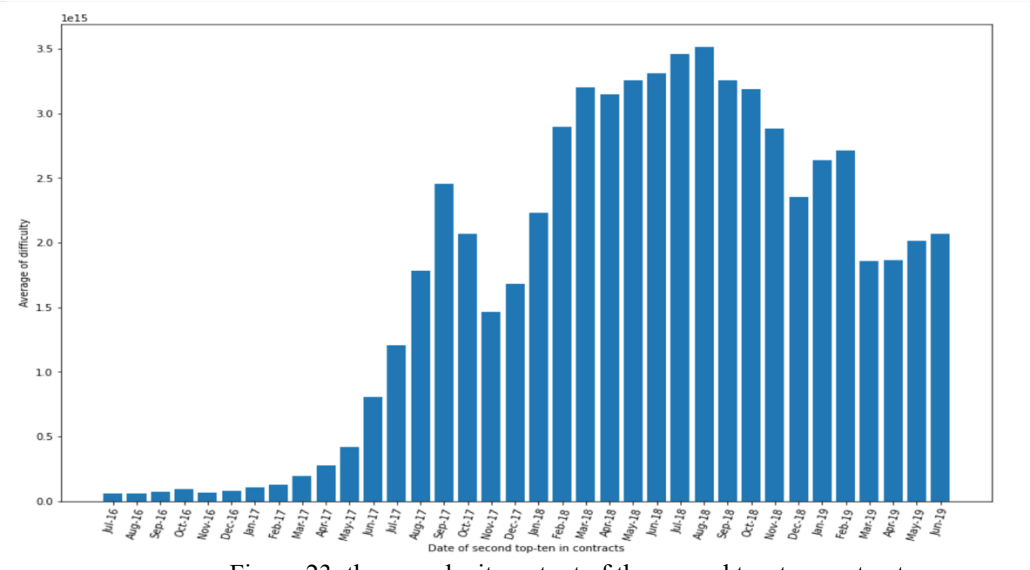


Figure 23: the complexity output of the second top-ten contracts.

According to the complexity output of the second top-ten contracts (see figure 23), generally the difficulty increased over the time. There are some points that is decreased, such as in Oct-2017 and Sep-2018.

- Third contract in top-ten is 0x7727e5113d1d161373623e5f49fd568b4f543a9e:

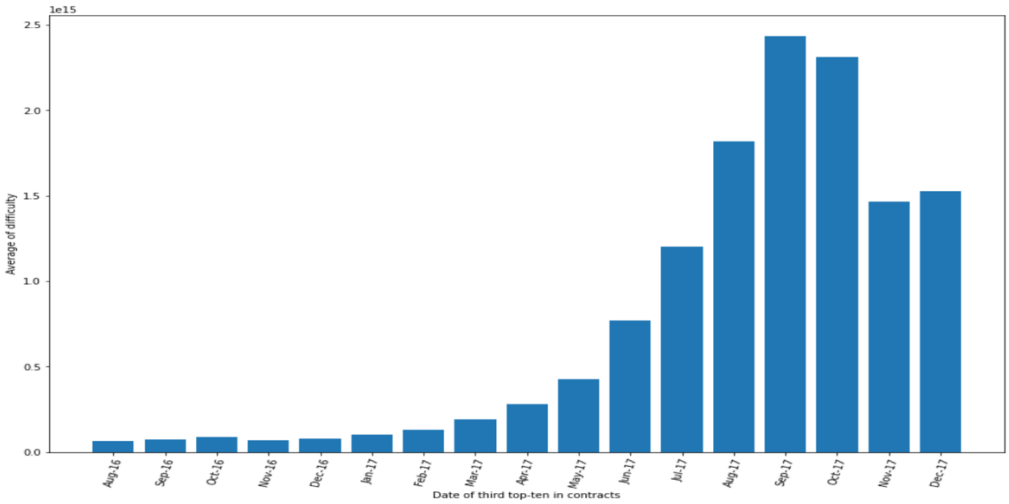


Figure 24: the complexity output of the third top-ten contracts.

According to the complexity output of the third top-ten contracts (see figure 24), the difficulty period was from Aug/2016 to Dec/2017. However, it was increasing until Sep/2017, then decreased but there is slightly grown in Dec/2017.

- Fourth contract in top-ten is 0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef:

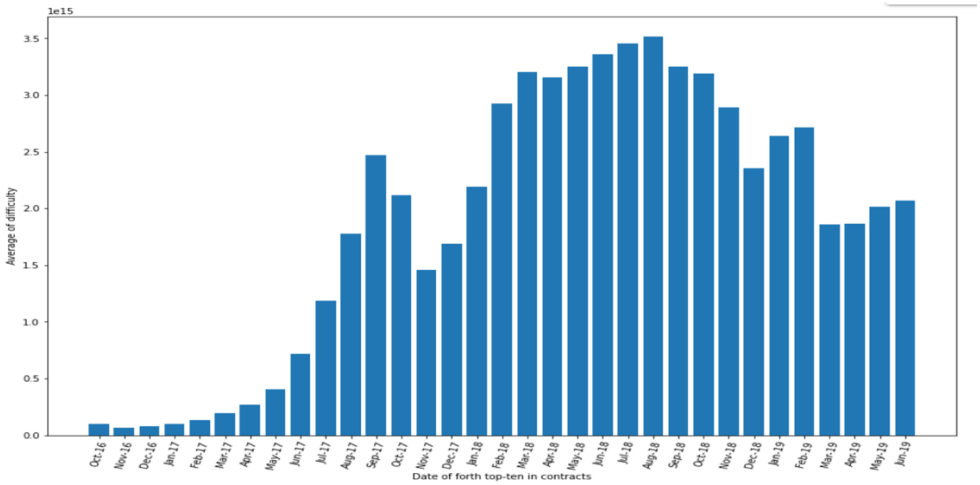


Figure 25: the complexity output of the fourth top-ten contracts.

According to the complexity output of the fourth top-ten contracts (see figure 25), difficulty period was from Oct/2016 to Jan/2019. However, it was increasing generally, but there are some points that dropped, such as Oct/2017 and Nov/2018.

- Fifth contract in top-ten is 0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8:

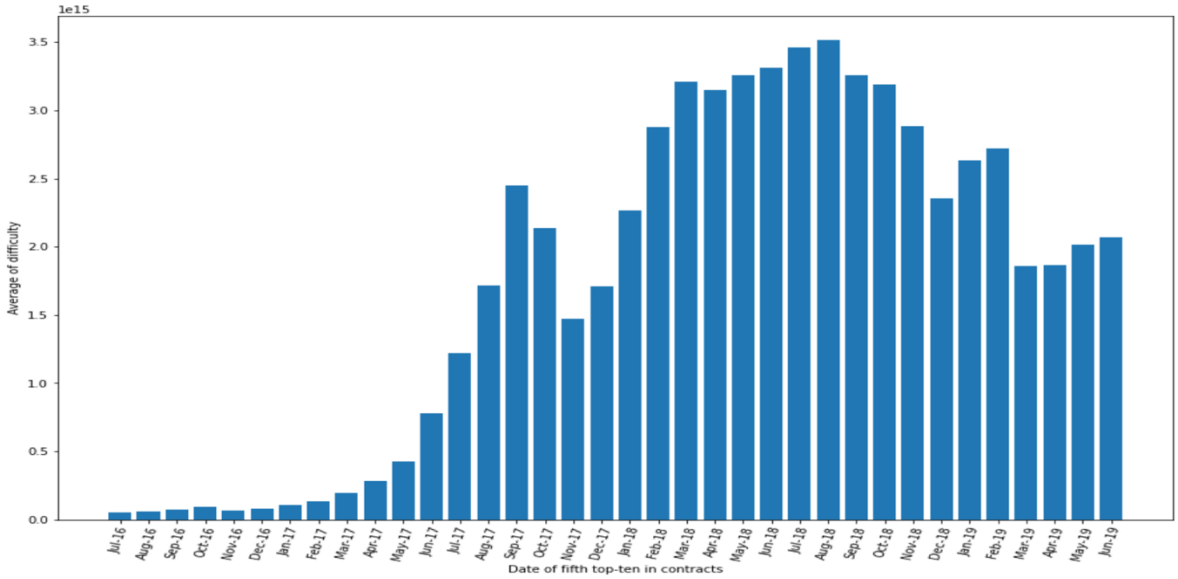


Figure 26: the complexity output of the fifth top-ten contracts.

According to the complexity output of the fifth top-ten contracts (see figure 26), difficulty period was from Jan/2016 to Jan/2019. However, it was increasing generally, but there are some points that dropped, such as Oct/2017 and Sep/2018.

3. Comparative Evaluation:

Part B has been solved by using MapReduce, so spark will be used in this task. SparkContext() function used to connect to spark cluster. Moreover, two functions have been used, the first function for the transactions dataset and the second for the contracts dataset. The first function will check about line length, which must be equal to 7. The second function has the same job, but it must be equal to 5. The if statement will check about those conditionals. If the line length has the same condition, mentioned above, the function will return True. Otherwise, it will return False. The textFile() function has been used to read files from HDFS, which will read transactions and contracts datasets separately. Then, filter() function used the two functions separately, which created first, to check about the line's length. The map() function is used to apply the function in each line. Also, map() has been used separately. For the transactions dataset, it will split (',') and select to_address in index two and value in index three by using the lambda function. For the contracts dataset, it will split (',') and select the address in index 0 as well as number 0 by using the lambda function. Furthermore, cache () function is used separately to mapper output, which uses as a checkpoint; when the spark fails, it will recompute it. The next step is near Job 1 in part B. The reduceByKey () function will use for transactions (after cache() it) , which will sum values by using lambda. The result of reduceByKey () will save by using persist() function. Then job 2, output of persist() function will join to contracts (after cache()) by using join() function. The filter() function will check that both values do not equal None. Finally, job 3, takeOrdered() function is used to order and take the first ten values. For-loop has been used to print values in index 0 and index 1.

Figure 27: output of Part B by using spark

Figure 27 is output of task B when using spark.
There is no difference in the result, but MapReduce took a longer time than Spark. However, three times have been run both programs; the start time and finish times will be mentioned below.
When using Spark:
In the first run, the start time is 00:51:35, and the finish time is 00:53:21, so the total time is 1 minute and 46 seconds, which is 106 seconds.
In the second run, the start time is 01:45:03, and the finish time is 01:46:49, so the total time is 1 minute and 46 seconds, which is 106 seconds.
In the third run, the start time is 01:47:21, and the finish time is 01:49:10, so the total time is 1 minute and 49 seconds, which is 109 seconds.
On the other hand, when using MapReduce:
In first run:
Job1: the start time is 02:17:15, and the finish time is 02:45:11, so the total time is 27 minutes, and 56 seconds.
Job2: the start time is 02:53:18, and the finish time is 02:59:06, so the total time is 5 minutes, and 48 seconds.
Job3: the start time is 03:03:14, and the finish time is 03:03:46, so the total time is 32 seconds.
Thus, total number of the first run is 34 minutes and 32 seconds, which is 2072 seconds.
In second run,
Job 1: the start time is 03:32:28, and the finish time is 03:57:41, so the total time is 25 minutes, and 13 seconds.
Job 2: the start time is 04:03:27, and the finish time is 04:09:20, so the total time is 5 minutes and 53 seconds.
Job 3: the start time is 04:11:46, and the finish time is 04:12:18, so the total time is 32 seconds.
Thus, total number of the third run is 31 minutes and 38 seconds, which is 1898 seconds.
In third run,
Job 1: the start time is 04:14:20, and the finish time is 04:39:26, so the total time is 25 minutes, and 6 seconds.
Job 2: the start time is 04:49:16, and the finish time is 04:55:02, so the total time is 5 minutes and 46 seconds.
Job 3: the start time is 04:56:12, and the finish time is 04:56:45, so the total time is 33 seconds.
Thus, the total number of the second run is 31 minutes and 25 seconds, which is 1885 seconds.
The average result for MapReduce ((2072+ 1898+ 1885)/3) is 1951.66 seconds, representing 32 minutes and 31 seconds.

The average result for Spark ((106+106+109)/3) is 107 seconds, representing 1 minute and 47 seconds.
According to three times time and the average for both programs, it is clear evidence that Spark is faster than MapReduce, because Spark is processing data in random access memory (RAM), whereas Hadoop MapReduce needs to continuously data back to disk after mapper or reducer step.
As a result, Spark is the best framework, that can be used in this task because Spark gives a fast performance. If MapReduce has been used, approximately 30 minutes is needed to finish as well as three mappers and reducer (job) requested. In contrast, Spark will be finished in around 2 minutes because of doing the entire task in-memory.