ECS765P – BIG DATA PROCESSING

COURSEWORK: ETHEREUM ANALYSIS

NAME: SAILAJ GHOSH

ID: 210990050

PART A - TIME ANALYSIS
NUMBER OF TRANSACTIONS

URL:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_4356/

The number of transactions occurring every month between the start and end of the dataset was obtained using MapReduce.

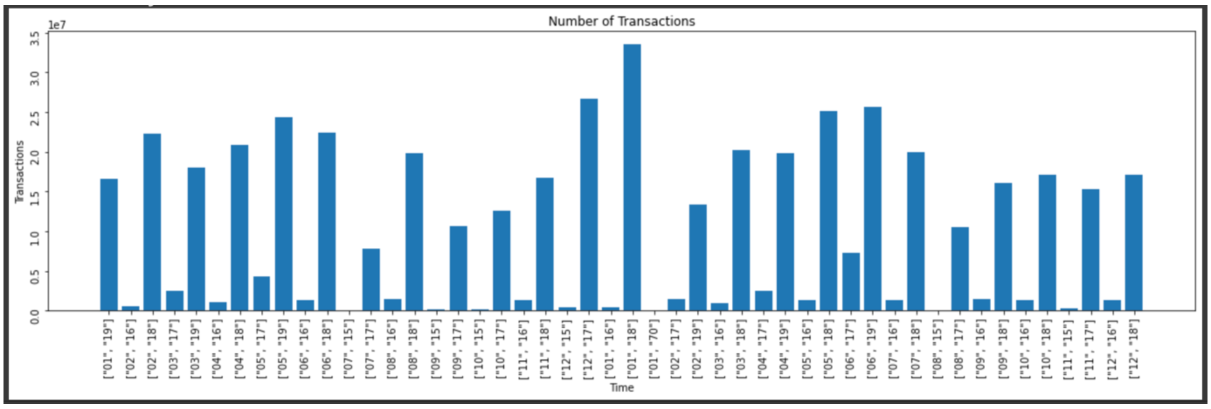
library: “mrjob” for MapReduce framework, “time” for converting epoch format.

mapper: Splitting the input file which is comma separated and checking the number of fields. Since we are using the ‘blocks’ dataset for this execution the number of fields should be 9. Then, initialising the required fields to aggregate the transaction count for each month of the years in the dataset.

combiner: Extracting the “key, value” pairs and adding the transaction count for each month of the years.

reducer: Extracting the “key, value” pairs and adding the transaction count for each month of the years.

For the bar plot, python was used where the number of transactions is plotted against the months of the years.



PART A - TIME ANALYSIS
AVERAGE VALUE OF TRANSACTIONS

URL:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_4366/

The average value of transactions occurring every month between the start and end of the dataset was obtained using MapReduce.

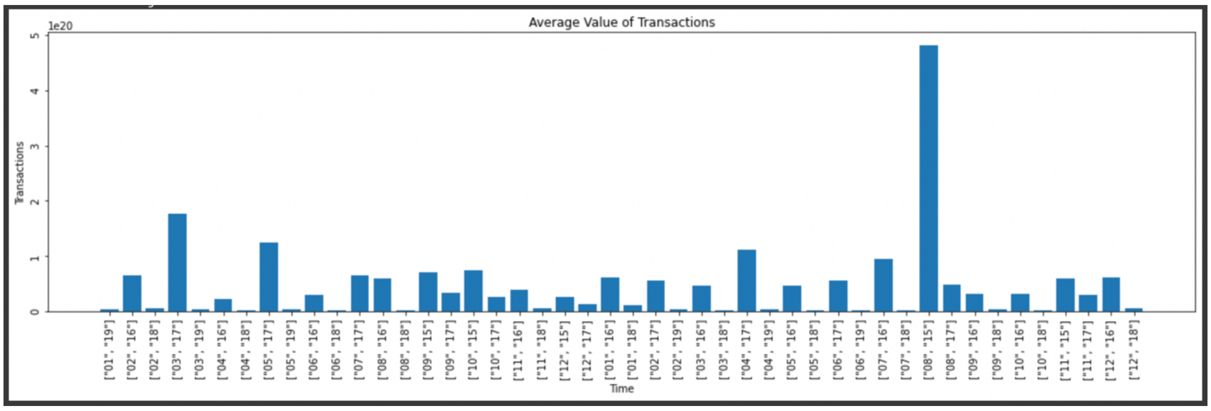
library: “mrjob” for MapReduce framework, “time” for converting epoch format.

mapper: Splitting the input file which is comma separated and checking the number of fields. Since we are using the ‘transactions’ dataset for this execution the number of fields should be 7. Then, initialising the required fields to aggregate the value and count for each month of the years in the dataset.

combiner: Extracting the “key, value” pairs and adding both value and count for each month of the years.

reducer: Extracting the “key, value” pairs and adding both value and count for each month of the years. Finally, dividing the total value by the total count for each month of the years to obtain the result.

For the bar plot, python was used where the average value of transactions is plotted against the months of the years.



PART B – TOP TEN MOST POPULAR SERVICES

URL:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_4487/
URL:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_4532/

To evaluate the top 10 smart contracts by total Ether received, MapReduce was used.

library: “mrjob” for MapReduce framework.

mapper1: Since there are two input files namely ‘transactions’ and ‘contracts’, we check the number of fields respectively. Now splitting the files which are comma separated and initialising the required fields which in this case are address and value.

reducer1: Extracting the “key, value” pairs and appending the list of values if the address exists in both the input files and adding the values for each address.

mapper2: Extracting the “key, value” pairs and yielding them as value field.

reducer2: Extracting the values and sorting them according to the value field in descending order for the top ten results.

steps: Joining the multi-step jobs.

Output:

```
"0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444"       84155100809965865822726776
"0xfa52274dd61e1643d2205169732f29114bc240b3"       45787484483189352986478805
"0x7727e5113d1d161373623e5f49fd568b4f543a9e"       45620624001350712557268573
"0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef"       43170356092262468919298969
"0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8"       27068921582019542499882877
"0xbfc39b6f805a9e40e77291aff27aee3c96915bdd"       21104195138093660050000000
"0xe94b04a0fed112f3664e45adb2b8915693dd5ff3"       15562398956802112254719409
"0xbb9bc244d798123fde783fcc1c72d3bb8c189413"       11983608729202893846818681
"0xabbb6bebfa05aa13e908eaa492bd7a8343760477"       11706457177940895521770404
"0x341e790174e3a4d35b65fdc067b6b5634a61caea"       8379000751917755624057500
```

PART C – TOP TEN MOST ACTIVE MINERS

URL:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_4542/
URL:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_4543/

To evaluate the top 10 miners by the size of the blocks mined, MapReduce was used.

library: “mrjob” for MapReduce framework.

mapper1: Splitting the input file which is comma separated and checking the number of fields. Since we are using the ‘blocks’ dataset for this execution the number of fields should be 9. Then, initialising the required fields which in this case are miner and size.

reducer1: Extracting the “key, value” pairs and adding size for each miner.

mapper2: Extracting the “key, value” pairs and yielding them as value field.

reducer2: Extracting the values and sorting them according to the value field in descending order for the top ten results.

steps: Joining the multi-step jobs.

Output:

```
"0xea674fdde714fd979de3edf0f56aa9716b898ec8"        23989401188
"0x829bd824b016326a401d083b33d092293333a830"        15010222714
"0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c"        13978859941
"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5"        10998145387
"0xb2930b35844a230f00e51431acae96fe543a0347"        7842595276
"0x2a65aca4d5fc5b5c859090a6c34d164135398226"        3628875680
"0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01"        1221833144
"0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb"        1152472379
"0x1e9939daaad6924ad004c2560e90804164900341"        1080301927
"0x61c808d82a3ac53231750dadc13c777b59310bd9"        692942577
```

PART D – DATA EXPLORATION
POPULAR SCAMS

URL:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_4581/
URL:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_4610/

To evaluate the analysis for popular scams, MapReduce was used.

library: “mrjob” for MapReduce framework, “json” for parsing json file.

mapper1: Since there are two types of input files, splitting the file which is comma separated and loading the other json file to read the lines. For the ‘transactions’ file, checking if the fields are equal to 7 and initialising the required fields which in this case are address and value. For the ‘scams’ file, looping through the json array, initialising the required fields which in this case are category, status and address and looping through the address field to yield the category and status for each. Also, yielding with a differentiating factor between the two files as ‘0’ and ‘1’ for the next step.

reducer1: Extracting the “key, value” pairs, looping through the value field and checking with our differentiating factor to get the values from the ‘transactions’ file for each pair of category and status.

mapper2: Extracting the “key, value” pairs and yielding them.

Reducer2: Extracting the “key, value” pairs and adding the values for each pair of category and status.

steps: Joining the multi-step jobs.

Output:

```
["Fake ICO", "Offline"]         1356457566889629979678
["Phishing", "Inactive"]        14886777707995030337
["Phishing", "Offline"]         2245125123675149225034
["Phishing", "Suspended"]       1639908130000000000
["Scam", "Offline"]             0
["Scamming", "Active"]          2209695235667979326461
["Phishing", "Active"]          4531597871497938301034
["Scamming", "Offline"]         16235500337815103385736
["Scamming", "Suspended"]       3710167950000000000
```

PART D – DATA EXPLORATION
FORK THE CHAIN

URL:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_6380/

To evaluate the analysis for forks of Ethereum in the past, MapReduce was used.
The fork in Byzantium was analysed which occurred on 16th Oct 2017.

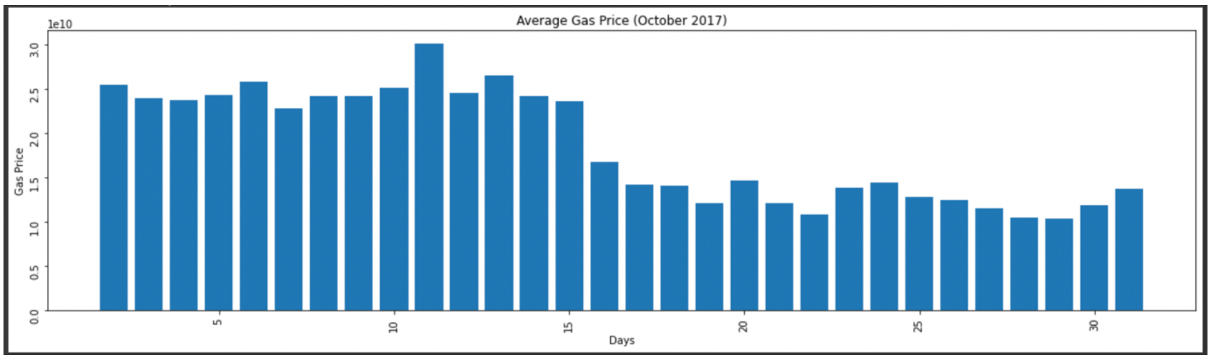
library: "mrjob" for MapReduce framework, "time" for converting epoch format.

mapper: Splitting the input file which is comma separated and checking the number of fields. Since we are using the 'transactions' dataset for this execution the number of fields should be 7. Then, initialising the required fields which in this case are gas price and time, also converting the time to days, months and years. After this checking if the month is October and the year is 2017 to yield all the gas prices for each day of that month.

combiner: Extracting the "key, value" pairs and adding both price and count for each day of the month.

reducer: Extracting the "key, value" pairs and adding both price and count for each day of the month. Finally, dividing the total price by the total count for each day of the month to obtain the result.

For the bar plot, python was used where the average gas price is plotted against the days of October 2017.



The date of fork was 16th Oct 2017. In the above graph, we can see that there was a significant decline in the price of gas compared to the previous days of the same month. Therefore, we can conclude that a price plummet had occurred.

PART D – DATA EXPLORATION
GAS GUZZLERS

URL: http://itl333.student.eecs.qmul.ac.uk:4040/
ID: application_1648683650522_6735

To evaluate the analysis of gas guzzlers, Spark was used.

library: “pyspark” for Spark framework, “time” for converting epoch format.

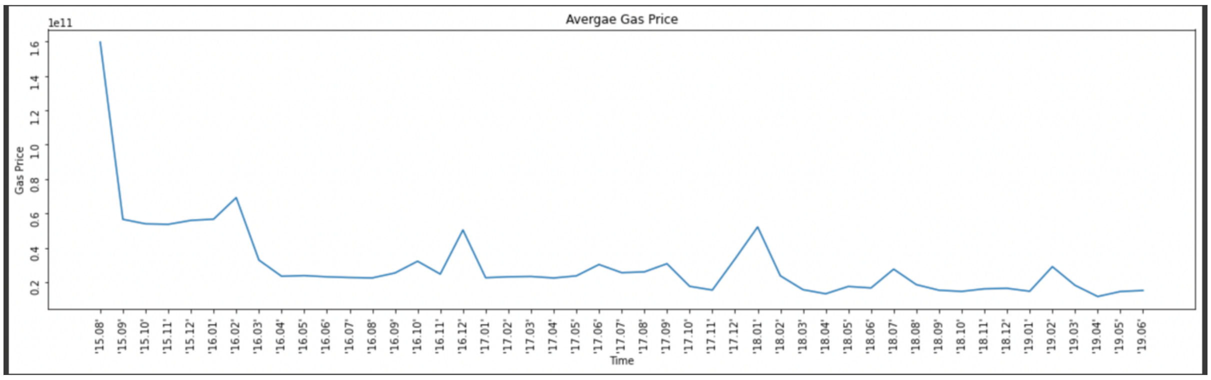
We are using three input files namely ‘blocks’, ‘transactions’ and ‘contracts. We will start by checking the number of fields for each dataset by creating functions for each of the files.

Firstly, initialising the required fields and converting the timestamp, we calculate the average price of gas for each month of all the years which is obtained by dividing the total gas price by total count of that month.

Secondly, initialising the required fields and joining the ‘blocks’ with the ‘contracts’ file using a common joining key so that we can extract the smart contracts only. Then, calculating the gas used and difficulty for every month of the years.

For all the graphs below, python was used.

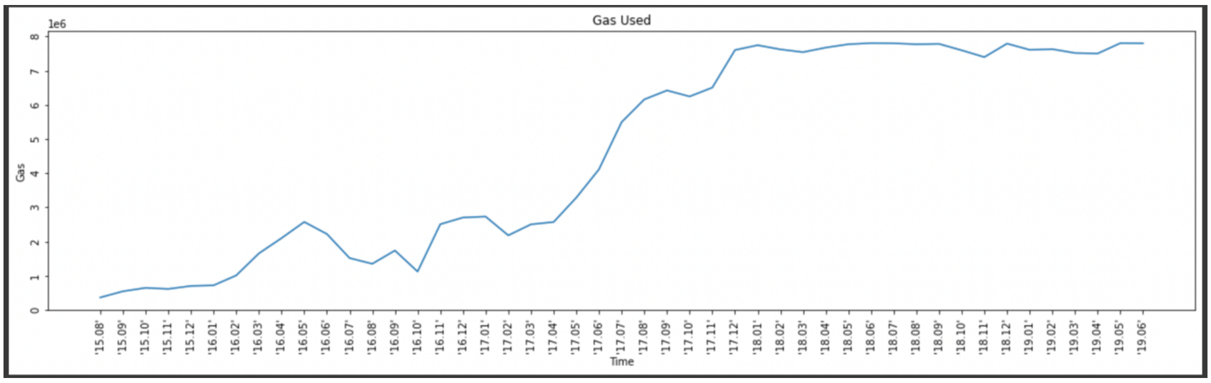
Here, the average gas price is plotted against the months of the years.



In the above graph, we can see that the average gas price has decreased over the timeline and there is spike in the prices at the beginning or end of most years.

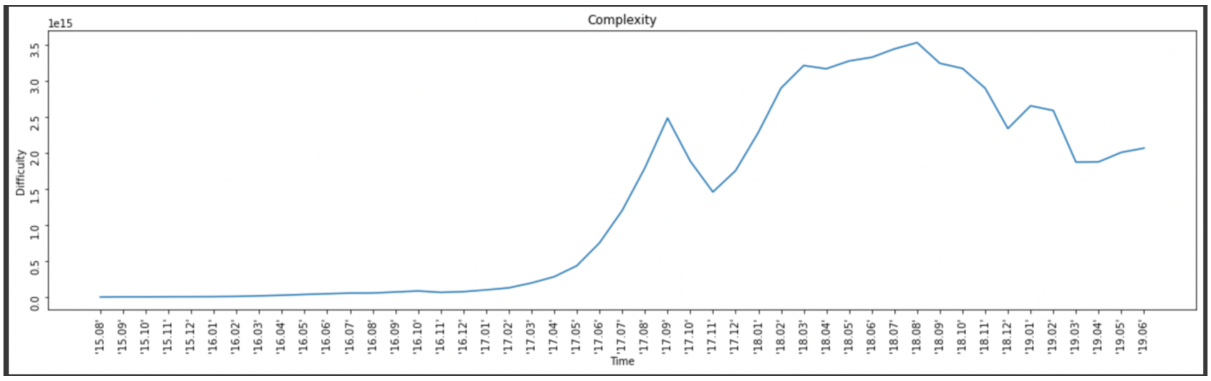
PART D – DATA EXPLORATION
GAS GUZZLERS

Here, the gas used is plotted against the months of the years.



In the above graph, we can see that there is a significant increase in demand over the last few years compared to the previous years and is in a steady state after the increase.

Here, the difficulty is plotted against the months of the years.



In the above graph, we can see that over the timeline contracts have become more complicated with significant changes that can be observed in the years 2017 and 2018.

PART D – DATA EXPLORATION COMPARATIVE EVALUATION

library: “pyspark” for Spark framework.

We are using two input files namely ‘transactions’ and ‘contracts’ where we start by checking the number of fields respectively. Now splitting the files which are comma separated and initialising the required fields which in this case are address and value. The joining operation is performed where we aggregate values from the ‘transaction’ dataset for the joining key which in this case is the address if it exists in the ‘contracts’ dataset, so that it can be labelled as a smart contract. Then, we sort them according to the values in descending order for the top ten results.

Output:

```
0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444:84155100809965865822726776
0xfa52274dd61e1643d2205169732f29114bc240b3:45787484483189352986478805
0x7727e5113d1d161373623e5f49fd568b4f543a9e:45620624001350712557268573
0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef:43170356092262468919298969
0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8:27068921582019542499882877
0xbfc39b6f805a9e40e77291aff27aee3c96915bdd:21104195138093660050000000
0xe94b04a0fed112f3664e45adb2b8915693dd5ff3:15562398956802112254719409
0xbb9bc244d798123fde783fcc1c72d3bb8c189413:11983608729202893846818681
0xabbb6bebfa05aa13e908eaa492bd7a8343760477:11706457177940895521770404
0x341e790174e3a4d35b65fdc067b6b5634a61caea:8379000751917755624057500
```

Comparison:

Executing the same task three times for each framework, the average time taken for MapReduce was around “30 minutes” and the average time taken for Spark was around “3 minutes”.

We can conclude without any doubt that Spark as a framework is more efficient over MapReduce by observing the time taken by them for the same task. Spark computes the given task at a much quicker rate due to the it’s data processing using the RDD(Resilient Distributed Dataset) data structure, whereas MapReduce computes at a much slower rate as it reads and writes files to HDFS.

MapReduce –

1. URL: http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_6525/
   URL: http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_6561/

2. URL: http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_6594/
   URL: http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_6625/

3. URL: http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_4487/
   URL: http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_4532/

Spark –

1. URL: http://itl333.student.eecs.qmul.ac.uk:4040/
   ID: application_1648683650522_6712

2. URL: http://itl333.student.eecs.qmul.ac.uk:4040/
   ID: application_1648683650522_6720

3. URL: http://itl333.student.eecs.qmul.ac.uk:4040/
   ID: application_1648683650522_6727