**BIG DATA PROCESSING**

**ECS765P**

**COURSEWORK: ETHEREUM ANALYSIS**

**ANALYSIS OF ETHEREUM TRANSACTIONS AND SMART CONTRACTS**

**NAME: DHRUV VERMA**

**STUDENT ID: 210879364**

**JANUARY 2022 INTAKE**

**INDEX**

1. **PART A: TIME ANALYSIS**
   1. **PART A**
   2. **PART B**
2. **PART B: TOP TEN MOST POPULAR SERVICES**
3. **PART C: TOP TEN ACTIVE MINERS**
4. **PART D: DATA EXPLORATION**
   1. **COMPARATIVE ANALYSIS**
   2. **FORK THE CHAIN**
   3. **POPULAR SCAMS**
   4. **GAS GUZZLERS**

**ASSIGNMENT**

**PART A1: TIME ANALYSIS (20%)**

**PROBLEM STATEMENT**: Graphical plot of average value of transactions in each month between the start and end of the dataset.

**SOLUTION:**

In order to execute this task I ran a Hadoop job consisting of a mapper which gets the timestamp(field 6) and month, year are calculated. Then months, year and field 1 are yielded. The reducer then counted the number of transactions occurring every month. The output is stored in PartA.txt. Output is then imported the into an excel spreadsheet, sorted the values by date chronologically and plotted the following chart:

**SAMPLE OUTPUT:**

["01", "17"] 1409664

["01", "19"] 16569597

["02", "16"] 520040

["02", "18"] 22231978

["03", "17"] 2426471

["03", "19"] 18029582

["04", "16"] 1023096

["04", "18"] 20876642

["05", "17"] 4245516

["05", "19"] 24332475

["06", "16"] 1351536

["06", "18"] 22471788

["07", "17"] 7835875

["08", "16"] 1405743

["08", "18"] 19842059

["09", "15"] 173805…..

Chart

Description automatically generated

**Job ID:** <http://andromeda.student.eecs.qmul.ac.uk:19888/jobhistory/job/job_1637317090236_22177>

**PART A2:**

**PROBLEM STATEMENT**: Graphical plot the average value of transaction in each month between the start and end of the dataset.

**SOLUTION:**

In order to execute this task previous tasks as previous job were done which gets the timestamp(field 6) and month, year are calculated along with gas price(field 5). Then months, year and field 1 are yielded. The reducer then counted the number of transactions occurring every month. A combiner was then used to count the number of transactions and their sum which are lastly yielded in the reducer after calculating the average. The output is stored in PartA2.txt. Output is then imported and sorted chronologically into an excel spreadsheet and plotted the following chart:

**SAMPLE OUTPUT:**

["01", "17"] 22507570807.719643

["01", "19"] 14611816445.785248

["02", "16"] 69180681134.3894

["02", "18"] 23636574203.828815

["03", "17"] 23232253600.81775

["03", "19"] 18083035519.5223

["04", "16"] 23361180502.721123

["04", "18"] 13153739247.929962

["05", "17"] 23572314972.01515

["05", "19"] 14479858767.711178

["06", "16"] 23021251389.812656

["06", "18"] 16533308366.813047

["07", "17"] 25460300456.232353

["08", "16"] 22396836435.958347….

Chart

Description automatically generated

**JOB ID:**

<http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1637317090236_22195/>

**PART B: TOP TEN MOST POPULAR SERVICE (25%)**

**PROBLEM STATEMENT**: Evaluate the top 10 smart contracts by total Ether received by performing 3 sub-tasks, namely aggregation, joining and top ten validation.

**SOLUTION:**

In this code we are going to use two datasets Transactions and Contracts.First, to aggregate **transactions** to see how much each address within the user space has been involved in. We will aggregate **value** for addresses in the **to\_address** field. In the first MapReduce job we are taking address as the key from both the dataset. We are taking value as a key value pair from the Transaction dataset and we are passing value 1 with it which will help us with segregation. For Contracts dataset we are taking 2 as another value and also passing 1 which is the counter for the reduce stage.

Secondly, in the reducer, if the address for a given aggregate from first part was not present within **contracts** this should be filtered out as it is a user address and not a smart contract. Then the reducer checks the values passed from the mapper and if they are true which means they exist and the record is valid. Then it sums all the values and key and passes this summed value.

Finally, take as input the now filtered address aggregates and sort these via a top ten reducer. In the next mapper we take the values from the reducer and combines the key and values as the reducer has to find the top 10 contracts with “None” being the key. Then in the reducer we sort the values in decreasing order and extract top 10 values by using lambda to specify second field as key. The output is present in the file partB.txt.

**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 ACTIVE MINERS (15%)**

**PROBLEM STATEMENT**: Evaluate the top 10 miners by the size of the blocks mined.

**SOLUTION:**

In this part we are using Blocks dataset to get the miner addresses(field 2) and size(field 4). First, we map all the miners to their respective sizes and then run first reducer using address as key and sizes as value.

Then we run the second mapper with “None” key and get the value as miner addresses with total sizes. After that, we reduce and sort in decreasing order using lambda and specify second field as key for sorting. The obtained output is stored in “partC.txt”, with column 1 specifying the addresses and second columns corresponds to their sizes.

**OUTPUT:**

"0xea674fdde714fd979de3edf0f56aa9716b898ec8" 23989401188

"0x829bd824b016326a401d083b33d092293333a830" 15010222714

"0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c" 13978859941

"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5" 10998145387

"0xb2930b35844a230f00e51431acae96fe543a0347" 7842595276

"0x2a65aca4d5fc5b5c859090a6c34d164135398226" 3628875680

"0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01" 1221833144

"0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb" 1152472379

"0x1e9939daaad6924ad004c2560e90804164900341" 1080301927

"0x61c808d82a3ac53231750dadc13c777b59310bd9" 692942577

**JOB ID:**

<http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1637317090236_22532/>

**PART D: DATA EXPLORATION (40%)**

**1. COMPARATIVE ANALYSIS (10%)**

**PROBLEM STATEMENT:** Running PART B in Spark and compare average run times.

**SOLUTION:**

The overall working of the program is same as in MapReduce, we get both the datasets, i.e., Transactions and Contracts. We check the fields (7 for Transactions, 5 for Contracts), then we map each address to their transaction value. Then we map the address as key for contracts, that then we join with the output given from the transaction reducer. The data is then sorted and the top 10 is given as output in a .txt file. Time library is also imported to calculate the time it takes to run the whole program. Time take by cluster jobs are calculated from the Job\_Counters produced when running the job.

**OUTPUT:**

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

193.629383087

*Last line is the execution time in seconds*. 5 runs were done as following:

1st run: 193.629383087

2nd run: 188.909346104

3rd run: 267.583372116

4th run: 220.59771204

5th run: 187.091112137

**Average run time:** 211.56 seconds or 3 minutes 30 seconds approximately.

For Hadoop, following runs were done:

1st run: 26 minutes 46 seconds

2nd run:29 minutes 33 seconds

3rd run:28 minutes 54 seconds

**Average run time**: 28 minutes 40 seconds approximately

**2. FORK THE CHAIN (10%)**

**PROBLEM STATEMENT**: There have been several forks of Ethereum in the past. Observe effect it had on price and general usage.

**SOLUTION:**

Here we have to choose one of the many forks in the Ethereum network from the past, and observe the change in number transaction or gas price which directly translates to usage increase/decrease in the network.

For the execution of the said problem, we pass the Transactions dataset and retrieve day and month of fork from “field 6” using time library. From field 5, we retrieve gas price and at end of it we put in an if-else condition specifying the date of fork, (16th October, 2017 Byzantium Fork). That way we can get the all the gas price and transaction number of that specific month. Then we use reducer and combiner to aggregate over total gas price and number of transactions. The data is stored in fork.txt. The obtained data is cleaned and plotted to observe the change during the month of fork deployment.

**SAMPLE OUTPUT:**

1 [283871, 100000000000.0]

10 [349605, 20000000000.0]

12 [363411, 28000000000.0]

14 [332169, 25000000000.0]

16 [408297, 4000000000.0]

18 [470449, 4000000000.0]

21 [462902, 20000000000.0]

23 [468168, 4000000001.0]

25 [460419, 4000000000.0]

27 [486147, 20000000000.0]

29 [494176, 20000000000.0]

3 [327906, 28135808892.0]

30 [532626, 20000000000.0]

Field 1 is date, Field 2 is number of transactions, Field 3 is total gas used on each day of month.

**PLOT 1: GAS PRICE VS DAYS**

Chart, line chart

Description automatically generated

**PLOT 2: NO. OF TRANSACTIONS VS DAYS**

Chart, line chart

Description automatically generated

**INFERENCE:**

From the above graphs we can observe that in Plot1, the gas price drops after fork deployment, which translates to faster transactions since congestion in network increases gas price. From Plot2, we can see that the number of transactions since the deployment on Oct 16th, we can observe a huge spike. The explanation for this is because the Byzantium Fork focused on faster transaction to support business transactions, it brought an increase in network usage and reduced congestion which in turn led to decrease in gas price.

**JOB ID:**

<http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_3679/>

**3.SCAM ANALYSIS: POPULAR SCAMS (15%)**

**PART 1:**

**PROBLEM STATEMENT:** Utilising the provided scam dataset, what is the most lucrative form of scam?

**SOLUTION:**

Here we will use Transactions dataset and the scams.json provided. The question asks us to find the most lucrative form of scam, i.e., we need to find out which category of scam yielded most transaction value in wei.

To tackle this problem our approach was to first read scams.json and get “addresses” and “category” from “result” dictionary and map all addresses to the kind of scam they were associated with. From transactions dataset we map all the “to\_address” to their respective “value”. Now for first reducer, we will find the intersection between the addresses from transactions table and the addresses obtained from the scams.json file. For second mapper, we are mapping each “category” to the intersected addresses’ transaction “value”.

Lastly we are aggregation by passing “category” as key and “value” as value for reducer. Output is stored in partD1.txt.

**OUTPUT:**

"Scamming" 3.833616286244429e+22

"Fake ICO" 1.35645756688963e+21

"Phishing" 2.699937579408742e+22

**INFERENCE:**

Scamming is the most lucrative.

**JOB ID:**

<http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1637317090236_24238/>

**PART 2:**

**PROBLEM STATEMENT:** For the correlation, we could produce the count of how many scams for each category are active/inactive/offline/online/etc and try to correlate it with volume (value) to make conclusions on whether state plays a factor in making some scams more lucrative.

**SOLUTION:**

Here we take the same approach as first problem in terms of extracting required record using “result” dictionary, but this time we also get “status” along with “category” and “address”. Then in reducer we will find the intersection of the addresses involved in scams with address/transaction values in transaction dataset. Then we map all the “status” and “category” to the aggregated “value”. Finally we run the second reducer and aggregate over each “category” / “status”. Output is stored in partD2.txt.

**OUTPUT:**

["Active", "Scamming"] 2.2096952356679117e+22

["Inactive", "Phishing"] 1.488677770799503e+19

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

["Offline", "Phishing"] 2.2451251236751494e+22

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

["Active", "Phishing"] 4.531597871497939e+21

["Offline", "Scamming"] 1.6235500337815102e+22

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

**INFERENCE:**

Active Phishing is most lucrative with highest transaction value whereas Offline Fake ICO is least lucrative.

**JOB ID:**

<http://andromeda.student.eecs.qmul.ac.uk:8088/cluster/app/application_1648683650522_5205>

**4. GAS GUZZLERS (10%)**

**PART1: AVERAGE GAS**

**PROBLEM STATEMENT:**

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

**SOLUTION:**

Here we have taken Transactions dataset and yield gas price and time stamp. We extract month and year and map gas price to all months. Then the value is reduced and aggregated over every month of year. The output is stored in gas\_avg.txt. First field has month and year and second field has average gas used respectively.

**SAMPLE OUTPUT:**

["15", "09"] 56511301521.03343

["15", "10"] 53901692120.53641

["15", "12"] 55899526672.36018

["16", "02"] 69180681134.38885

["16", "04"] 23361180502.73329

["16", "06"] 23021251389.826454

["16", "08"] 22396836435.96108

["16", "11"] 24634294365.28291

["17", "01"] 22507570807.72001

["17", "03"] 23232253600.850437

["17", "05"] 23572314972.035686

["17", "07"] 25460300456.315617

["17", "09"] 30675106219.15495…

**JOB ID:** <http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_3586/>

**PART 2: COMPLEXITY IMPACT**

**PROBLEM STATEMENT:**

Have contracts become more complicated, requiring more gas, or less so?

**SOLUTION:**

Here we used two datasets, i.e., blocks and contracts. From blocks dataset we map each block number to value tuple of time stamp, gas used and a check. From contracts we yield block number and an existence check. Further we reduce the values obtained and find the intersection between two datasets and reduce blocks to their respective value tuple. Another mapper and reducer to average out and obtain average difficulty and average gas used over time. The output is stored in gas\_2.txt. First field has month and year, second and third field has gas and complexity.

**SAMPLE 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]…

**PLOT:**

**JOB ID:**

<http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_3513/>