COURSEWORK: ETHEREUM ANALYSIS

Yuexiang Li
161187811

Yuexiang Li

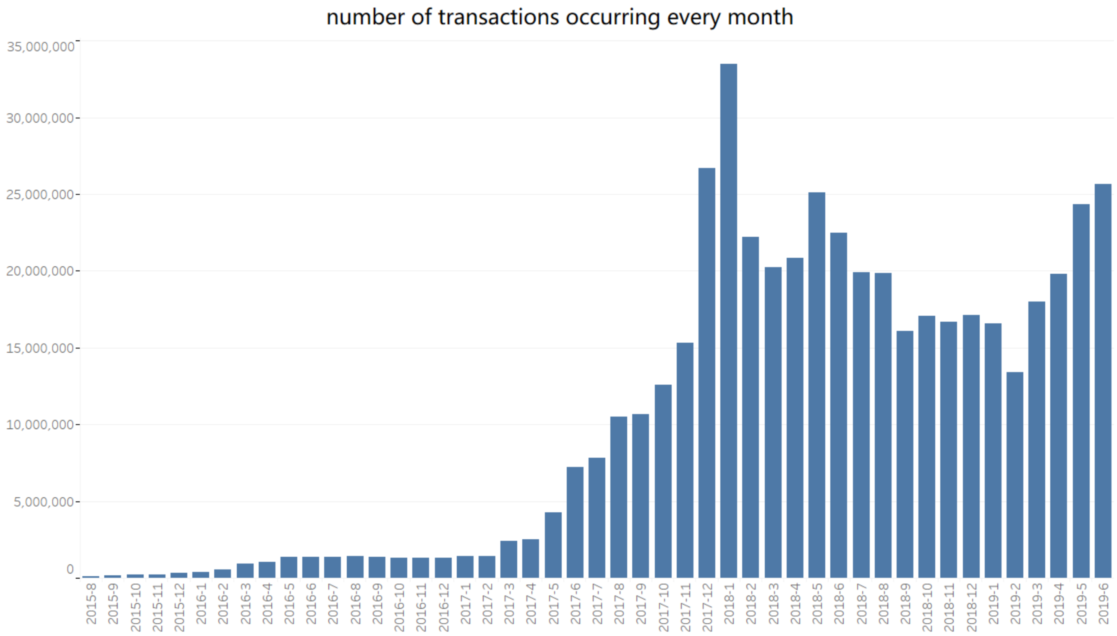
Part A TIME ANALYSIS

1. Create a bar plot showing the number of transactions occurring every month between the start and end of the dataset.

To count the number of transactions occurring every month, we need to extract year and month from /data/Ethereum/transactions in the mapper. Then calculate the number of transactions in every month of each year.

Job ID:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_5257/



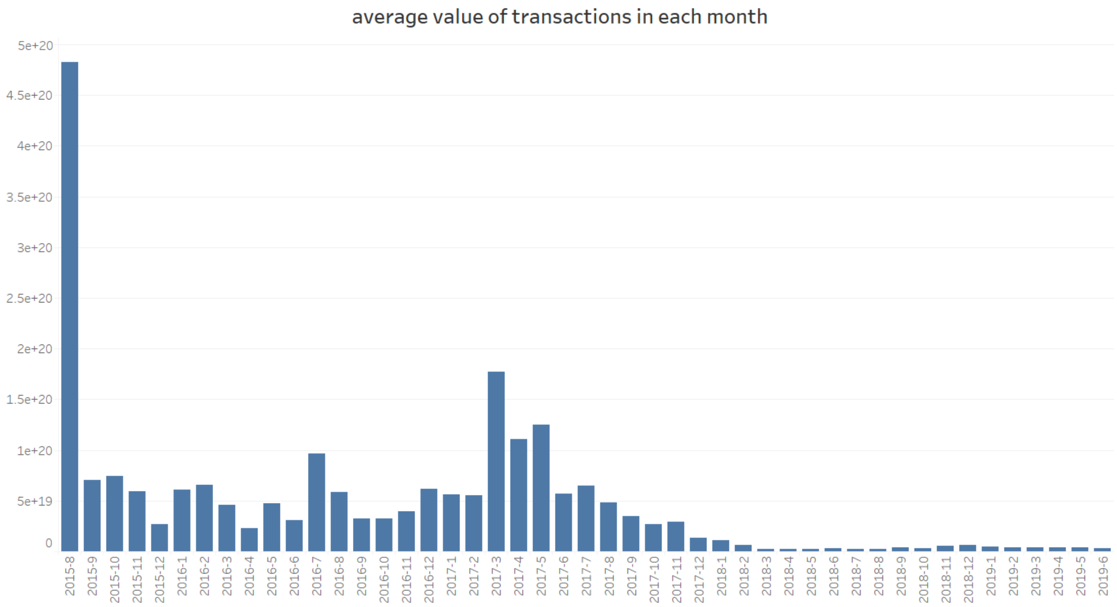
2. Create a bar plot showing the average value of transaction in each month between the start and end of the dataset.

To count the number of transactions occurring every month, we need to extract year and month from /data/Ethereum/transactions and the value of each transaction in the mapper. Then calculate the average value of transactions in every month of each year.

Job ID:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_5278/

Yuexiang Li


average value of transactions in each month
5e+20
4.5e+20
4e+20
3.5e+20
3e+20
2.5e+20
2e+20
1.5e+20
1e+20
5e+19
0
2015-8
2015-9
2015-10
2015-11
2015-12
2016-1
2016-2
2016-3
2016-4
2016-5
2016-6
2016-7
2016-8
2016-9
2016-10
2016-11
2016-12
2017-1
2017-2
2017-3
2017-4
2017-5
2017-6
2017-7
2017-8
2017-9
2017-10
2017-11
2017-12
2018-1
2018-2
2018-3
2018-4
2018-5
2018-6
2018-7
2018-8
2018-9
2018-10
2018-11
2018-12
2019-1
2019-2
2019-3
2019-4
2019-5
2019-6

Part B TOP TEN MOST POPULAR SERVICES

For the MRJob based approach, the first mapper will receive lines from both datasets, /data/ethereum/transactions/ and /data/ethereum/contracts/. We check where each line came from by checking the length of the returned array when the line has been split. The lines from /data/ethereum/transactions/ have 7 fields, whilst the lines from /data/ethereum/contracts/ have 5 fields. Once the line has been distinguished, extract out the information we are interested in and yield it to the reducer. The key for mapper's output is the address, allowing the join to take place on the reducer side. The value contains a field to indicate the source of the information ("value" for /data/ethereum/transactions/, "contract" for /data/ethereum/contracts/).

In the first reducer, iterate around all values yielded for a given key. In each iteration, use the first field in the value to distinguish the information included, adding the value from /data/ethereum/transactions/ and filtering the addresses that was not present within contracts. This reducer will yield contract addresses and the sum value of each address.

The second MapRuduce job will sort pairs from the first one and yield the first ten pairs.

Job ID:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_5394/

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_5421/

Results:

"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

The mapper extracts the information from /data/ethereum/blocks/. And the first reducer will aggregate block size for each miner. Then, the second reducer will sort the result and yield the top ten miners.

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_5444/

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_5445/

Results:

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

Yuexiang Li

Part D DATA EXPLORATION

1. Popular Scams: Utilising the provided scam dataset, what is the most lucrative form of scam? How does this change throughout time, and does this correlate with certain known scams going offline/inactive?

   The most lucrative form of scam:
   To calculate the most lucrative form of scam, we need to aggregate the information from /data/ethereum/transactions/ and /data/ethereum/scams.json. Take the *to_address* and *value* from /data/ethereum/transactions/ and use the field *category* and *addresses* in the scams.json file. Mark the source of the information in the value field of mapper.
   In the first reducer, calculate the total transaction value of each address and yield the address's value with its category. Then, the second reducer will aggregate the value of each category.

   Job ID:
   http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_5544/
   http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_5612/

   Results:

"Scamming"	3.833616286244427e+22
"Fake ICO"	1.3564575668896297e+21
"Phishing"	2.699937579408742e+22
"Scam" 0	

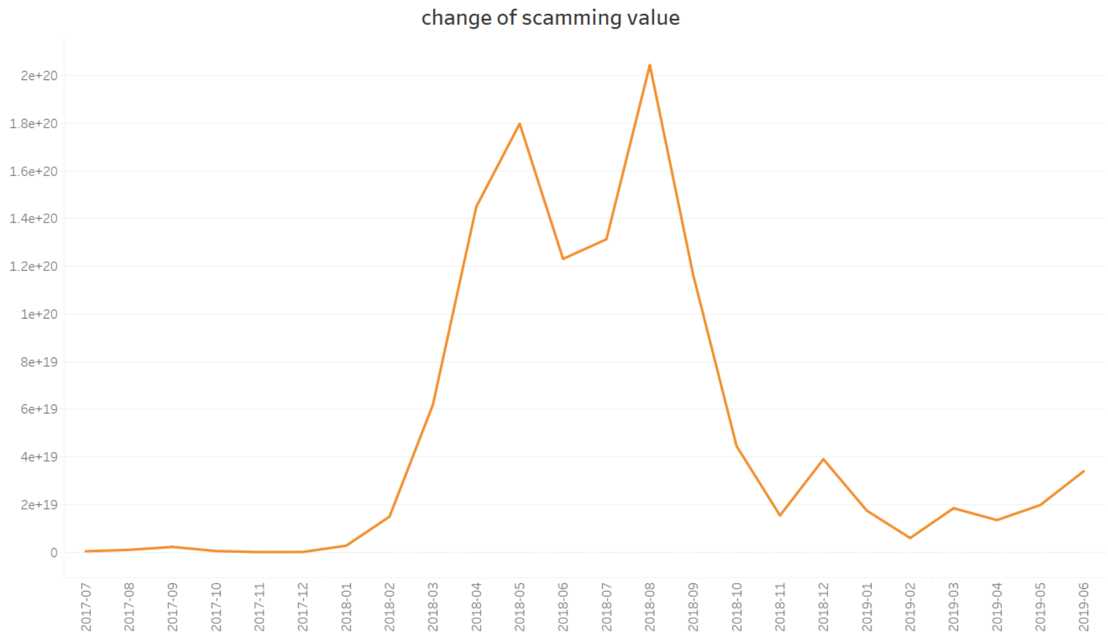
   The most lucrative form of scam is Scamming which had received 3.833616286244427e+22 Wei of value.

   How does this change throughout time?
   In this section, we use the *year_month* and *category* as the key in the first reducer and perform aggregation in the second reducer for each category in each individual month. But we only use the Scamming data to draw the graph.

   Job ID:
   http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_5656/
   http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_5692/

   Results:
   From the following line graph, we can see the transaction value about Scamming increase dramatically during the 2018 and reached the peak in August. Then, it declined sharply. But it showed a trend to increase again.

Yuexiang Li



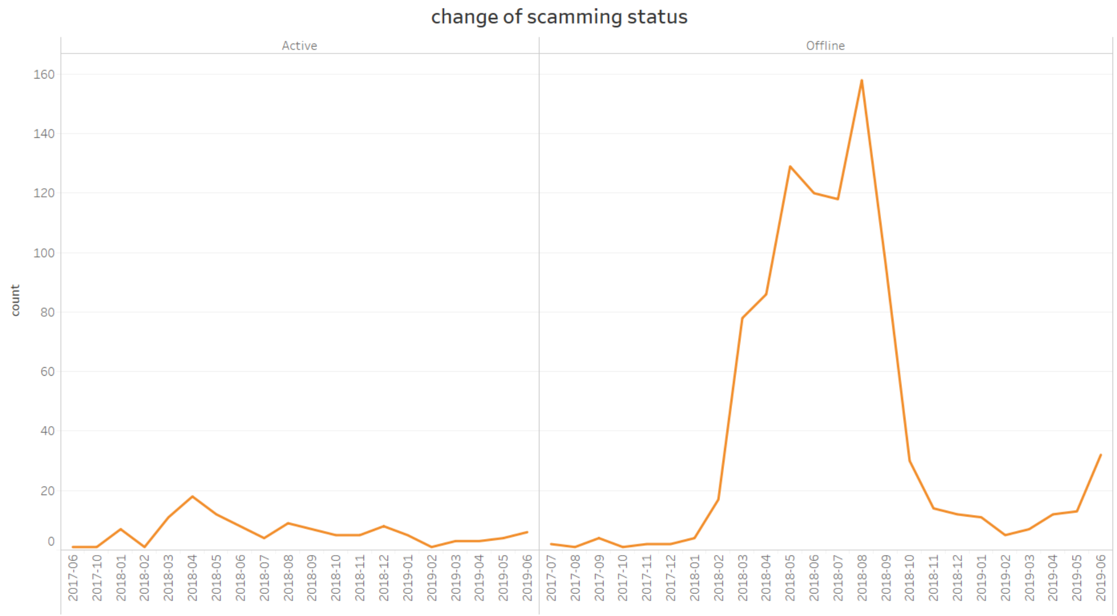
It seems that ETH Scamming was very popular in 2018 and I have found this article: **Ether Cryptocurrency Scammers Made $36 Million In 2018 -- Double Their 2017 Winnings.**

Does this correlate with certain known scams going offline/inactive?
For this part, we extract the status for each category in the mapper. After join operation, we yield category and status in each individual month. Again, we only use data of Scamming to draw the graph. Note that there is a status called *suspended* in Scamming in Feb 2018. We will ignore the data of this category.

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_5731/

Results:
As the graph shown below, we can see that the trend of Scamming going offline is similar to the Scamming value change. I guess it is because the Scammer addresses are detected in time after transactions happened on these addresses.

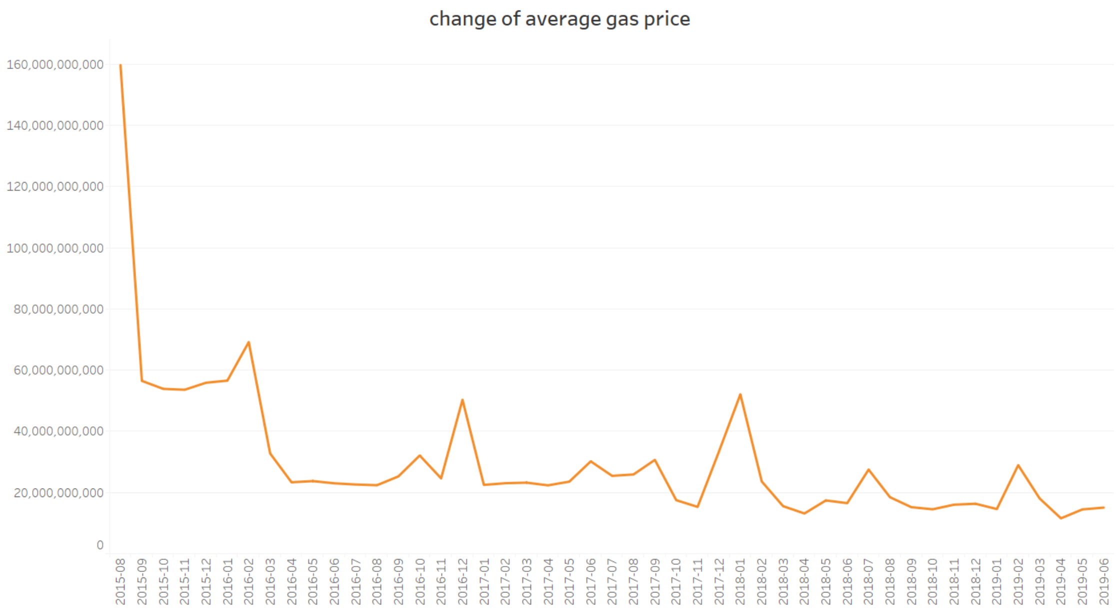
Yuexiang Li



2. MISCELLANEOUS ANALYSIS
Gas Guzzlers:
How has gas price changed over time?
Mapper extracts individual month and gas price from /data/Ethereum/transactions/.
And Reducer calculates the average value of each month.

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1606730688641_7357/

Result:



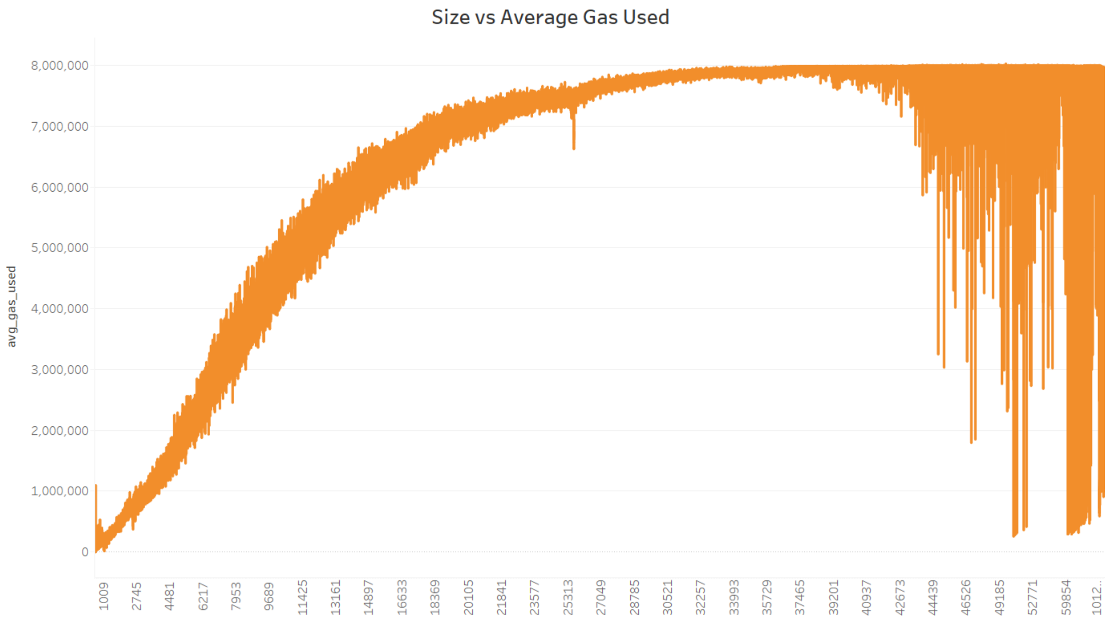
The initial average gas price was very high, but it dropped dramatically.

Have contracts become more complicated, requiring more gas, or less so?
To understand the relationship between contract complexity and gas required, we need to use the *size* and *gas_used* in the data/Ethereum/blocks/.

The Mapper will extract the *size* and *gas_used* from data/Ethereum/blocks/, and the reducer will calculate the average gas used for each block size.

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_2538/

Result:



This graph shows the relationship between the block size and average gas used. There is a clear trend that with the contract becoming more complicated, requiring more gas. And it seems that there is a limitation at 8 million. Notice that the line fluctuates violently when the value approaches the 8 million, with the block size exceeds 42000.

How does this correlate with your results seen within Part B?
Use the addresses from Part B to filter the *address* and *block number* from data/Ethereum/transactions/. And extract *block number*, *size*, *gas_used* from data/Ethereum/blocks/. In the mapper, use the *block number* as the key to join the addresses, the size of the block addresses locating and the gas used for transactions in the address.

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_7148/

Result:

```
"1946708"       ["0xe94b04a0fed112f3664e45adb2b8915693dd5ff3", 3001, 575668]
"1966054"       ["0xfa52274dd61e1643d2205169732f29114bc240b3", 1708, 375125]
"1969372"       ["0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8", 4783, 1101578]
"2462919"       ["0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef", 2310, 464781]
"1428757"       ["0xbb9bc244d798123fde783fcc1c72d3bb8c189413", 13824, 3711215]
"1919996"       ["0x341e790174e3a4d35b65fdc067b6b5634a61caea", 3999, 734742]
"1920419"       ["0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444", 3475, 731981]
"1935363"       ["0xbfc39b6f805a9e40e77291aff27aee3c96915bdd", 1474, 228056]
"2041128"       ["0x7727e5113d1d161373623e5f49fd568b4f543a9e", 1015, 139259]
"2206259"       ["0xabbb6bebfa05aa13e908eaa492bd7a8343760477", 6435, 1353738]
```

The first column is the block that addresses locate, the third is the block size and the fourth is the gas used for contract's transactions. We can conclude that the most popular service addresses are in the smaller size block and need less gas for transactions.

Comparative Evaluation:
Rewrite the Part B in Spark, we obtain the same result with the MapReduce version.

```
yl007@itl400 ~/Desktop/ecs765p/coursework/part_d/MISCELLANEOUSANALYSIS/spark> spark-submit partB_spark.py
20/12/11 11:48:17 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineage doesn't exist or is not writable. Lineage for this application will be disabled.
0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444: 84155100809965865822726776
0xfa52274dd61e1643d2205169732f29114bc240b3: 45787484483189352986478805
0x7727e5113d1d161373623e5f49fd568b4f543a9e: 45620624001350712557268573
0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef: 43170356092262468919298969
0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8: 27068921582019542499882877
0xbfc39b6f805a9e40e77291aff27aee3c96915bdd: 21104195138093660050000000
0xe94b04a0fed112f3664e45adb2b8915693dd5ff3: 15562398956802112254719409
0xbb9bc244d798123fde783fcc1c72d3bb8c189413: 11983608729202893846818681
0xabbb6bebfa05aa13e908eaa492bd7a8343760477: 11706457177940895521770404
0x341e790174e3a4d35b65fdc067b6b5634a61caea: 8379000751917755624057500
time: 108.434192181
```

Remove printer, and execute the job 5 times. The average performing time is (107.09+110.30+131.63+127.00+116.37) ÷ 5 = 118.478 seconds.

```
yl007@itl307 ~/Desktop/ecs765p/coursework/part_d/MISCELLANEOUSANALYSIS/spark> spark-submit partB_spark.py
20/12/09 11:08:42 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineage doesn't exist or is not writable. Lineage for this application will be disabled.
107.094811916
yl007@itl307 ~/Desktop/ecs765p/coursework/part_d/MISCELLANEOUSANALYSIS/spark> spark-submit partB_spark.py
20/12/09 11:10:32 WARN cluster.YarnSchedulerBackend$YarnSchedulerEndpoint: Attempted to request executors before the AM has registered!
20/12/09 11:10:32 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineage doesn't exist or is not writable. Lineage for this application will be disabled.
20/12/09 11:10:43 WARN nio.NioEventLoop: Selector.select() returned prematurely 512 times in a row; rebuilding Selector io.netty.channel.nio.SelectedSelectionKeySetSelector@6de732a0.
110.303998947
20/12/09 11:12:18 WARN nio.NioEventLoop: Selector.select() returned prematurely 512 times in a row; rebuilding Selector io.netty.channel.nio.SelectedSelectionKeySetSelector@74c232a4.
yl007@itl307 ~/Desktop/ecs765p/coursework/part_d/MISCELLANEOUSANALYSIS/spark> spark-submit partB_spark.py
20/12/09 11:12:30 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineage doesn't exist or is not writable. Lineage for this application will be disabled.
131.628465891
20/12/09 11:14:35 WARN nio.NioEventLoop: Selector.select() returned prematurely 512 times in a row; rebuilding Selector io.netty.channel.nio.SelectedSelectionKeySetSelector@48595d2.
yl007@itl307 ~/Desktop/ecs765p/coursework/part_d/MISCELLANEOUSANALYSIS/spark> spark-submit partB_spark.py
20/12/09 11:14:52 WARN cluster.YarnSchedulerBackend$YarnSchedulerEndpoint: Attempted to request executors before the AM has registered!
20/12/09 11:14:52 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineage doesn't exist or is not writable. Lineage for this application will be disabled.
127.004572868
yl007@itl307 ~/Desktop/ecs765p/coursework/part_d/MISCELLANEOUSANALYSIS/spark> spark-submit partB_spark.py
20/12/09 11:16:58 WARN cluster.YarnSchedulerBackend$YarnSchedulerEndpoint: Attempted to request executors before the AM has registered!
20/12/09 11:16:58 WARN lineage.LineageWriter: Lineage directory /var/log/spark/lineage doesn't exist or is not writable. Lineage for this application will be disabled.
116.370979786
20/12/09 11:18:47 WARN nio.NioEventLoop: Selector.select() returned prematurely 512 times in a row; rebuilding Selector io.netty.channel.nio.SelectedSelectionKeySetSelector@2082be2d.
20/12/09 11:18:47 WARN nio.NioEventLoop: Selector.select() returned prematurely 512 times in a row; rebuilding Selector io.netty.channel.nio.SelectedSelectionKeySetSelector@3277cc41.
```

While the MapReduce version takes 36 minutes to execute two jobs for finding the most popular service.

2020.12.07 14:22:19 GMT	2020.12.07 14:22:28 GMT	2020.12.07 14:25:45 GMT	job_1606730688641_5421	streamjob1983345379554260680.jar	yl007	root.users.yl007	SUCCEEDED	8	8	2	2	00hrs, 03mins, 16sec
2020.12.07 13:49:02 GMT	2020.12.07 13:49:07 GMT	2020.12.07 14:22:15 GMT	job_1606730688641_5394	streamjob1228587133553459691.jar	yl007	root.users.yl007	SUCCEEDED	1104	1104	2	2	00hrs, 33mins, 08sec

For this task, Spark Framework is better than MapReduce. Because spark performs in-memory processing to avoid unnecessary I/O operations.