ECS640U CourseWork
Ibrahim Elmagbari

Part A - Time Analysis

JOB - Number of monthly transactions

JOB ID: http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1604349877093_2264/

I implemented this by using MRJob, python API for Mapreduce.
This task asked for the number of transactions occurring every month for the dataset. I implemented this by taking the transaction dataset as input. In mapper line by line is passed in and is split by comma, because this is how the data is formatted. Then I check if there is any malformed data/line so as to not use this data by checking the length of line when split.

Then I extract the time field to get month and year and emit this along with a count of 1 to indicate for a given month there has been 1 transaction. This is to keep count. This is repeated for all lines in the transaction dataset.

A combiner is used to add efficiency. Both reducer and combiner aggregate the count for a given month at a given year. The month and year is the key to which the shuffle and sort brings together all same months of a given year. Shuffle and sort also has the list of values for the given key. This list is the count which is then summed. Each month at a given year is then emitted along with total number of transactions for that month as output
The bar plot can be seen below:

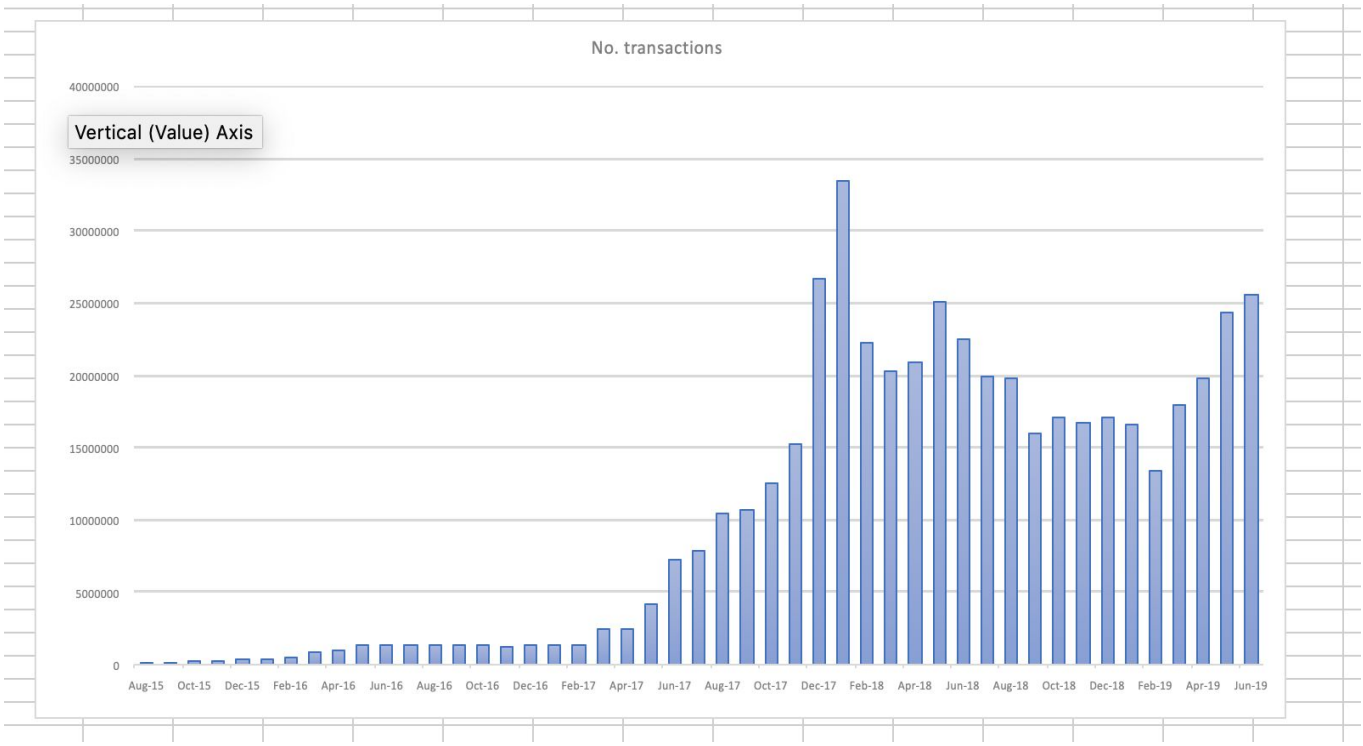


Figure 1

Job - Average value of transaction for each month

Job ID: http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1604349877093_2308/

I implemented this by using MRJob, python API for Mapreduce.
This task included getting the average transaction for a given month in a year. Its very similar to the number of monthly transactions job above. However this time I emitted the month as the key and a pair of count and money as a tuple for the value in the mapper phase.
This allows for shuffle and sort to bring together all the count and money values for a given date.
The combiner is used to add efficiency. It is like a mini reducer but it does it work on the mapper node.
Both the reducer and combiner then did the calculation of the average which is then outputted for a given date.

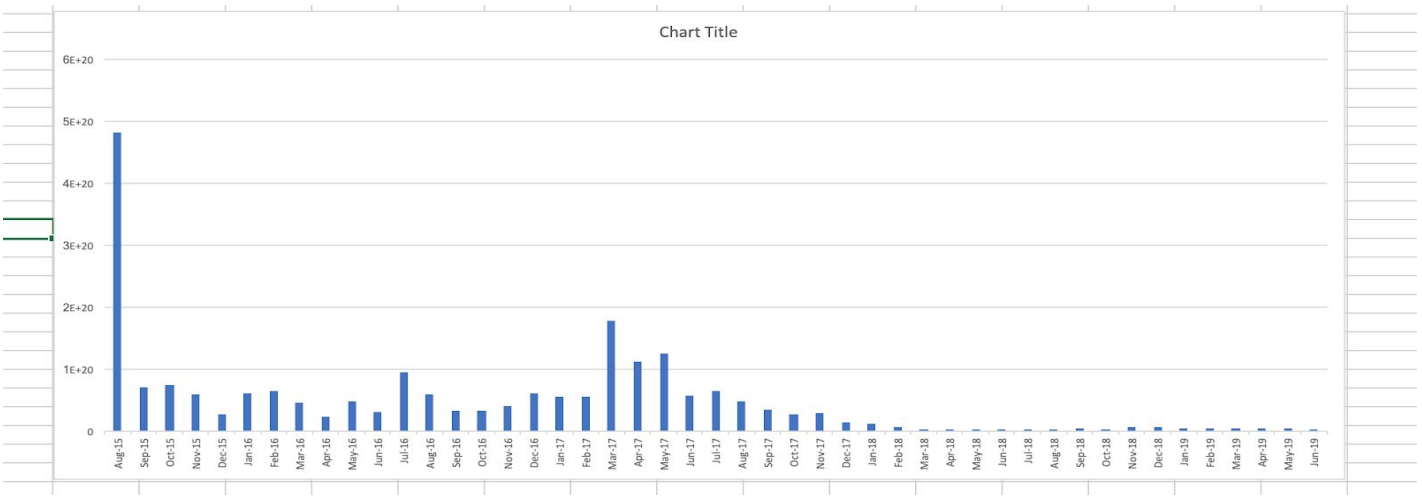


Figure 2

Part B - Top Ten Most popular services

JOB1- Initial Aggregation

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1604349877093_2518/

I implemented this by using MRJob, python API for Mapreduce.
The task is to aggregate the value/money for each address. In the map phase I filtered and chose the fields I needed which is address and money for transactions. I then emitted this with the key being the addresses so shuffle and sort can bring together all the keys of the same addresses and their values to then aggregate the money. The combiner was used for efficiency. The reducer and combiner then both summed up the money for each key. The combiner did this at the mappernode before the data was sent to the reducer node. The reducer then made sure to complete this as it has all the key and their value pair. Output is then the unique address and aggregate money for that address

JOB 2 - Joining transactions/Contracts and Filtering

Job ID: http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1604349877093_4086/

I implemented this by using MRJob, python API for Mapreduce.
In this task I joined 2 datasets, the output data from job 1 above and contracts dataset. This is a repartition join so,this join was done on the reduce side.
In the mapper phase I differentiate what line the dataset comes from by the number of fields for a line. The contract has five fields and output from job1 has 2 fields. Any malformed data is ignored. This is done through try and except.

So, I then get the key which I wish to join the both datasets. This is the address. I emit this along with the value which is a tuple pair of value/money along with a number 1 or 2. The 1 or 2 number is used to tell us where the dataset came from. In the reducer 1 tells us the data came from output of job and 2 tells us it came from the contracts dataset.

At the reducer the key address and all values are together(to one address). I then iterate through to differentiate which data set it came from. If it came from both datasets I emit the address and money. This makes sure that the transaction was a contract. As transactions can be other than a contract. Now we have the contract transaction and value money for each as the output.

JOB 3 - TOP TEN

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1604349877093_4122/

I implemented this by using MRJob, python API for Mapreduce.
In this task I took the now filtered addresses. Which is all the addresses which are contracts as input and passed it into the mapper phase.
To add efficiency since I want the top 10 contracts by value of the amount of money. I discarded all the data where money is 0 in the mapper phase. Anything greater passed through to then be emitted with nothing as key as we do not need to shuffle and sort this time or do any join.

The combiner then gets all the data in a particular node passed through at once. Both combiner and reducer then sort the values with greatest to lowest and stopping after we do 10. Hence you see the break in the code. Top ten contracts by amount of money/value is then output at the end. The reducer specifically looks at the top ten from each mapper to choose its top ten.

Part C - Top Ten Most Active Miners

JOB IDs:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_2932/
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_2934/

Note: there are 2 jobs because of the use of MRStep to allow a job to run consecutively.

I implemented this by using MRJob, python API for Mapreduce. I also used MRStep which allows for one job to be followed by another as many times as you want.
In the first job the map phase takes the block dataset specifically the fields address and size. The address being the key so all values for a key are put together in shuffle and sort after mapping phase. The combiner and the reducer then take a given key which is address and aggregate the size of the block. Combiner is used for efficiency since its code runs on the mapper node. The output is the aggregate of the size of the block for a given address.

This is then passed to job 2 which also has the mapping phase. This time there is no key so all data is passed through from mapper to reducer at once, with no shuffle or sort. The reducer then takes all the data and sorts it in terms of size of block from greatest to lowest. Only gives a top ten since there is a break in the code. This top ten is then outputted. So we now have the top ten addresses of miners by the size of total block mined.

PART D - Data Exploration

For part D of the coursework I decided to do Scam Analysis, Machine learning and all three of Fork the chain, Gas guzzlers and comparative evaluation which are part of the Miscellaneous Analysis.

Scam Analysis - Needs both me and josephs parser.

Note There are 3 sections each with its own code file.
For scam analysis there are three code files. In this report each section refers to a code file. So there will be three sections.
The first implementation gets all scam addresses and aggregate of money/ether to that scam address. This is section 1

Next implementation gets the top ten scam addresses with the aggregate of ether/money sent to it. So the most lucrative scams. This is section 2.

The next code file then does an implementation that gives you the most lucrative scam for each month. This is section 3.

Also PLEASE note ALL sections require a parser to convert from JSON to csv format (given to us by Joseph) followed by my own parser to remove empty strings between data in each line in this csv file.

Section 1- All Scams and subsequent aggregate of ether/money associated with that Scam address

I implemented this by using MRJob, python API for Mapreduce. I also used MRStep which allows for one job to be followed by another. In this specific task I had it do three jobs in total.

So, we start by doing a replication join. This is a join done on the mapper side. The scam.csv file is used as the small dataset which is stored in memory at all nodes where mapping is done. This is allowed because it is a small file. This data is stored in a dictionary with the key being the scam ID and join value being a list of all the scam addresses associated with that scam ID. This is the first stage of mapping.

In the next mapping phase we take in input that is the big transaction dataset specifically looking at the to_address field. This is the field where the join occurs. With an inner join of scam address and to_address of transaction file. The join occurs because we compare the address of from transaction dataset to the dictionary which is the from scams.csv file. Since the scams.csv file had a number of addresses for each scam. It could be a list. I had to take this into account and use this code to iterate through the list in the dictionary so comparison can be done with all scams addresses associated with scam and to_address part of transaction dataset. The code line is seen below
if join in [x for v in self.sector_table.values() for x in v]:

After this another mapper is done. This is an intermediate and is only needed because MRstep requires a mapper and reducer in pairs. So it changes nothing just emits the same key value pair that arrived at it.

At the reducer all the same scam addresses are joined together with all its values/amount of money. This is needed so we then aggregate this value to each scam address.
This gives an output of all scam addresses and associated money/value sent to each scam address.

Section 2 - Top Ten Scams (Most lucrative)

In this part we take the in as input the output of the job from section 1. So the scam addresses and sum of money/ether for a scam address. We now use this dataset to find the top ten in terms of amount of money associated to a scam address. This is similar to the top ten code of job 3 part B. However this time the data is sent through all at once and there is no key to reduce/join by.

Section 3 - Top scam for each Month

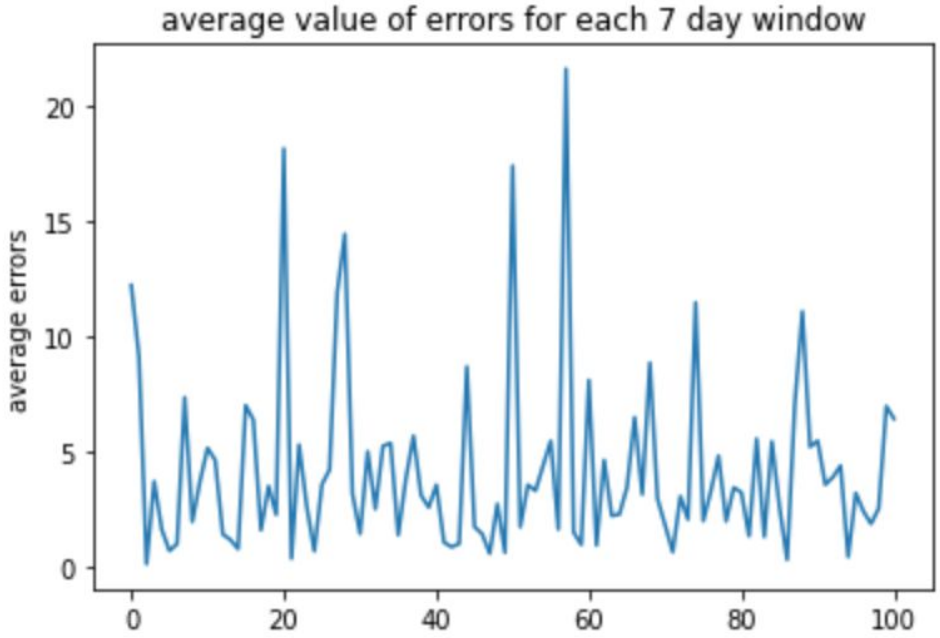
So in this implementation we start off similar to section 1. We do a replication join between both datasets transactions and scams.csv. In the same instance the join occurs with addresses from both datasets. However this time we will not just yield the address. We add the year and month along with address as the key as seen here:
join_key = month + "," + addr
This allows me to reduce by a key involving 2 values. The month variable in the code includes the year and month as a string. We then emit the key and money associated with the key. So in other words the key which is the date of transaction along with address is emitted along with the value money/ether for a given date of transaction.

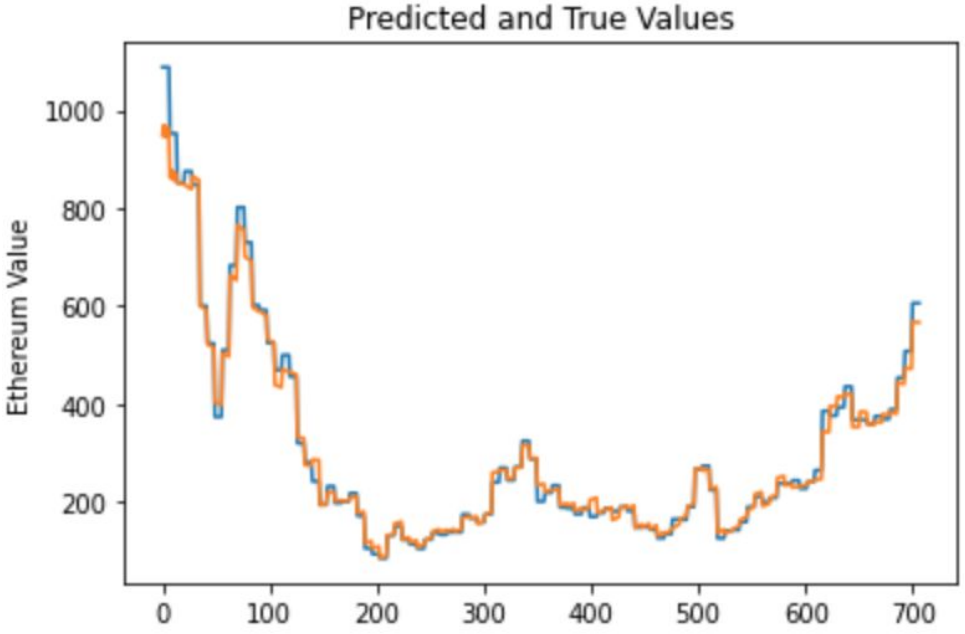
We sum the money/ether for a given key. We then have an aggregate of money for a given address for a given month. This is then emitted to the last job. MRStep final step. In this step we get back the month and yield this along with value pair of addresses and aggregated money. We then do a top ten job similar to what is seen in PART B and Part C. With the sorting of values based on the amount of money. In the end we then emit the date, scam address and aggregated money associated with that address. This can be summed as saying the output is the Top scam specifically beneficiary address for a given month and amount he profited.

Machine Learning

This task was carried out in google colab using python pandas. The first task was to prepare the training dataset. with a dataset that consisted of the price of Ethereum from 2015 to 2020. For this dataset we are only forecasting the prices of Ethereum as we are dealing with univariate data using an Auto Regression Integrated Moving Average (ARIMA) model to train the data. ARIMA models make use of both Auto Regression and Moving Averages Model to learn the impact of both autocorrelations and predictions in time value series, which is appropriate for this dataset. The prediction using the ARIMA model was made on 7-day windows. We then iterated through this window so that each prediction was stored in a table, as well as the date and real price, so we can compare the predictions to the real values and test the accuracy. I used a dataframe to then display the results of my training model in comparison to the actual values the dataset consists of which are presented below in both a table format and as graph to measure the accuracy of the prediction. Studying the visualisation methods concluded that the forecast was accurate past June 2019, however, became less accurate when nearing the end of 2020.

We calculated the average errors for each 7 day window by dividing the sum of errors by the length of errors. This would then show us the errors between the training model and the actual prices shown in the dataset. This would give us a rough understanding of how accurate our model is before we actually plot the predictions vs actual value.


average value of errors for each 7 day window
average errors
0
5
10
15
20
0
20
40
60
80
100


Predicted and True Values
Ethereum Value
200
400
600
800
1000
0
100
200
300
400
500
600
700

Miscellaneous Analysis

Fork the Chain

A cryptocurrency fork is an update to the software governing the distributed network that makes existing rules either valid or invalid. A fork occurs when the user base or developers decide that a fundamental change is needed for the cryptocurrency. This can be due to a fundamental disagreement within the community or due to a major hack as was the case with Ethereum.

The major hack happened in june 2016 where users exploited a vulnerability in the DAO code. DOA stands for decentralized autonomous organization. 3.6 million ether was stolen because of an abuse of a loophole. This lead to a major decision to be made. To either do nothing or close the loophole and restore the funds. At first the development team voted to do a soft fork. However this was discarded due potential security flaws.

A soft fork is a change to the software protocol where only the previously valid block/transactions are made invalid. A hard fork in comparison is when nodes of the news version of the blockchain no longer accept the older version of the block leading to a permanent divergence from the previous version of the block chain.

So in the end a hard fork is what happened to Ethereum. This was to reverse the funds that were stolen and prevent further security breaches. This however was controversial with many people not agreeing with a hard fork as it violated the notion of being decentralized. Seeing the developers as having had too much involvement. This then led to a fork in ethereum where the original unforked block chain called Ethereum classic was separate from Ethereum. In conclusion they are separate block chains and each being its own cryptocurrency.
Ethereum has a market cap of 19 billion dollars and the original Ethereum classic has a market cap of 531 million dollars.
Note that most popular is ethereum, with ethereium classic remaining in the shadows of etheruem with a small number of hardcore loyal supporters mostly left it is estimated that around 10% of users remained. Most made the switch to Ethereum.

So if we look at the price of ether around june and july time. We can see at the time of the hacking in june there was a sharp rise but subsequent fall. This mostly attributed to confusion and excitement to what has occurred which lead to a steep rise because of interests of people. But over time as the the news became clear of the hack it then led a steep downfall. On july 20 the decision was made to do a hard fork. This caused an increase to $14.50 as people wanted to be involved the experiment. A 33% increase.
Further down the line this led to a stabilisation of the currency as can be seen by the graph over the coming months. The peaks and the trough being 22 dollars and 9 dollars respectively at the time of June to July. Gives u the picture of what happens before and after Fork changes.

Another example of a fork includes the Byzantium hard fork which was implemented on October 16, 2017 at block 4,370,000. According to Investopedia, the fork consisted of nine EIP's designed to enhance the privacy, scalability and security of Ethereum. This hard fork never led to a split unlike the fork discussed before. With no significant community argument arising.
Now the prices of ETH remained relatively stable both before after the forks execution, it rose around 40 dollars soon after the execution. On release day due to the uncertainty many traders waited rather than make investments, in case anything goes wrong. This can be seen on the graph below.

This addition of new features set the stage for further investments and stability in place attracting more users to use ethereum leading to increase in the network usage. Ethereum is set to have a bright future.

So after all this research is done I did some implementation code to do analysis based on this research. All the datasets used below were given to use and are on the hadoop distributed file system.

Firstly using the transaction dataset and Map/Reduce framework I was able to get the number of monthly transactions for that year of 2017. Year of 2017 is when the Byzantium hard fork was done.

Looking at figure 1 in Part A (Time analysis) we see that 2017 was a year with an increase in the number of transactions month by month. There was a 8x increase in transactions from the start of year 2017 compared to the end of that same year.

Looking at the month of october onwards of 2017 we see a sharp increase of transactions which is a good indicator of general usage. The rate of increase was greater after the fork in comparison to before the fork. So we can conclude it was a successful Fork in the short term.

Digging a little deeper I wrote code that was able to get the number of daily transactions for the month of october. This can give us a more detailed picture. The result of this can be seen in the graph below. Also for comparison sake there is also a graph of the number of daily transactions over a few years(change to exact no of years!)

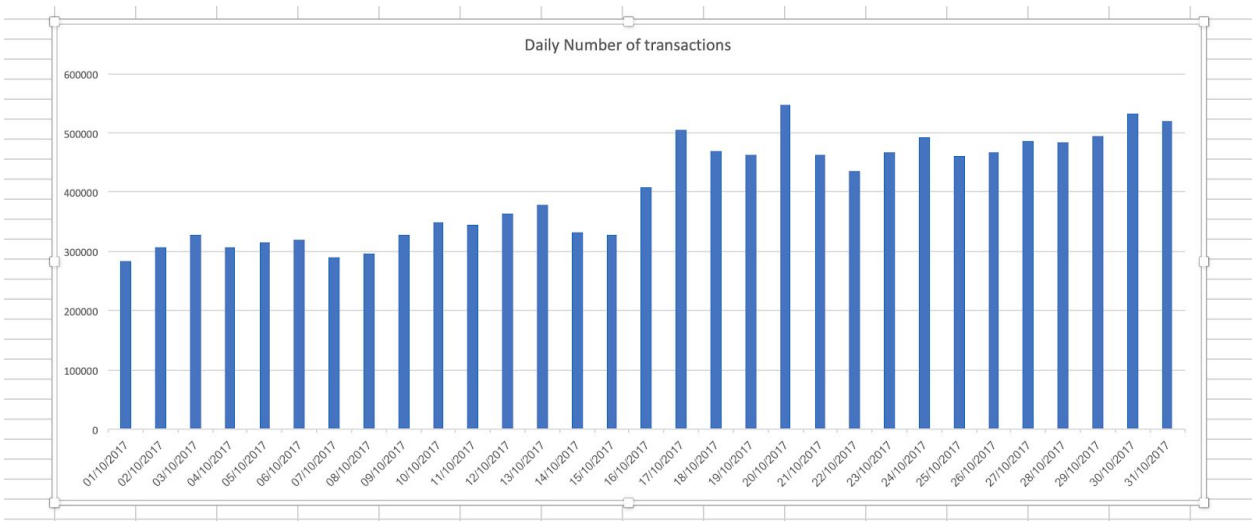


Figure 3 - Daily number of transactions for october 2017.

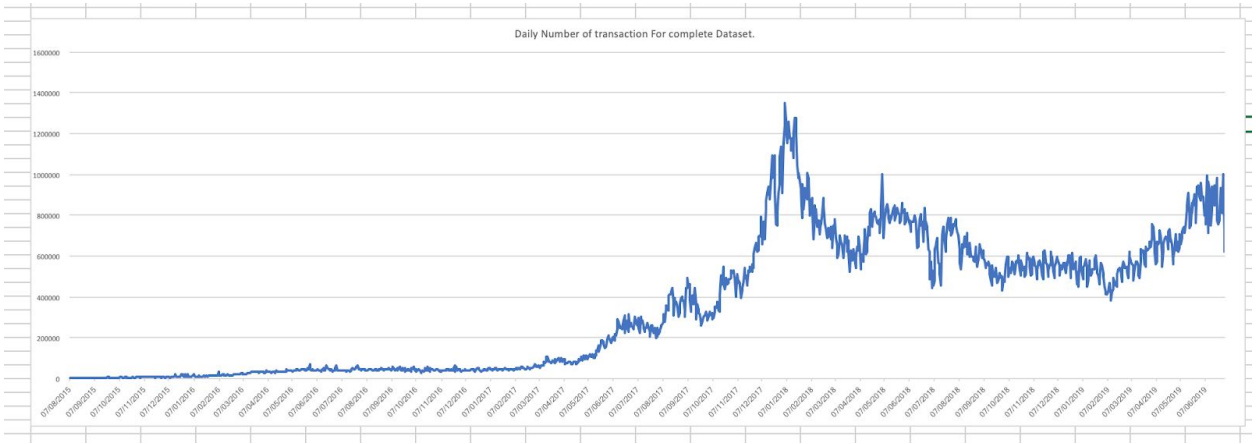


Figure 4 -Daily number of transactions for the whole dataset. From 2015 to 2019

Figure 3 and 4 back the conclusion that the Byzantium hard fork was a success and the catalyst for increase usage over the coming months to peak levels.

Lastly I was able to write a script to see who profited the most during this period of time. By looking at the top ten addresses which received the most ether/money in that month of october 2017. This can be seen in the table below:

Top 10 Addresses that profited the Most	Ether/money transferred to that address

at time of Fork.	in month of october 2017
0x7727e5113d1d161373623e5f49fd568b4f543a9e	2718625397579823314263045
0xf4b51b14b9ee30dc37ec970b50a486f37686e2a8	994608971610000000000000
0x22b84d5ffea8b801c0422afe752377a64aa738c2	981005017110000000000000
0xe94b04a0fed112f3664e45adb2b8915693dd5ff3	883776796936846657069409
0xc257274276a4e539741ca11b590b9447b26a8051	819897340283634897686000
0xfa52274dd61e1643d2205169732f29114bc240b3	6707564151793931365942 82
0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8	624365319318700022473757
0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef	551148777950749000000000
0x3f5ce5fbfe3e9af3971dd833d26ba9b5c936f0be	464666027699541188726491
0xe49b8bfc949a678eaa8cec84f2babc8cfb495fc4	441018804460156110846444

.

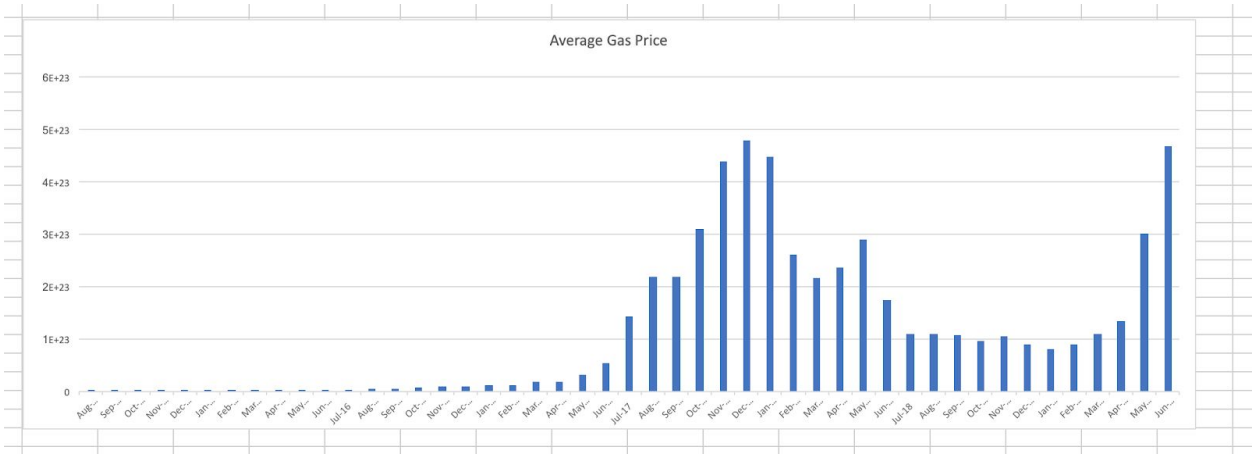
Gas Guzzlers

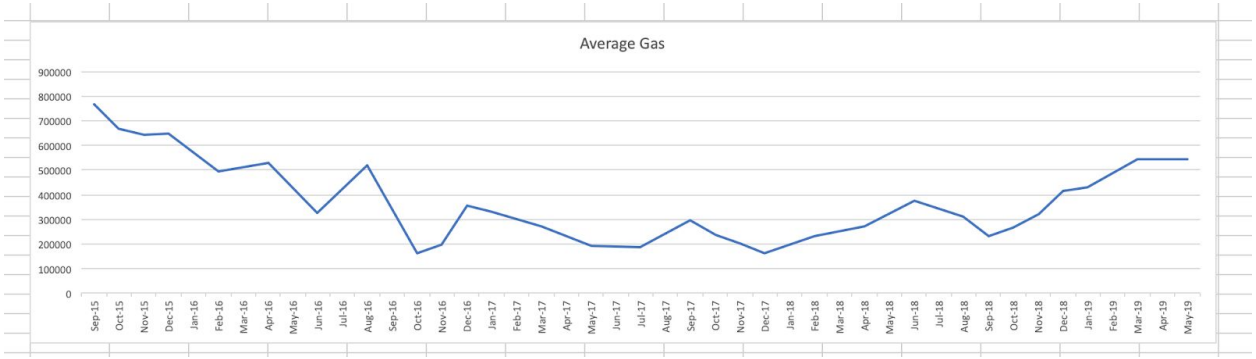
Gas Guzzlers: For any transaction on Ethereum a user must supply gas. How has gas price changed over time? Have contracts become more complicated, requiring more gas, or less so? How does this correlate with your results seen within Part B. (10/50)

2 separate code files were needed to complete gas guzzlers. The first gets the average gas price over each month using the transaction dataset.

The second gets the average gas for contracts over each month. In order to do this I used both transactions and contracts data sets. A repartition join was done (reduce side join) to do this. Then using MrStep I was able to get the average gas of each contract for a given month.

Using the output data of the implementation of code I was able to construct various graphs seen below





Transaction fee is equal to gas price multiplied by gas. This is a fee you must pay miners for a transaction.
IF we look at figure 2 in Part A job 2 with the result of multiplying average gas and average gas price. We see clearly that as the average value of transaction increases or decreases the transaction fee follows suit. The trend is extremely similar which shows a clear correlation between the average transactions and the data averages from the results of gas guzzlers.

Comparative Evaluation

For this section I repeated the Hadoop map/reduce approach 5 times. I also completed the implementation using Spark framework and repeated the Spark job a further 5 times. This is so I get a clear average of the speed it takes to complete part B which is finding the top ten most popular services. The results are seen below:

Running part B- Top ten most popular services	Hadoop Map/Reduce (Total time)	Spark
Run 1	58 minutes 34 seconds	6 minutes 58 seconds
Run 2	49 minutes, 12 seconds	3 minutes 27 seconds
Run 3	72 minutes, 10 seconds	10 minutes 23 seconds
Run 4	78 minutes, 47 seconds	8 minutes 44 seconds
Run 5	63 minutes, 38 seconds	8 minutes 19 seconds

Using the results table (median/average):

- The median time is calculated to be 63 minutes for Hadoop Map/reduce job (rounded to the nearest minute).
- The average time for Hadoop Map/reduce job is 64 minutes (rounded to nearest minute)

- The median time is calculated to be 8 minutes for Spark framework.
- The average time is calculated to be 7 minutes for Spark Framework.

<u>Analysis of Findings:</u>

The results are as I expected with hadoop Map/reduce framework implementing task B in a much slower time on average then Spark. The difference is significant with the Hadoops Map/reduce fastest time being roughly 39 minutes slower then Spark Framework slowest time. The median and average being close tell us there is little extremity in the results data. However I have to take into account the cluster had a lot of jobs running on it and this may have skewed the results and introduced a bias either way. At one point there were 70 jobs running on the cluster.

However if we look at both frameworks we can conclude the data is correct. The hadoop Map/reduce Job is supposed to be used batch processing and was not created to be fast. This is because there is I/O operations which involve reading from disk and writing to disk. This read operation is relatively slow, as well as the writing operation to disk. Now if we look at Spark Framework. It is fast because it is able to process everything in main memory (RAM). Accessing main Memory is considerably faster than disk access. Also spark has fewer I/O operations which give it the advantage in terms of speed to the point where it is used for real time analytics for data.It must be said that hadoop does have the advantage of working with large datasets compared, but is evidently faster than Hadoop Map/Reduce framework.

In conclusion, Spark framework is the best approach to undertake part B task compared to Map reduce. The main reason being the speed. Another reason is the ease of use with Spark having a friendly API to use.