****Timely submission badge. This report was submitted before 9am on Monday 3rd December 2018.*

ECS640U Big Data Processing

Analysis of Bitcoin Transactions

UZAIR ISHAQ

Part A: Time Analysis:

Part A asks us to find the number of bitcoin transactions that occurred each month and plot a bar graph.

This requires using the transactions.csv file which has the following fields:

tx_hash: The unique id for the transaction

blockhash: The block this transaction belongs to

time: The time when the transaction occurred

tx_in_count: The number of transactions with outputs coming into this transaction

tx_out_count: The number of wallets receiving bitcoins from this transaction

Writing this job in Map Reduce only requires the **time** field as we need to find the number of transactions in every month.

Mapper:

In the mapper I first check the line being mapped to make sure its not malformed.
Then I split the line at every "," to extract the month and year from the unix timestamp field of that line. Finally I output the month and year as a key and 1 as the value (representing one transactions for that that month in that year).

Reducer:

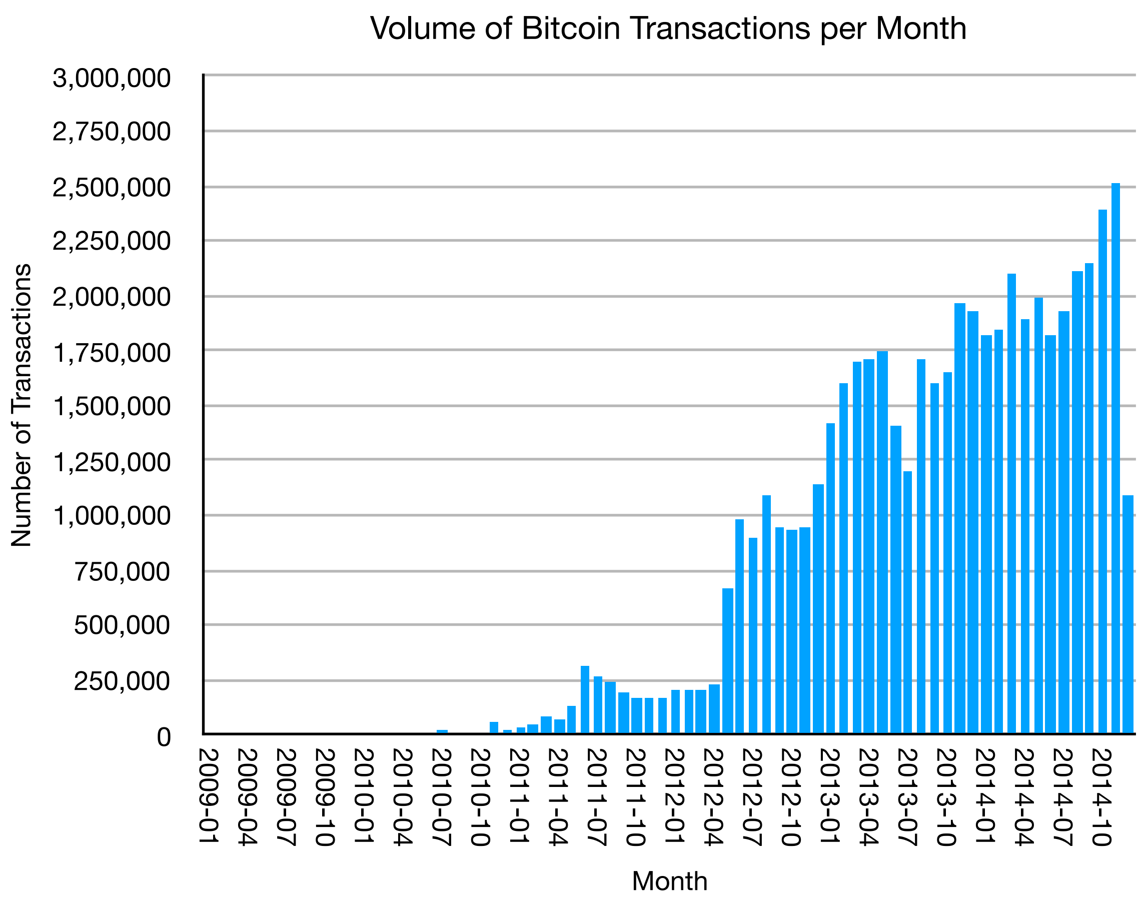
The reducer receives each month in each year as a key and a list of 1's as the values. The reducer sums all the 1's emitted by the mapper for each month year grouping and outputs the month year followed by the number of transactions that occurred in that specific month in that specific year.

Combiner:

The combiner performs the same function as the reducer; it receives the same key as the reducer and a subset of the values. However it runs straight after the mapper to aggregate the partial values so less data needs to be transferred to the reducer. As this is a counting job, the combiner makes it more efficient.

Results:

1970-01	1	2010-03	5391	2011-06	317278	2012-09	945958	2013-12	1935103
2009-01	2575	2010-04	9629	2011-07	267302	2012-10	936158	2014-01	1817513
2009-02	3417	2010-05	6215	2011-08	236416	2012-11	944891	2014-02	1846258
2009-03	3484	2010-06	6675	2011-09	194789	2012-12	1145572	2014-03	2100199
2009-04	3459	2010-07	26486	2011-10	168939	2013-01	1425459	2014-04	1890396
2009-05	3404	2010-08	11968	2011-11	169012	2013-02	1597550	2014-05	1994475
2009-06	2241	2010-09	13184	2011-12	172435	2013-03	1697441	2014-06	1817151
2009-07	1931	2010-10	14398	2012-01	199876	2013-04	1706496	2014-07	1930528
2009-08	1570	2010-11	63408	2012-02	208362	2013-05	1746398	2014-08	2113922
2009-09	2170	2010-12	17142	2012-03	203140	2013-06	1407729	2014-09	2143632
2009-10	2141	2011-01	34900	2012-04	225083	2013-07	1206800	2014-10	2393129
2009-11	2232	2011-02	47168	2012-05	660323	2013-08	1714751	2014-11	2512559
2009-12	4084	2011-03	83167	2012-06	985629	2013-09	1596635	2014-12	1094574
2010-01	5056	2011-04	73865	2012-07	900495	2013-10	1647771		
2010-02	5751	2011-05	136494	2012-08	1097563	2013-11	1961108		



This data clearly shows us that utilisation of bitcoin has been increasing over the period of this data set. More specifically the number of transactions were increasing consistently from 2009 to 2011, and then was increasing more rapidly for the first half of 2011. The volume of transactions was then fluctuating between a consistent range until mid 2012 when the number of transactions per month doubled and then was increasing consistently until the end of the dataset.

Part B: Top Ten Donors

Part B asks us to find the 10 largest donors to the WikiLeaks public key.

I have decided to implement this in spark as doing so in map reduce would require multiple independent jobs.

To obtain the donors we need to use the **vout.csv** and **vin.csv** files.

The **vout** file contains:

> **hash**: The associated transaction
>
> **value**: The amount of bitcoins sent
>
> **n**: The id for this output within the transaction. This will equal the **vout** field within the vin table above, if the coins have been respent
>
> **publicKey**: The id for the wallet where the coins are being sent

The **vin** file contains:

> **txid**: The associated transaction the coins are going into
>
> **tx_hash**: The transaction the coins are coming from
>
> **vout**: The ID of the output from the previous transaction - the value equals n in vout below

Implementation in python:

1. We need to filter the **vout** file to only get rows containing the WikiLeaks bitcoin address.

   In pysaprk I call the **textFile**() method to load the **vout** file and then use the **filter**() transformation with a function I defined to remove all rows which do not contain the WikiLeaks bitcoin address.

2. Then we need to join (inner join) this to the **vin** file. The join will use the **hash** field from the **vout** file and the **txid** field from the **vin** file as the join key. This gives us all the transactions that were sent to the WikiLeaks address and the transactions they came from.

   This is done by calling the **map**() transformation on the Wikileaks vout data to get it in the format with the **hash** as the key. Then using the **textFile**() method to load the **vin** file and use the **map**() transformation on it to format the **txid** as the key. Then I call the **join**() transformation on the WikiLeaks vout data with the vin data as an argument.

However the vin file doesn't contain the address of the wallet the BTC was sent from.

3. We need to join this again to the **vout** file to get the address that BTC was sent to before this transaction. We use the **tx_hash** and **vout** fields from the previously joined data, and the **hash** and **n** fields from the **vout** file as the join key.

This is done by calling the **map**() transformation on the joined data to format it so the **tx_hash** and the **vout** fields are the key (in a tuple). Then I load the **vout** file again using the **textFile**() method and call the map transformation to format the **hash** and the **n** fields as the key (in a tuple). Then I call the join transformation on the previously joined data with the vout data as an argument.

4. We then need to get only the address of the sender and the amount of bitcoins sent from the joined data. This is all the donations made to the WikiLeaks.

   This is done by calling the the **map**() transformation on the joined data to format the data so we only have the public key (address of the sender) as they key and the amount of bitcoins sent as the value.

5. As this is all the donations, we need to group donations made by the same address to find all the donors.

   This is done by calling the **reduceByKey**() transformation on the donations so all the donations from the same address will be aggregated.

6. We then need to order this by the amount of bitcoins sent in descending order and take the top 10 rows.

   This is done by calling the **takeOrdered**() action to give us the top 10 donors to the wikiLeaks bitcoin address.

Results:

The output of part B is:

```
('{17B6mtZr14VnCKaHkvzqpkuxMYKTvezDcp}', 46515.1894803)
('{19TCgtx62HQmaaGy8WNhLvoLXLr7LvaDYn}', 5770.0)
('{14dQGpcUhejZ6QhAQ9UGVh7an78xoDnfap}', 1931.482)
('{1LNWw6yCxkUmkhArb2Nf2MPw6vG7u5WG7q}', 1894.3741862400002)
('{1L8MdMLrgkCQJ1htiGRAcP11eJs662pYSS}', 806.13402728)
('{1ECHwzKtRebkymjSnRKLqhQPkHCdDn6NeK}', 648.5199788)
('{18pcznb96bbVE1mR7Di3hK7oWKsA1fDqhJ}', 637.04365574)
('{19eXS2pE5f1yBggdwhPjauqCjS8YQCmnXa}', 576.835)
('{1B9q5KG69tzjhqq3WSz3H7PAxDVTAwNdbV}', 556.7)
('{1AUGSxE5e8yPPLGd7BM2aUxfzbokT6ZYSq}', 500.0)
```

The only address for which I was able to find information about who it belongs to is the one highlighted in bold above. It is the address of what used to be a bitcoin exchanged called **Mt. Gox**.

The price of bitcoin at the time of the donations was roughly £10. The price of bitcoin now is roughly £3200. Using these prices we can derive the table below

Address	BTC	Value then £	Value now £
17B6mtZr14VnCKaHkvzqpkuxMYKTvezDcp	46515.1894803	£465,151.89	£148,848,606.34
19TCgtx62HQmaaGy8WNhLvoLXLr7LvaDYn	5770.0	£57,700.00	£18,464,000.00
14dQGpcUhejZ6QhAQ9UGVh7an78xoDnfap	1931.482	£19,314.82	£6,180,742.40
1LNWw6yCxkUmkhArb2Nf2MPw6vG7u5WG7q	1894.3741862	£18,943.74	£6,061,997.40
1L8MdMLrgkCQJ1htiGRAcP11eJs662pYSS	806.13402728	£8,061.34	£2,579,628.89
1ECHwzKtRebkymjSnRKLqhQPkHCdDn6NeK	648.5199788	£6,485.20	£2,075,263.93
18pcznb96bbVE1mR7Di3hK7oWKsA1fDqhJ	637.04365574	£6,370.44	£2,038,539.70
19eXS2pE5f1yBggdwhPjauqCjS8YQCmnXa	576.835	£5,768.35	£1,845,872.00
1B9q5KG69tzjhqq3WSz3H7PAxDVTAwNdbV	556.7	£5,567.00	£1,781,440.00
1AUGSxE5e8yPPLGd7BM2aUxfzbokT6ZYSq	500.0	£5,000.00	£1,600,000.00

Part C: Data Exploration

Ransomware often gets victims to pay via bitcoin. Find wallet IDs involved in such attacks and investigate how much money has been extorted. What happens to the coins afterwards? [25 Marks]

Ransomware is a type of malicious software from cryptovirology that threatens to publish the victim's data or perpetually block access to it unless a ransom is paid.

CryptoLocker was a ransomware attack that occurred between Sep 2013 and May 2014. The CryptoLocker bitcoin address is `{1AEoiHY23fbBn8QiJ5y6oAjrhRY1Fb85uc}.`

Implementation in pyspark:

1. We need to filter the **vout** file to only get rows containing the Cryptolocker bitcoin address.

   In pysaprk I call the **textFile**() method to load the **vout** file and then use the **filter**() transformation with a function I defined to remove all rows which do not contain the CryptoLocker bitcoin address.
2. To get the number of victims who paid the ransom and the total amount of BTC extorted we need to count the number of rows in the vout file after we've filtered it to only contain the CryptoLocker rows.

   The number of transactions is found by calling the **count()** action on the vout data only containing the CryptoLocker rows. The total BTC extorted is found by calling the **map()** transformation on the vout CryptoLocker rows to get them in the format with the key as any constant value, I use the string "total btc" and the value as the amount of BTC sent. Then we call the **reduceByKey**() transformation on the previous RDD to sum the amount of BTC sent for all the transactions.
3. Then we need to join (inner join) the CryptoLocker vout to the **vin** file. This join will use the **hash** and **n** fields from the CryptoLocker vout RDD and the **tx_hash** and the **vout** fields from the **vin** file as join key. This gives us all the transactions that were sent from the CryptoLocker address.

   This is done by first calling the **map**() transformation on the CryptoLocker vout data to get it in the format with the **hash** and **n** as the key. Then I use the **textFile**() method to load the vin data set and use the **map**() transformation on it to format the

tx_hash and **vout** as the key. Then I call the **join**() transformation on the CryptoLocker vout data with the **vin** data as an argument.

However the vin file doesn't contain the address of the wallet the BTC was sent to.

4. We need to join this again to the vout file to get the address that BTC was sent to. We use the **txid** field from the previously joined data, and the **hash** field from the **vout** file as the join key.

   This is done by calling the **map**() transformation on the joined data to format it so the txid field is the key. Then I load the vout file again using the **textFile**() method and call the map transformation to format the hash field as the key. Then I call the join transformation on the previously joined data with the vout data as an argument.

5. We then need to get the address of where the BTC was sent to and the amount of bitcoins sent from the joined data. This is all the transactions of BTC sent from the CryptoLocker wallet.

   This is done by calling the the **map**() transformation on the joined data to format the data so we only have the public key (address of where the BTC was sent) as they key and the amount of bitcoins sent as the value.

6. As this is all the transactions, we need to group transactions sent to the same address to find all the distinct addresses BTC was sent to.

   This is done by calling the **reduceByKey**() transformation on the previous RDD so all the BTC sent to the same address will be aggregated.

7. We then need count the number of rows to get the number of addresses they sent the extorted BTC to.

   This is done by calling the **count()** action to get the number of addresses the extorted BTC was sent to.

Results:

The CryptoLocker bitcoin address received a total of **5338.98289731 BTC** (roughly £53,400 then, roughly £17,088,000 now) from **115** different people (different bitcoin addresses). On average each person paid **46.4 BTC** (roughly £464 then, roughly £148,480 now).

The extorted money in the CryptoLocker bitcoin address was then sent to 454 different bitcoin addresses from which it was either withdrawn, sent to other addresses or spent.