BlockChain Evolution

Viktor Bakayov
Vrije Universiteit
Amsterdam, The Netherlands
v.bakayov@student.vu.nl

Alexandru Custură
Vrije Universiteit
Amsterdam, The Netherlands
a.custura@student.vu.nl

1 INTRODUCTION

The Bitcoin, currently the world's most popular virtual currency, released in 2009, relies on the revolutionary technology of the BlockChain[1], a distributed, public ledger, which records transactions between different parties in a secure, efficient, verifiable and permanent manner.

Unlike traceable paper cash or digital currency, cryptocurrency keeps the identity of its owner hidden by means of periodically generatic different public keys for the owner's cryptowallet. However, if the digital cryptocurrency wallet is lost or stolen, it is hard or even impossible to recover or replace. Furthermore, cryptocurrencies' security and trust are derived from mathematical properties based on established and trusted cryptographic primitives, unlike physical or chemical properties of paper money.

The Bitcoin, as stated earlier, is decentralized, distributed and voluntary. Unlike traditional money, cryptocurrencies are easier to secure and transport anywhere in the world, triggering a growing number of entrepreneurs to accept or base new business concepts on cryptocurrencies.

Bitcoin has no central authority for issuing and verification. The Bitcoin Foundation[1] (founded in 2012) is the recognized non-profit organization coordinating the Bitcoin community towards standardizing the use of Bitcoin and promoting its worldwide use.

As opposed to other electronic currencies, cryptocurrencies are immune to sovereign censorship, shutdown, confiscation, and even inflation or - in light of more or less recent economic events - bank defaults.

The Bitcoin itself is a cryptoledger protocol based on the BlockChain, a growing general public ledger of transactions which are cryptographically signed.

Given the public nature of the BlockChain, anyone may access the complete history of transactions since the Bitcoin's inception eight years ago. Having access to all the transaction history, which is otherwise anonymized by making use of a tokenization algorithm which generates different public keys for every user, we can , at any moment in time, analyze the data and extract vital information with regards to transactional patterns or trends.

The BlockChain holds the history of all past transactions organized into blocks. Each block contains, among other information, a record of some or all recent transactions, and a reference to the block that came immediately before it. The process of "mining" is the process of computing the find an answer to very difficult and computationally expensive mathematical problem which is unique to each block. Once this answer is found a new block is created and added to the BlockChain. The validity of the newly created block now can be effortlessly verified by other miners and the transactions inside will be approved. Miners' incentive are the award of newly-minted bitcoins or transaction fees for successfully finding blocks.The collective computing power of the *miners* provides security and trust to the Bitcoin.

A transaction consists of one or multiple input addresses (a reference to an output from a previous transaction), and one or two outputs (one/multiple outputs for the recipients of the transaction, and one for any eventual change the sender receives if the total amount of the input transactions exceeds the desired amount to transfer). This is known as a 'change' address and is due to the fact that each output from one transaction can only ever be referenced once by an input of a subsequent transaction, the entire combined input value needs to be sent in an output if you don't want to lose it.

All transactions in the ledger are public. However, they are not tied to anyone's real identity by default, although anonimity and traceability are *user-defined*, as counter parties can be as anonymous as they choose to be.

Transferring *bitcoins* (units of account), does not entail physically moving an object from a source to its destination, but rather consists of adding a new, publicly accepted transaction entry to the BlockChain ledger. Once added, transactions may never be removed or modified.

For the scope of this paper, we have analyzed the temporal graph produced by the BlockChain in order to provide information visualization according to several metrics. Visualizing information may come in aid of identifying trends and patterns in the course of such analysis and when exploring the Bitcoin blocks and transactions. In section 2 we are to explore what others did in the field of working with the bitcoin blockchain, in section 3 we are to define our research question, in section 4 will describe what technologies we used to accomplish our task and what is the project setup, in section 5 we are to give more technical insight followed by our results. We will end with suggestions for future work and conclusion.

2 RELATED WORK

Since the Bitcoin ledger is public and the protocol has been used for approximately eight years at the date of the present paper, it is not unexpected that a plenitude of research has been carried out on the topic of cryptocurrencies, or the Bitcoin itself specifically.

McGinn et al.[2] were motivated by the numerous bottom-up approaches of derriving useful information from the BlockChain (such as analysis of individual addresses), and limited number of top-down approaches at the time, and have sought to generate a system-wide visualization in order to aid explaining the Bitcoin to the general public and aid in explorative analyses patterns and behaviours in transaction data.

In their approach to visualizing transactions, McGinn et al. have made the following decisions:

[1]https://bitcoinfoundation.org/

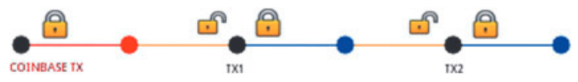


Figure 1: McGinn et al.'s visualization of a chain of spends, indicating a coinbase transaction coloured in red, and blue outputs from one transaction becoming orange outputs in the next.

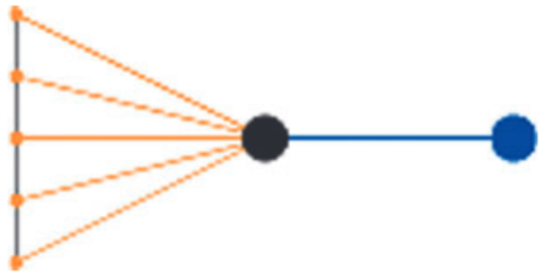


Figure 2: McGinn et al.'s visualization of a transaction consisting of five equal input transactions (denoted by the orange colour and equal sizes of the inputs) originating from one source address (denoted by the gray line connecting the inputs) and one blue-coloured output.

- *Transactions* have been marked in a dark gray colour, with all transactions having a fixed, equal size, representing the miner's reward.
- *Inputs* of transactions are coloured orange, and their size is relative to the amount of the input transaction.
- *Outputs* are coloured blue, and their size is also relative to the amount.
- *Addresses* are identified by gray lines connecting inputs and outputs of transactions.

On a larger scale, McGinn et al.'s visualization takes a globular shape. This visualization already provides insight into transaction patterns in the BlockChain. As can be seen in 3, we can observe a payout system, and what is believed to be a coin-tumbling service.

Applying techniques of information visualization on the Bitcoin transaction graph may also come in aid to forensics analysis. McGinn et al.'s approach eases the identification of possible money-laundering operations, where a very large amount of bitcoins is transfered to one account which spreads and shuffles the money around. The need for visualizing the data comprised in the Bitcoin transaction chain and the benefits which incur are, therefore, undoubted.

The Yu et al [3] paper provided insight how to proceed with the project as the authors explore the network structure, devise their own community detection algorithm from scratch, and employ different techniques for user grouping, whereas the Prat-Pérez at el [4] paper provides knowledge of what algorithms can be used for community-like analysis. In the above paper they use algorithms such as Clustering Coefficient,TPR, Bridge Ratio (d) Diameter and Conductance. The Ron at el [5] provides for quantitative analysis

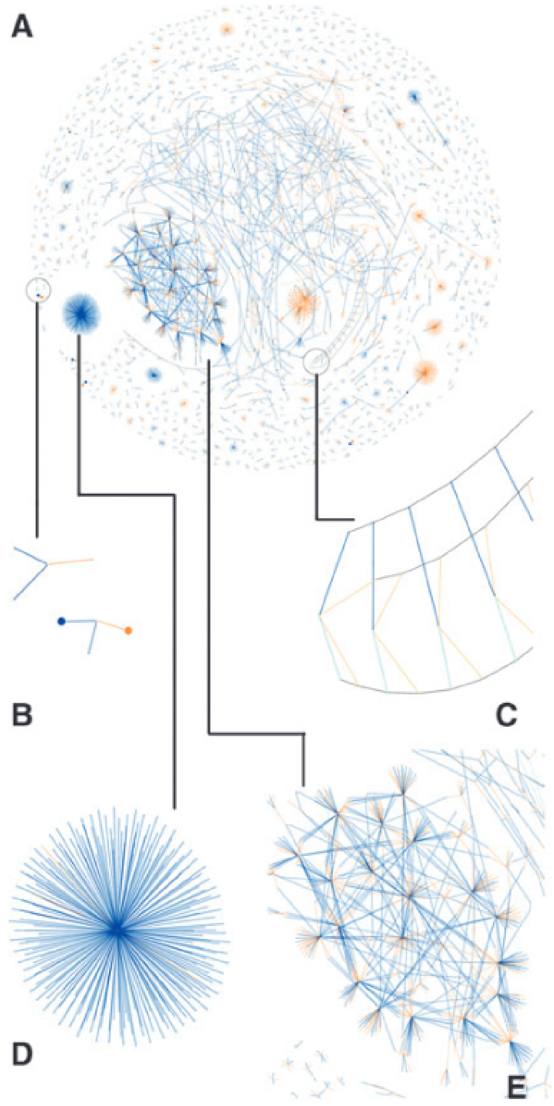


Figure 3: McGinn et al.'s high-resolution visualization of a transaction block (A), small and high value transactions (B), connected Bitcoin addresses (C), a payout system (D), and a disconnected component believed to be a service which moves money rapidly between addresses in order to obfuscate sources and destinations of transactions (E).

of the full bitcoin transaction graph. Their analysis is very comprehensive as they provide numerous insight about the transactions, BTC flows, addresses and, bitcoin exchanges etc.

3 RESEARCH QUESTIONS

During the course of our experiments performed on the BlochChain data acquired, we seek to learn about the evolution on the BlockChain over time, as the Bitcoin transactions form a temporal graph.

The end-goal of our project is the visualization of all the data collected in the form of a Web page in order to provide an insightful manner of viewing the evolution of the transaction graph.

Our primary research question is, therefore:

- Can we provide better insight into the evolution of the Bitcoin BlockChain by studying the graph structure through information visualization techniques We will provide possible insights and measurements, among of which clustering, community detection and centrality and study the blocks and the transactions inside.

Furthermore, we are seek to parse the data in a reliable and timely manner. The collection of blocks we have analyzed adds up to 130GB of data which needs to be parsed and processed. Therefore, another objective of our project is finding the appropriate technology and tools which will allow us to easily parse, filter and process the BlockChain transaction data.

Therefore, our second research question is:

- Which tools and technologies can we appropriately make use of in order to process large datasets such as the Bitcoin BlockChain in an efficient and flexible manner?

4 PROJECT SETUP

4.1 Data collection

In the scope of this paper, we have received access to approximately 130GB-worth of compressed data encompassing part of the Bitcoin transaction history. The data holds Bitcoin transactions spanning from 2009 until 2016. The complete set of files has been downloaded to the SURFsara[2] cluster, where our experiments were carried.

4.2 Technologies used

One of the objectives of our experiments was to find the appropriate technology and tools in order to parse, filter and process our large dataset efficiently and reliably.

For the purpose of our project, we have identified and made use of several tools for different tasks during the experiments.

We had begun with Java and Apache Spark[3], the latter being chosen over Hadoop MapReduce[4] for the task of analyzing multiple large datasets, not only due to its novelty, but also the emphasis placed on speed and flexibility.

Moreover, since the BlockChain transaction data described events occurring over time, it makes sense to describe it as a temporal graph. Naturally, we have opted for Spark's GraphX[5] API in order to map our data as a graph. GraphX also provides several graph algorithms, such as PageRank, identifying connected components, strongly connected components or triangle count, which we have used as metrics for getting insight into the data at hand. We shall describe the metrics later on in the paper.

Since GraphX Java API is still in Alpha Version, we choose to programme in Scala[6] and we had to add it as a dependency to our project. Java is a verbose programming language and one of the many benefits of Scala is to aid in boilerplate reduction and it further simplifies our code.

As of March 2016, Spark team introduces GraphFrames [7]. It support the full set of algorithms available in GraphX. The key difference is that GraphFrames are based upon Spark DataFrames, rather than RDDs, and therefore benefit from it's scalability and high performance.Also, GraphFrames allow users to phrase queries in the familiar, powerful APIs of Spark SQL and DataFrames.We interleave both graph libraries for best results.

Furthermore, in order to run queries on the data run inside the Spark program, we have used the Spark SQL[8] module, which allows querying using semantics of the well-known Structured Query Language (SQL).

Moreover, we included the *hadoopcryptoledger*[9] library into our project. This is an open-source library which aids the task of analyzing/parsing CryptoLedgers such as the Bitcoin BlockChain, and integrates very well with our selected technologies.

Finally, for reaching our end-goal of providing a visual manner of obtaining insight into the BlockChain transaction graph, we have plotted out results in a Web page. For this purpose, we have used common technologies such as HTML and JavaScript. Specifically for the task of plotting the charts, we have used a number of amCharts[10] JavaScript libraries. The library is extremely interactive, all charts can be zoomed-in or panned, annotated, downloaded/saved and exported in various formats or printed. Individual data categories, can be turned on/off for easy exploration in some graph types.

5 EXPERIMENTS

5.1 Implementation

In our implementation cycle we set up two development environments. A single machine local set up which we used for testing and debugging purposes and a cluster environment where we ran the jobs on the fully intended data range. This division would save us the whole hassle to run jobs on the cluster during rapid development and testing, that is packaging the jar, logging to the logging node, interacting with the HDFS and submitting jobs. In addition, starting a cluster job can take up to 2 minutes, if the job is not put in a queue, whereas a local job would start almost instantly. Eclipse IDE integrated well with what we used, the pre-build Apache Spark release for Hadoop 2.7 [11] and we had the convenience to run our jobs directly from Eclipse. For dependency management and jar building we used Maven [12] with the 'maven-assembly-plugin' for dependency compilation and the 'scala-maven-plugin' for scala compilation. In addition we had to integrate Eclipse with Scala by installing the JDT Weaving plug-in [13] and Eclipse with Maven by installing M2Eclipse Eclipse plug-in [14]. It is important to match all dependencies with completable library/Scala versions or there will be compilation or run-time errors.

[2]https://userinfo.surfsara.nl/systems/lisa
[3]https://spark.apache.org/
[4]https://hadoop.apache.org/
[5]https://spark.apache.org/graphx/
[6]https://www.scala-lang.org/

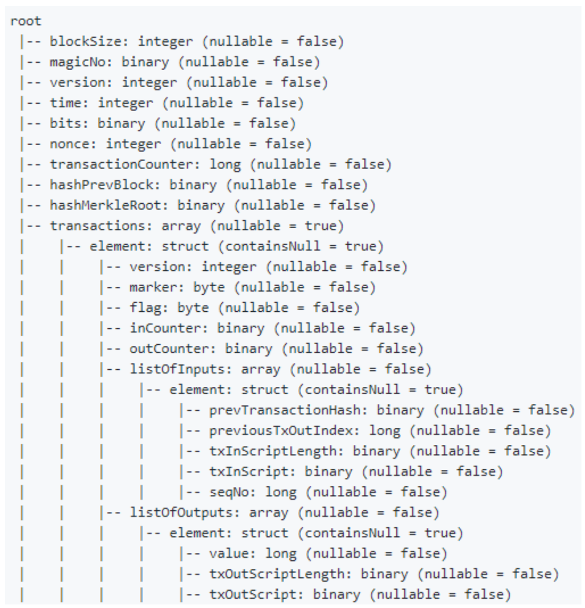
[7]https://graphframes.github.io/
[8]https://spark.apache.org/sql/
[9]https://github.com/ZuInnoTe/hadoopcryptoledger
[10]https://www.amcharts.com/
[11]https://spark.apache.org/downloads.html
[12]https://maven.apache.org/
[13]https://wiki.eclipse.org/JDT/_weaving/_features/
[14]http://www.eclipse.org/m2e/

The Bitcoin blockchain network is downloaded and put into HDFS. The file is partitioned into 789 blocks which are stored into different Data Nodes. The file is replicated several times and again replicated blocks are store in different Data Nodes. In order to load the data and read transactions and blocks from files in HDFS we would need to parse the data and set the hadoop file format, input split, etc. Luckily, the 'hadoopcryptoledger' library already provides a Bitcoin block input format, which deserializes blocks containing transactions. The Bitcoin blocks will be put into RDDs. Resilient Distributed Dataset (RDD) is the main abstraction Spark provides, which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel.

Next, we would need to understand more in-dept the structure of the blockchain. The format for blocks as implemented in the parsing lib is shown in fig. 4. Among the important properties of a block is the 'time'- when the block was created, 'hashPrevBlock' - the link to the previous block, 'transaction' - an array containing all the transactions.

Proof of work in Bitcoin's mining takes an input the 'hashMarkleRoot', 'timestamp', 'hashPrevBlock','nounce' etc. If the output results in hash is smaller than the target hash you win the block and the consensus is reached. The 'nounce' is a completely random number between 0 and 2 to the power of 31 which is brute forced in order to luckily find a hash smaller than the target hash, which is calculated based on the difficulty.

The 'transactions array' contains the transactions per block. Each transaction has a list of input addresses and a list of output addresses.



```
root
 |-- blockSize: integer (nullable = false)
 |-- magicNo: binary (nullable = false)
 |-- version: integer (nullable = false)
 |-- time: integer (nullable = false)
 |-- bits: binary (nullable = false)
 |-- nonce: integer (nullable = false)
 |-- transactionCounter: long (nullable = false)
 |-- hashPrevBlock: binary (nullable = false)
 |-- hashMerkleRoot: binary (nullable = false)
 |-- transactions: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- version: integer (nullable = false)
 |    |    |-- marker: byte (nullable = false)
 |    |    |-- flag: byte (nullable = false)
 |    |    |-- inCounter: binary (nullable = false)
 |    |    |-- outCounter: binary (nullable = false)
 |    |    |-- listOfInputs: array (nullable = false)
 |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |-- prevTransactionHash: binary (nullable = false)
 |    |    |    |    |-- previousTxOutIndex: long (nullable = false)
 |    |    |    |    |-- txInScriptLength: binary (nullable = false)
 |    |    |    |    |-- txInScript: binary (nullable = false)
 |    |    |    |    |-- seqNo: long (nullable = false)
 |    |    |-- listOfOutputs: array (nullable = false)
 |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |-- value: long (nullable = false)
 |    |    |    |    |-- txOutScriptLength: binary (nullable = false)
 |    |    |    |    |-- txOutScript: binary (nullable = false)
```

Figure 4: Block Format

5.2 Descriptive Statistics

Few descriptive statistics for some global property of the network over time can be derived solely from the blockchain. The **transactions per day**, **average block size** and **average transactions per day** were calculated. The .dat files were parsed and loaded into a *spark.sql.DataSet.* We though it would be better to shift the complexity, cumbersomeness etc. of writing map-reduce tasks or Spark tasks into writing SQL Queries (on top of Spark) for this kind of analysis. After building the graph the Top 5 addresses with most inputs/ outputs were found. A discussion of the results is to be found in the results section.

5.3 Building the graph

To explore further the chain we would like to model the network in such a way that we can study the relationships/patterns/trends between the different entities and for such purpose we would need to transform our data into a graph. **Our main goal is to study the structure of the resulting graph.** We have several possibilities to model the graph shown in Table 1. We choose to proceed with Address/Address graph. The graphs are all fundamentally different, they have different nodes, different edges and they encompass different relationships. They also have very different properties, and as a result no all of them can be used for the same type of analysis. We have to be very careful of how we are interpreting graphs, and that we understand the repercussions of the particular graph we choose for the particular type of analysis. Lets briefly discuss the two most obvious for our research question.

Transaction/ Transaction graph – Transactions are the vertices with a directed weighted edge from each input transaction to an output transaction with the value of the transaction being the weight.

Address/Address graph - Addresses are the vertices; the edges are the transactions that encompass moving of money between those nodes, or more specifically a directed edge from one source Bitcoin address to a destination Bitcoin address. Each user can have unbounded number of addresses and each address most commonly belongs to a single user. This is the case because every address is associated with a unique private key. When a user send money back to itself there will be a self-edge. There will be more than one edge between two users when multiple transactions between them occur. There is no 1:1 or 1:n mapping from input to output, but n:m (all inputs are assigned to all outputs) as a single transaction can take money from multiple addresses and move it to multiple output address. **Our current graph model is in this state**. A major improvement will be to migrate to User graph where a node will denote public addresses of anonymous individuals or "entities" and the directed edge represents a particular transaction from a source address to a target address. Finding the entities is not a straightforward method and can be a project on its own. For the sake of clarity we are to briefly discuss it in the next paragraph.

For a user graph (which is an improved transaction/transaction graph) a few assumptions must be introduced. Assume that a transaction is constructed by a single user and thus that all the inputs are controlled by the same entity. Pairs of vertices can be connected with undirected edges, where each edge joins a pair of public keys

	Directed	Acyclic	Bipartite
Address/Address	Y	N	N
Address/Transaction	Y	N	Y
Input/Output	Y	Y	Y
Output/Output	Y	Y	N
Transaction/Transaction	Y	Y	N

Table 1: Graph Types

that are both inputs to the same transaction and are thus controlled by the same user. Another heuristic that can be used to combine users is looking at so called 'change' addresses [6]. The change addresses usually stay in control of the user. It is common practice to generate a new and previously unused address for this change. If a transaction, thus, has multiple outputs and only one of them is unused then can be assumed that this is the change address which is under the control of the sender [7]. The user or entity will consist of a collection of public key addresses that were used during separate transactions. Furthermore, the user's identity can be deanonymized given publicly available information such as scraped bitcoin forum users, posted addresses for donation purposes, or publicly known addresses.

In order to create a Graph object the vertices and edges for our model must be found. Going through each block, and through each transaction we would create a tuple for each transaction input with the destination address, input transaction hash, current transaction hash, current transaction output index. We would create the vertices being a tuple (vertex Id, Bitcoin Address). To create the edges we need to determine which input vertex id, refers to which output vertex id. This is a self join, where ((current Transaction Hash,current Output Index), identifier) is joined with ((input Transaction Hash,current input Index), identifier). Luckily, the hadoop crypto ledger library provides for such an example. We can now create our GraphX object and start running our graph algorithm which are described in the section below. GraphFrames has an integration with GraphX via conversions between the two representations and therefore we can easily create GraphFrames graph.

5.4 General Discussion

The metrics we use for evaluating our graph structure are mostly iterative algorithms. With the increasing size of the graph we should pay more attention to performance optimization. According the the GraphX documentation uncaching may also be necessary for best performance as intermediate results from previous iterations will fill up the cache. Though they will eventually be evicted, the garbage collection will be slowed down by the unnecessary data stored in memory. An alternative will be to use the Pregel API which will correctly unpersist intermediate results.

An improvement will be to account for the data locality in the cluster. Spark applies a 'Vertex Cut' technique in GraphX to distribute the data throughout the cluster. One have to be very conscious where the edges and vertices are as operations can vary dramatically in terms of parallelism and data locality depending where you put your data.

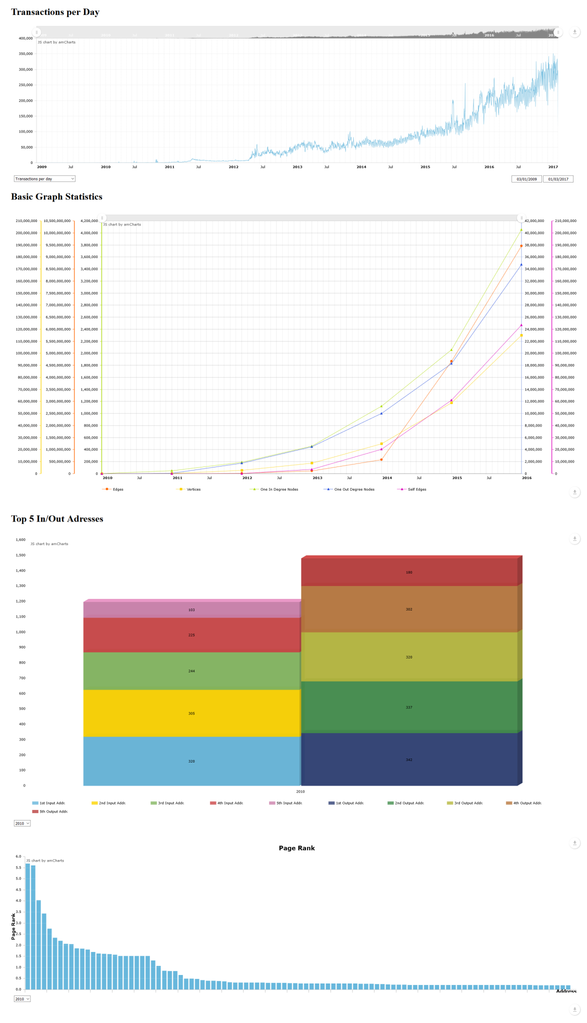


Figure 5: Visualizations of our applied metrics on the Bitcoin BlockChain. From top to bottom: the number of transactions per day, core graph statistics, the top 5 in/out addresses, and PageRank.

6 RESULTS

Our projected aimed to analyze the evolution of the BlockChain over time. For this purpose, we split the graph in multiple time based snapshots. The time range of each consecutive snapshot is incremented by 1 year. For example, the first snapshot will range from 2009 to 2010 year and the last snapshot will range from 2009 to 2016, seven snapshots in total. In addition, for the graph algorithms we made a snapshot on 1st of February 2017 when most transactions occurred for a single day. Below we are to briefly describe the metrics we used for evaluation and the results we obtained.

Unfortunately, we could not provide a complete overview for all of the following algorithms, as most of them are iterative and their performance degrade with the increase of problem size. Also, parts of them are non-parallelizable which gives additional overhead. For

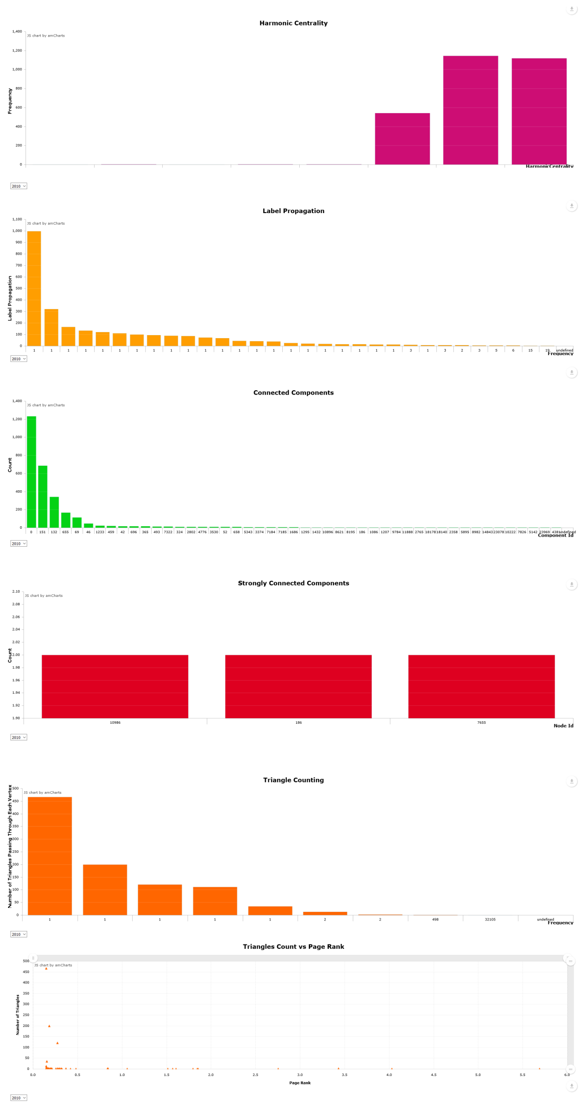


Figure 6: Visualizations of our applied metrics on the Bitcoin BlockChain. From top to bottom: the harmonic centrality, label propagation, the number of connected components, the number of strongly connected components, triangle count and triangle count versus PageRank.

example an algorithm would run for 4 days using 1/4th of cluster resources and still would not be able to produce results.

The front-end 'amCharts' is able to process and visualize a lot of data points. Some graphs will use logarithmic scale value axis as it reduces the wide range of values to more manageable size. To improve further the user experience custom java scripts were created to further group the data as most of the graph algorithms produced data points with same values. Also it is important how the data is ordered, and which data is used as the value axis. The scripts also handled unexpected behavior. For example, ordering big amounts of data will distribute it among the reducers as part of the ordering process. If one is to repartition to a single reducer the collected data will be ordered per reduced and will not be globally ordered as one might expect. Solution was either to not repartition and use the hdfs 'getmerge' command or handle ordering by additional script.

6.1 Descriptive Statistics

Our descriptive statistics are time based serial data divided into one calender day interval. The time span range is from the first transaction on 3 of January 2009 to the last available data point 1st of March 2017. When doing time series analysis we can have insights in the evolution of the studied component over time. The transaction per day graph is shown 1st at Fig. 5. We see that until April 2012 the bitcoin network was not particularly active having around 8 thousand transactions per day. From there on the transactions count started to steadily and gradually increase having 200 thousands transactions in the end of 2015. We can see that the most transactions happened on 1st of February 2017 - 351 376. The other two time series graphs have identical trends of increasing. As more transactions were generated per day the average block size per day will expectedly increase as well. The block size is determined by miners and intuitively one could say that big blocks will allow more transactions to be carried, and therefore more fees would be generated. However, a miner would like to limit the supply of transaction space so the fees for putting the user's transaction in the next block will increase. The optimal block size for miners is "small enough to drive congestion" [8]. We also provide a metric for average transactions per block per day. We see there were peaks on 17th of September 2015 (1789), 29 of February 2016 (1995) with most being at 15h of December 2016 (2212). Generally, we see that the network handles the increasing demand/traffic well.

The top 5 In/Out address graph (3rd graph in Fig. 5) compares the accumulative total of transaction used as input and output. In 2009-2010 time range stands out an address which was used mored times as output than the commutative total of the 5 input address. Checking this address in a popular bitcoin blockchain explorer [15] we see that this account received enormous amount of BTC 77k with 113 transactions. In 2012 the first big mining pool address was created as evident from the chart. For a short period in 2012 this address generated 10k transactions and received total of 315k BTC [16]. The funds were distributed among of its participants. Gradually mining pool address prevail (as used as input for a transaction), when comparing to the output addresses. For example the 2014 snapshot is saturated with address starting with '06F1' and it is practice for mining pools to generate similar addresses.

6.2 Core Graph Statistics

To describe the resulting graph we use found the number of **nodes**, **edges**, **one in degree nodes** - addresses used only once as input, **one out degree nodes** - addresses used only one as output, **self edges** - an address used both as source and destination in a transaction.

We plot the core graph statistics on a multiple value axis for easy comparison and to hide the different value range. The graph

[15]https://blockchain.info/

[16]https://blockchain.info/address/1PJnjo4n2Rt5jWTUrCRr4inK2XmFPXqFC7

is 2th in Fig. 5. We again observe increasing trend, but from 2014 it becomes more steep. When comparing between axis we see the edges (that is the transactions) followed by self-edges, have the steepest slope. The graph as of January 1st has 114 million nodes as of 4 million were used once as an input and 34 were used once as an output, 9,5 billion edges as of 123 million are self-edges.

6.3 PageRank

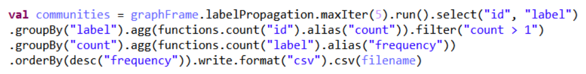
PageRank[9] is a famous algorithm used by Google to rank search results in its search engine. In our use case Page rank will work by counting the number and quality of the edges (transactions) to the vertices (addresses) to give a rough estimate how important the addresses are. We ran Page Rank for 10 iterations, and output the top 100 addresses with biggest Page Rank.

It can be noticed that 5 to 10 addresses are very active during each year and then the distribution becomes more uniform. Intuitively, the page rank increases each consecutive year. There is some correlation with the results of the previous graph, but yet there are high ranking address which did not pass the cut-off of 5. Similar pattern is observer for the 2017 snapshot.

6.4 Label Propagation

This algorithm is used for detecting communities in a network. Each node in the network is initially assigned to its own community. At every 'super step', nodes send their community affiliation to all neighbors and update their state to the mode community affiliation of incoming messages [10]. The algorithm is displayed in Figure 7. We run it with 5 iterations, and each node is assigned label Id. Then we would like to aggregate by counting the nodes having the same label Id. However the output is still too big as there are a lot of communities with same same count. Next, we group by the count and aggregate by counting the labels. Finally, the results are ordered. The result is of the form label count, frequency.

All snapshots follow similar pattern. A big amount of nodes have small label propagation value, but will gradually, uniformly increase in value, while the frequency will decrease. 5 to 10 nodes will have drastically more propagation measurement for 2013 and 2014, while 2010 and 2011 are more well distributed. The 2017 snapshot is similar. This suggests that most communities have few members, indicating one-off transfers between one user and another. The data is consistent with the findings of Yu et al.[3]



```
val communities = graphFrame.labelPropagation.maxIter(5).run().select("id", "label")
.groupBy("label").agg(functions.count("id").alias("count")).filter("count > 1")
.groupBy("count").agg(functions.count("label").alias("frequency"))
.orderBy(desc("frequency")).write.format("csv").csv(filename)
```

Figure 7: Lebel Propagation Algorithm and Following Result Groupings

6.5 Harmonic Centrality

Centrality is similar to Page Rank as it tries to account for the importance of the nodes. A whole plethora of centrality measures have been proposed and we found a public library [17] for GraphX which measures the harmonic centrality of a node. This is the sum of the reciprocal of the shortest path distances from all other nodes to x. We would again group on the centrality value and aggregate by counting the vertex IDs and furthermore group on value. However, we had to round our harmonic centrality values before the final grouping as the value range is too much for the front-end graph to handle efficiently each individual data point. The 2011 snapshot is rounded down to each 5, the 2012 snapshot to each 1,000 and the 2017 snapshot to each 50. This way we loose granularity, but we can visualize all the data efficiently and without specifying thresholds.

This graph is the only one plotted with switched values. The Harmonic centrality is plotted on the x-axis and the frequency on the y-axis as this measurement has big frequency fluctuations. The y-axis is on logarithmic scale. The data is ordered on harmonic centrality. There is subtle pattern that with the decrease of centrality the frequency increases, very obvious in the 2012 snapshot, less obvious in the others. We see that quite few nodes have big centrality, but prevailing are node with small centrality values. For example, nodes rounded down to 0 centrality (in reality ranging form 0 to 1000) in the 2012 graph are 920k, accounting for the biggest frequency value.

6.6 Triangle Components

A vertex is a part of a triangle when it has two adjacent vertices with an edge between them [10]. The algorithm count the number of triangles passing through each vertex, providing a measurement of clustering. The output is again grouped by the count, the output being the number of triangles passing through each vertex, frequency of occurrence (addresses). The result is ordered by decreasing number of triangles.

Again, the results suggest clustering for few addresses, having a lot of triangles passing through each of them. The first 4 addresses for the 2011 graph ,which are quite an outliers, have 3.660, 3.034, 1.509, 1.423 triangles passing through them. The following node counts decrease more gradual as the frequency decreases. Finally there are 150k nodes which have no triangles passing though them. The 2012 graph shows even more prominent results, with 105 addresses having triangle count more than 25k. The rest of the distribution is decreasing more gradually, with sporadic high value frequency counts, for example 361 nodes with 4,976 triangle counts. The 2017 graph is following a similar pattern.

6.7 Connected Components

A connected component in graph theory is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph [11]. The connected components algorithm labels each connected component of the graph with the ID of its lowest-numbered vertex [12]. At the end of iteration each vertex is assigned a component ID. The result is grouped by components, counting the individual vertices and excluding vertex components by their own, ordered by count and in the form - (component Id, number of components). Finally, the component Ids with same counts were grouped and outputted as (value, frequency) pairs.

For the 2012 snapshot we observer there is one big connected components having 2,726k addresses connected together. This can be correlated with our previous results of the emergence of mining

[17]https://github.com/webgeist/spark-centrality

pools. With the frequent money transactions it is easy most of the addresses to be connected. However, this seems somewhat unplausible to the authors of this paper as there are only few more connected components. There is a bug both in GraphX and GraphFrames concerning Spark chain-indexes and producing similar incorrect results [18]. We used versions in which the bug is claimed to be fixed, and tried workarounds to this issue, but all we had was the same results. We believe the result is plausible, but it should no be trusted exclusively as both libraries are still considered production not ready.

The 2017 snapshot has more dispersed values. For example, 3793 clusters of 3 components connected was on the top for frequency. Again one big sub-graph is observed with 500k connected components.

Generally we see one common pattern that there are a lot of clusters with small count of connected components and as the cluster count (frequency) decrease the connected components (count) increases.

6.8 Strongly Connected Components

A graph is strongly connected if there is a path between all pairs of vertices. Similarly to above, the algorithm returns a graph with each vertex assigned to the SCC containing that vertex. This is a stronger form of 'connected components' as it required that every vertex is reachable from every other vertex as well. This algorithm is able to find entities interacting with each other and therefore being a community. We run the algorithm for 10 iterations, the result is grouped by component Id with aggregating function being the vertex frequency count and ordered in descending order. Similarly, the component Ids with same counts were grouped and outputted as (value, frequency) pairs. This algorithm is extremely slow and it is unfeasible to be run on big graphs.

In the 2010 snapshot there is 3 strongly connected components having 2 nodes. We don't have the data for 2011 for an unknown reason. The 2012 data shows one big cluster of 1,613,000 connected components and several smaller, but more frequent. As they are few, we are to mention them all in the format (connected components count, frequency) - (2,134),(3,26),(4,13),(5,3),(1,613,000,1),(8,1).

6.9 Page Rank vs Triangular Count

We would like to find if the Page Rank is related to the number of triangles. For such purposes we would join the two results on the vertex Id and filter appropriately to reduce the result enough for the front-end JavaScript visualization. The output is in the form (pagerank, triangle count).

The results show there is no correlation between Page Rank and Triangular count. As observer, there might be a node with hight Page Rank, but low triangle count and vise verse. Again most of the values are clustered in the lower left corner, with few outliers in both dimensions, which will certainly be of interest.

6.10 Local Clustering Coefficient

Clustering coefficient is the degree to which the nodes tend to cluster together. However, both GraphX and GraphFrames does not provide a clustering coefficient implementation. We found a public library [19] with an implementation for clustering coefficient for GraphX. It ran perfectly on our local machines, but it would fail when running on the cluster with an assertion error from within the library. We did not proceed further with this error and so we would not present results for this measurement.

The results of our metrics and their corresponding visualizations are publicly accessible online[20].

7 FUTURE WORK

The experiments conducted provide the foundation to obtaining information in a visual manner from the Bitcoin blockchain data using MapReduce for processing the large dataset in an efficient and timely manner. Our experiments have lead us to deduce the following observations which may be advisable for future work conducted on the topic of analyzing the blockchain:

- Instead of building the graph each time when job is started, save the snapshots to parquet files in HDFS and load the pre-build snapshots. The process itself is not that demanding when compared to the iterative algorithms, but it would save more time as the time range increases and therefore the 2016 savings will be significant.
- Migrate to User graph which will be more representative for the community analysis.
- More in-dept analysis on the money flow. Explore BTC accounts holding, study what part of the BTC are in circulation and which accounts are the biggest holders, provide functionality for a money flow of transaction which appears particularly bad.
- Partitioning the graph from 2009 onwards with 1 year increasing interval makes it hard for analyzes due to the size of the problem even in such a distributed environment. Migrating to User graph could solve this issue, but alternative is to partition the available time range by 1 year alone. However, this would loose the 'evolution' aspect of the study, but rather localize it for a single year.
- Analyzing the results in the front-end would actually benefit by including the actual bitcoin address when possible. Also some graphs can be connected by 'linking and brushing' techniques.

8 CONCLUSIONS

Our project aimed to provide insight into the evolution of the Bitcoin transaction graph, spanning from 2009 until 2016, by means of information visualization techniques. We have chosen several metrics which we applied to the BlockChain in order to extract useful information about it's network structure as well as we provided an overview statistics for the blocks and the transactions inside.

The results show that the graph size and complexity increases with time. From 2012 the network started to be heavily used and with the emergence of mining pools it become heavily clustered around 5 to 10 popular addresses. The rest of the network's clustering/centrality is well spread across the range of all the metrics, but

[18]https://github.com/graphframes/graphframes/issues/159

[19]https://github.com/SherlockYang/spark-cc

[20]https://vbakayov.github.io/

also we observer a lot of nodes which are non active and not well connected to the rest of the graph.

Working with a very large dataset, comprising 130 GB-worth of Bitcoin transaction-data, incentivised us into running our experiments in Spark jobs on the SURFsara cluster. This approach definitely eased and quickened our task. However, we ran into the bottleneck of running expensive algorithms onto large graph, which hindered our results analysis.

Spark's GraphX & GraphFrames APIs naturally helped modeling the transaction history as a graph, and also provided us the necessary tools to perform our metrics analysis on the BlockChain, with added help from the Spark SQL module in order to query the data inside our Spark programs.

Finally, the outputs of the Spark jobs were included in a static Web page, and plotted, with the hopes of providing a visual and facile way of obtaining insight and identifying patterns in the evolution of the Bitcoin cryptoledger.

REFERENCES

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[2] Dan McGinn, David Birch, David Akroyd, Miguel Molina-Solana, Yike Guo, and William J Knottenbelt. Visualizing dynamic bitcoin transaction patterns. *Big data*, 4(2):109–119, 2016.

[3] April Yu and Benedikt Bünz. Community detection and analysis in the bitcoin network cs 224w final report. 2015.

[4] David Dominguez-Sal Arnau Prat-Pérez. How community-like is the structure of synthetically generated graphs?

[5] Adi Shamir Dorit Ron. Quantitative analysis of the full bitcoin transaction graph.

[6] Marc Roeschlin Tobias Scherer Elli Androulaki, Ghassan O Karame and Srdjan Capkun. Evaluating user privacy in bitcoin, 2013.

[7] Benedikt Bunz April Yu. Community detection and analysis in the bitcoin network, 2015.

[8] Willy Woo. Determining the ideal block size for bitcoin. https://www.coindesk.com/charts-determining-ideal-block-size-bitcoin/, 2017.

[9] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[10] GraphFrames. Graphframes user guide. https://graphframes.github.io/user-guide.html.

[11] Wikipedia. Connected component (graph theory). https://en.wikipedia.org/wiki/Connected_component_(graph_theory).

[12] GraphX. Graphx programming guide user guide. https://spark.apache.org/docs/latest/graphx-programming-guide.html.