Efficiently Answer Clustering Queries on Memory Constrained Devices by Summarizing and Caching Bitcoin Blockchain

W. A. D. I. M. Perera

Efficiently Answer Clustering Queries on Memory Constrained Devices by Summarizing and Caching Bitcoin Blockchain

Isuranga Perera

Index : 14001179

Supervisor : Dr. C I Keppetiyagama

Co-Supervisor : Dr. P V K G Gunawardana

Submitted in partial fulfillment of the requirements of the

B.Sc. (Hons) in Computer Science Final Year Project (SCS 4124)

January 2019

Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate

.....
W.A.D.I.M. Perera
Date: 08/01/2019

This is to certify that this dissertation is based on the work of Mr. W.A.D.I.M. Perera under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Supervisor

.....
Dr. C.I. Keppetiyagama
Date: 08/01/2019

Co-Supervisor

.....
Dr P. V. K. G. Gunawardana
Date: 08/01/2019

Abstract

Bitcoin blockchain analysis helps law enforcement and financial institutions identify and stop bad actors who are using cryptocurrencies for illicit activity such as fraud, extortion, and money laundering. Blockchain address clustering is a major analysis technique in which all addresses appeared in blockchain is clustered so that each cluster contains addresses only from a single wallet. With a proper address tagging mechanism in place, clustering can be used to trace back transactions to its users.

Bitcoin blockchain which keeps growing rapidly can be seen as the greatest challenge faced by blockchain analytics tasks, specifically address clustering. The execution time of existing address linking models are inversely proportional to available memory and exponentially increases with the growth of blockchain. But none of the available major Bitcoin analytics platforms hasn't addressed this problem. In order to address this problem, we'll be providing a solution which involves summarization and caching of Bitcoin blockchain to improve Bitcoin address clustering speed on memory constrained devices.

Our address linking model outperformed all major Bitcoin analytics platforms including BlockSci, GraphSense and BitIodine in term of clustering speed and proved to perform efficiently on memory constrained devices as well.

Keywords *- Bitcoin, Blockchain, Address clustering, Tagging, Optimization, Summarization, Caching, Prefetching, Blockchain analysis, Linking*

Preface

This study tries to develop a new Bitcoin address linking model through summarization and caching of blockchain data. Since we're only focussing on reducing the execution time of the existing clustering approaches without affecting their clustering accuracy we are using existing address linking heuristics. Instead of building from the scratch we have used BlockSci parser and other APIs exposed by BlockSci platform to parse Bitcoin data. Our clustering model which uses summarization and caching of Bitcoin blockchain technique has outperformed all available major Bitcoin analytics platforms in terms of speed and has proven to be efficient in memory constrained devices as well.

Acknowledgements

I would like to express my sincere gratitude to my research supervisor, Dr C. I. Keppetiyagama, senior lecturer at University of Colombo School of Computing and my research co-supervisor Dr P. V. K. G. Gunawardana, lecturer at University of Colombo School of Computing for providing me continuous guidance and supervision throughout the research.

I would also like to extend my sincere gratitude to Dr Kasun de Zoysa, Dr T. Sritharan and Mr Charith Madusanka for providing me with feedback and showing the best possible paths.

Many thanks to my beloved mother and my dear father for always being my strength, showing me the correct direction, and making me who I am today. This thesis is dedicated to all my family members, to primary and secondary school teachers, to lecturers of University of Colombo School of Computing and to anyone who supported even by words to make my dreams come true.

Table of contents

List of figures

List of table

List of Acronyms

1D One Dimensional.

2D Two Dimensional.

ASIC Application Specific Integrated Circuit.

AWS Amazon Web Services.

BF Bloom Filter.

BTC Bitcoin.

CPU Central Processing Unit.

CR Correction Rate.

DBMS Database Management System.

FPR False Positive Rate.

GCP Google Cloud Platform.

GPU Graphics Processing Unit.

PMM Partial Memory Mapping.

ShBF Shift Bloom Filter.

Chapter 1

Introduction

1.1 Bitcoin as a cryptocurrency

Bitcoin[1] is the world's first cryptocurrency, a form of electronic cash sent peer-to-peer without the need for a financial intermediary(third party such as bank). It is the first decentralized digital currency to provide a solution to double spending problem: the system works without a central bank or single administrator. Bitcoins are sent from user to user on the peer-to-peer Bitcoin network directly, without the need for intermediaries where each Bitcoin owner transfers the coin by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin (Figure 1.1). A payee can verify the signatures to verify the chain of ownership. These transactions are verified by network nodes through cryptography and recorded in a public distributed ledger called a blockchain.

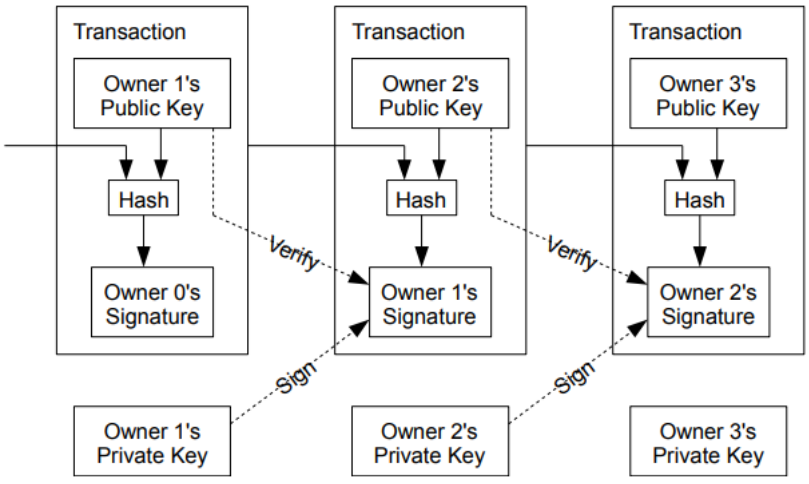


Figure 1.1: Transactions signing in Bitcoin [1]

The blockchain is a set of ordered blocks each containing a number of transactions. When adding a new block to the blockchain hash of the previous block is embedded in the new block and hash of the new block is calculated and widely published(Figure 1.2). This way if a miner wants to rollback a Bitcoin payment he has to recalculate hashes of all previous blocks. Since this consumes a lot of energy it is practically impossible to rollback transactions. In this way, the Bitcoin protocol has made it practically impossible to double spend bitcoins.

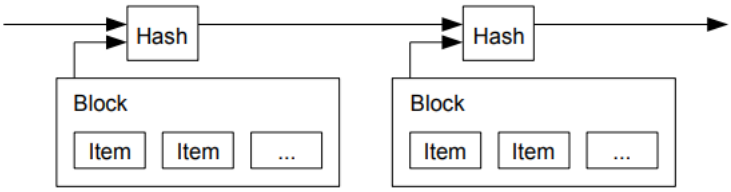


Figure 1.2: Blockchain architecture [1]

Bitcoin's[1] blockchain alone is 162 GB as of 25th March 2018 and growing rapidly. This data holds the key to measuring the privacy of cryptocurrencies in practice, studying new kinds of markets that have emerged, and understanding the non-currency applications that use the blockchain as a database.

1.2 Bitcoin blockchain analysis

Bitcoin blockchain is a publicly available data structure which contains all the transactions that took place with all relevant details such as input and output addresses, transactions amount etc. With all these information we can analyse different patterns and trends in Bitcoin. By carefully observing those data we can also identify transactions made to illegal organizations and in the best case, it is possible to identify exact users involved in those transactions as well.

There are many publicly available blockchain analysis online tools such as BlockExplorer, WalletExplorer, Blockchain etc. In addition to that, there are some offline tools such as BlockSci, GraphSense which provides all blockchain analysis tasks such as transaction graph, address clustering, querying the blockchain etc. Among them, BlockSci[2] is a standard tool widely used in blockchain analysis tasks.

BlockSci[2] is an open source software platform that enables the science of blockchain. It addresses three pain points of existing tools: poor performance, limited capabilities, and a cumbersome programming interface. BlockSci is 15x–600x faster than existing tools, comes bundled with analytic modules such as address clustering, exposes different blockchains through a common interface, imports exchange rate data and "mempool" data, and gives the programmer a choice of interfaces: a Jupyter notebook for intuitive exploration and C++ for performance-critical tasks. In addition to that BlockSci includes a blockchain parser which supports Bitcoin as well as forks of Bitcoin such as Zcash, Litecoin etc.(Figure 1.3).

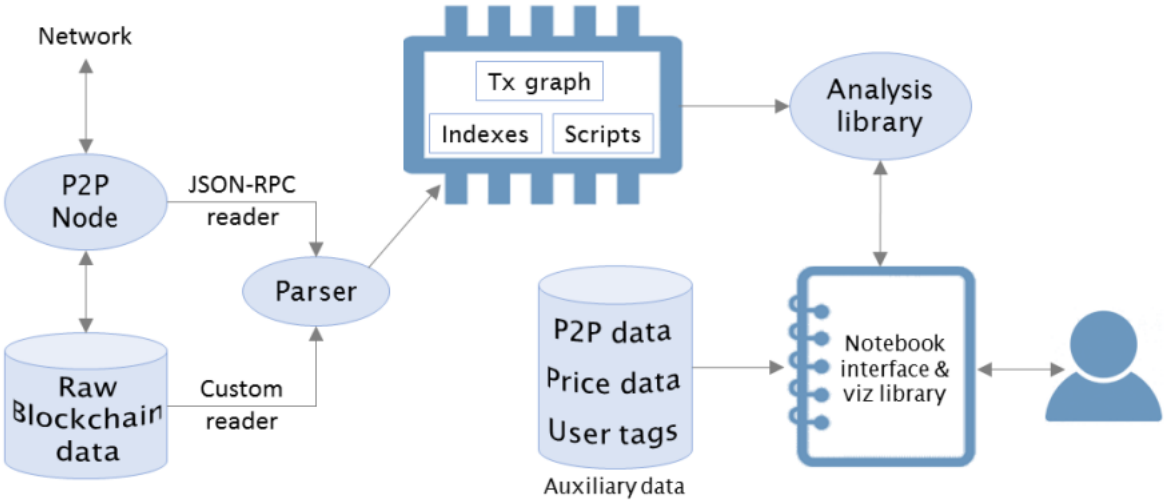


Figure 1.3: Overview of BlockSci's architecture[2]

1.2.1 Bitcoin address clustering

Even though Bitcoin claims to provide anonymity to user transactions, there have been some researches that prove otherwise. They have analysed Bitcoin transactions and come to the conclusion that Bitcoin transactions are not anonymous. They consider Bitcoin to be pseudo-anonymous that transactions of the Bitcoin cannot be traced back to its original users. But they can be tracked to the same wallets. In other words, all transactions or bitcoin addresses which originated from the same wallet can be grouped together. When combined with publicly available user information such as publicly made available addresses, tags etc. those bitcoin addresses can be traced directly back to its users. Since blockchain is publicly available this imposes a serious security threat on Bitcoin users. Attackers can use this clustering attacks to

gain personal and confidential information of users.

Recall that cryptocurrency users can trivially generate new addresses, and most wallets take advantage of this ability. Nevertheless, addresses controlled by the same user or entity may be linked to each other, albeit imperfectly, through various heuristics. Address linking[3, 4, 5] is a key step in analytic tasks including understanding trends over time and evaluating privacy. However, Bitcoin address clustering is fairly different from classical clustering problems as there is no direct information about the objects' (addresses) such as coordinates or distances. The other peculiarity of the problem is the vastness of the Bitcoin blockchain, which requires designing computationally efficient algorithms for its clustering[6, 7].

The accuracy of the bitcoin address clustering depends on heuristics being used. Basic address clustering heuristics are

1. **MultiInput Heuristic**: inputs spent to the same transaction are controlled by the same entity(Figure 1.4)
2. **Change Address Heuristic**: change addresses are not reused (Figure 1.5)
3. **Shadow Heuristic**: output address that has appeared for the first time in the history of the blockchain (Figure 1.6)

However any of these heuristic doesn't provide 100% accuracy when making clusters. Latest version of BlockSci (0.5.0) uses multi input and change heuristics to improve the accuracy of clustering. These heuristics create links (edges) in a graph of addresses. By iterating over all transactions and applying the union-find algorithm on the address graph, we can generate clusters of addresses.

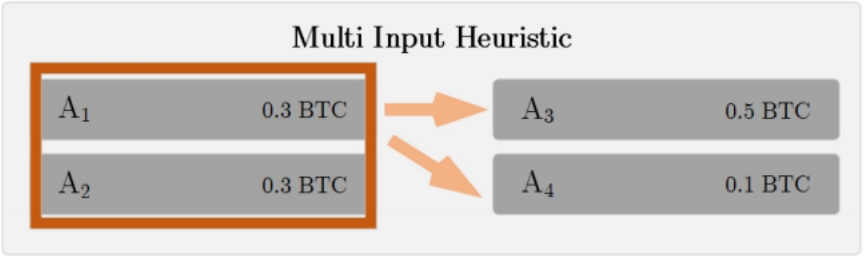


Figure 1.4: Multi-Input heuristic example. Addresses A1 and A2 are often from the same wallet

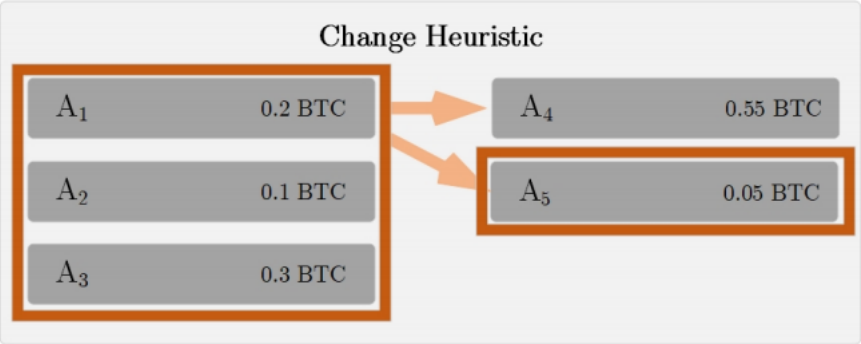


Figure 1.5: Change heuristic example. Addresses A1, A2, A3 and A5 are often from the same wallet

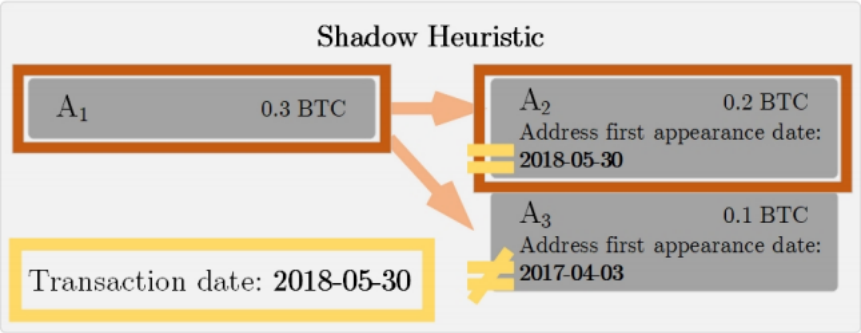


Figure 1.6: Shadow heuristic example. Addresses A1 and A2 are often from the same wallet

1.2.2 Applications of blockchain analysis

1. In e-crime investigations of illegal Tor hidden services[8], such as Silk Road, analysts often try to link cryptocurrency transactions to user accounts and activities. This can start with a known transaction that is part of a crime, such as a Bitcoin payment to buy drugs on the Silk Road [9]

2. Organizations responsible for consumer protection, such as trade commission agencies and financial regulatory authorities, have a mandate to research and identify fraud cases involving cryptocurrencies, including unlawful initial coin offerings and Ponzi schemes. Given the popularity of Ponzi schemes in Bitcoin [10, 11] blockchain analysis can be used to find pattern of user contributions to such schemes [9]

1.3 Research problem and research questions

1.3.1 Research problem

An increasing number of online merchants now offer the ability to pay using the cryptocurrency Bitcoin. One of the great promises of this technology is anonymity: the transactions are recorded and made public, but they are linked only with an electronic address. So whatever you buy with your bitcoins, the purchase cannot be traced specifically to you. This is handy for some, but the anonymity is by no means perfect. Security experts call it pseudonymous privacy, like writing books under a nom de plume. You can preserve your privacy as long as the pseudonym is not linked to you. But as soon as somebody makes the link to one of your anonymous books, the ruse is revealed. Your entire writing history under your pseudonym becomes public. Similarly, as soon as your personal details are linked to your Bitcoin address, your purchase history is revealed too.

Attackers and other malicious users can use blockchain analysis techniques to extract personal details of users [12]. Address clustering is one of the major blockchain analysis technique which allows the attackers to extract users' confidential information by tracing transactions back to owners. On the other hand, legitimate users can use this technique to evaluate the vulnerabilities of their transactions and prevent possible threats to their anonymity.

However, existing address clustering techniques focus more on the accuracy of clusters. Since Bitcoin's blockchain is a data structure with several gigabytes(165GB+) it takes a considerable amount of time to make clusters.

1.3.1.1 Blocksci clustering performance analysis

Blocksci[2] parses the Bitcoin blockchain and creates its own data format. Instead of using a database it keeps parsed blockchain data in files. When a particular query is made BlockSci analysis library reads relevant files and provide results. One of the major drawbacks of this approach is that the whole file cannot fit into the memory. Therefore there may be several IO operations to provides results to a simple query (may need to bring portions of that particular file into the memory in several rounds). This causes unnecessary delay when providing query responses. This problem can be largely seen when clustering the blockchain. Unlike most big data operations when creating clusters, for each transaction algorithm has to scan the whole blockchain to

GCP Machine Types with 16GB Memory			
n1-standard-2 (2 VCPUs)	n1-standard-4(4 VCPUs)	n1-standard-8(8 VCPUs)	n1-standard-16(16 VCPUs)
15.4 hrs	113.8 hrs	172 hrs	183.5 hrs

Table 1.1: Changing VCPU count by keeping memory constant

GCP n1-standard-8 (8 VCPUs)			
8GB memory	16GB memory	56GB memory	128GB memory
187.3 hrs	172 hrs	0.3 hrs	0.27 hrs

Table 1.2: Changing memory size by keeping VCPU count constant

verify some information. Hence there will be a huge number of IO operation which makes it inefficient on machines with low resources (especially low memory since the number of file chunks bring back and forth to the memory is high). Table 1.1 and Table 1.2 shows clustering time taken by Blocksci platform on machines with different specifications.

In addition to that one other problem associated with BlockSci's address clustering algorithm is it doesn't provide ability to dynamically update address clusters. When new blocks are added to the blockchain, clustering process should be started from the beginning. This is a cumbersome process considering the time taken by the algorithm.

1.3.1.2 GraphSense clustering performance analysis

Unlike BlockSci[2] GraphSense[13] is built on Apache Cassandra[14] and Spark[15]. Since GraphSense is built on scalable (horizontally scalable) and distributed cluster technology it can support Bitcoin analysis tasks without deprecating performance as blockchain grows. Apart from horizontal scalability, GraphSense's clustering performance is almost same as BlockSci performance. Even though it uses casandra it has to bring all transaction information when performing a clustering task. Therefore there is a higher number of IO operation involved when running the clustering algorithm.

1.3.1.3 Conclusion on clustering performance

So far we analyzed different clustering approaches proposed by different platforms such as BlockSci and GraphSense. Even though they have used file and database approach they have failed to provide an efficient solution to improve Bitcoin blockchain clustering performance. Unlike most of the big data tasks clustering requires the whole blockchain to reside in the memory in order to provide optimal performance.

This problem gets worse as the blockchain grows. Currently recommended memory size for BlockSci platform is 64GB. But with these approaches, it will require more and more memory to run efficiently as blockchain grows. Therefore in this research, we are focussing on providing a solution to improve the blockchain clustering performance (speed and resource consumption) without affecting the clustering accuracy.

1.3.2 Research questions

- Is it possible to reduce the time complexity of Union-find algorithm used in BlockSci address clustering and if so which gains can be made by doing so?
- Is it possible to improve the speed of Bitcoin address clustering with the use of probabilistic data structures?
- How Bitcoin address clustering can be made scalable without affecting accuracy and speed?
- Can clustering queries be answered efficiently by summarizing and caching blockchain data?

1.4 Justification for the research

Most of the available Bitcoin addressing linking approaches have the time complexity of O(N). Theoretically, this should take several minutes to complete even though Bitcoin blockchain contains several hundred million transactions. But when it comes to practical implementation of those approaches there are several limitations.

Bitcoin blockchain is a large data structure which is more than 170GB in size and keeps growing rapidly. This huge dataset cannot be contained in the memory at once (most machines don't have this much memory). Major Bitcoin analytics platforms such as BlockSci[2] and GraphSense[13] follow different techniques such as DBMS, files and distributes stores such as Cassandra[14] to store blockchain data. No matter which approach they follow blockchain data has to be brought into the memory as chunks from a secondary disk. This causes a large number of IO operations and even IO operations need to be performed increases linearly as available physical memory becomes smaller.

One other limitation of address linking is that some heuristics being used in address

BlockSci Clustering Algorithm	
Sequential Verison	Parallel Version
3 hrs	172 hrs

Table 1.3: Two algorithms were executed on a GCP n1-standard-16 (8 VCPUs) instance with a Linux operating system. In this experiment, we used a 16GB memory instance which is 0.25 times of BlockSci memory recommendation.

linking needs to refer all the previous blocks. As an example in change heuristic, we need to check if a particular output has appeared before in the blockchain (to determine if the current address is a new one). In the file-based approach used in BlockSci, it is done by continuously bringing chunks of transactions file to the memory. On the other hand, the DBMS approach handles this by making an addition query(to check if the address already exists) for each transaction. Either way, it involves additional IO operations which linearly increases in number as memory becomes smaller in size.

Using parallel workers in address clustering is a common approach used in almost all the major Bitcoin analytics platforms. This significantly increases the processing speed of blockchain hence increase the clustering speed. But there is an exception which this approach perform well if it has memory above a certain threshold, in BlockSci[2] it is 64GB. But as available memory becomes lesser parallel approach takes more time than the sequential one and time taken by each approach exponentially increases as memory becomes lesser(time consumption is inversely proportionate to the available memory. Table 1.3 depicts results we obtained using both parallel and sequential version of BlockSci address clustering algorithm.

As explained earlier time complexity of Bitcoin address clustering is not a problem at all. But the problems such as the size of the blockchain makes it harder to provide a better clustering algorithm which can perform well with optimized memory usage. None of the available major Bitcoin analytics platforms has been able to address this problem so far. Hence this has caused a major problem for users who are not capable of acquiring costly machines. Even though there exist clustering techniques which use DBMSs, files and even distributed store such as Cassandra[14], none of them has provided a solution to this problem. In our research, we're focussing primarily on this goal to design and develop an optimized Bitcoin address clustering approach which can also be extended to other forks of Bitcoin such as Litecoin, Zcash etc.

1.5 Methodology

Existing clustering algorithms showcase a poor performance when running on memory constrained devices (As an example it provides better performance when running on a machine with 64GB memory. But takes longer, maybe weeks when running on a machine with 16GB memory). Valgrind (C++)[16] profiling tool will be used on BlockSci clusterer tool to detect potential memory leaks, running time of each function and more importantly kernel buffer/cache usage of the tool(Bitcoin blockchain is currently 170+GB in size. Even after parsing the blockchain, the dataset required by the clustering tool is about 60GB(exclusive of index files). As memory constrained devices cannot hold the whole file in its buffer at once, identifying a buffer/cache usage policy is really important when reducing number I/O operations caused by page faults.

After identifying the exact files used by the clustering tool we will be designing a buffer replacement policy which will reduce I/O operations caused by page faults. Reason for this is larger files cannot be contained in the memory (if memory is below 64GB) at once. Therefore when clustering tool fetches transactions, it first checks memory buffer for that particular transaction. If it doesn't already exist in the memory, transactions details should be fetched from a secondary device which is an I/O operation which takes comparatively more time. By introducing a proper buffer replacement policy we can reduce the number of disk access and hence reduce the execution time of the clustering tool.

So far we talked about how we can optimize the speed of clustering algorithm by introducing a new kernel buffer replacement policy. Our statistical data shows that BlockSCi clustering tool uses around 4GB stack space when it is running. Most of the space is allocated for the address clusters. In the BlockSci implementation, they have used union by rank algorithm. Even though the algorithm usually uses a forest like data structure we will be using an array of integers so that cluster search can be performed with constant time complexity. By considering the size of Bitcoin's currently available address space we may use an array of 32-bit unsigned integers. This will reduce the stack space used by the existing clustering algorithm.

The evaluation of this methodology will be a qualitative approach as it focuses on

enhancing a current clustering method. The objective will be to assess the performance and memory consumption of the proposed method over the existing method for blockchain address clustering. The evaluation approach will be a two-fold approach.

First, we need to evaluate the actual improvement in execution time compared to existing clustering models developed by major Bitcoin analytics platforms. We have observed that the poor clustering speed of the existing address clustering algorithms lie within the memory limitations (Even though memory modules which have a higher capacity(64GB) can contain all the required blockchain information files in the memory at once, memory modules with comparatively lower capacities have to heavily rely on secondary disks). Therefore when evaluating the performance we should primarily focus on memory constrained devices.

The next most important evaluation stage is accuracy. Our concentration is only towards optimizing the time and memory consumption of the blockchain address clustering algorithm. Therefore the accuracy of address clusters should never be affected as it will require some complex analysis techniques to test the validity(to test if accuracy is increased or decreased) of the changed accuracy of the algorithm. When evaluating the accuracy of the optimized clustering algorithm we will compare clusters formed by the optimized algorithm against clusters formed by the current algorithm(BlockSci clustering algorithm).

1.6 Outline of the dissertation

The rest of the dissertation is organized as follows. Chapter two explores the significance of Bitcoin as a cryptocurrency and evaluate existing Bitcoin blockchain analysis techniques and their weaknesses. Later in the chapter, we describe union find algorithms which can be used in address linking and their suitability towards our approach. The first part of chapter three discusses a probabilistic approach to address linking and its tradeoffs. Later in the chapter, we describe an address linking approach using partial memory mapping technique proposed in our study. In chapter four it describes the implementation details of the proposed methodology. Chapter five dedicated to the results evaluation of the proposed approach. The final chapter presents the conclusion and future work.

1.7 Delimitations of scope

There are two crucial components (speed and accuracy) that we need to consider when talking about Bitcoin address clustering. Throughout this chapter, we explained the need for improving the speed of Bitcoin address clustering. Since there are several studies that have been conducted to improve the clustering accuracy we're not interested in the effect the clustering accuracy (heuristics being used to link addresses) of available approaches. In short, we will only concentrate on improving the clustering speed (time) without affecting the clustering accuracy.

Affecting the clustering accuracy of our study will lengthen the process of evaluation. In [5] they have joined mining pools, exchange services, wallets to evaluate the clustering accuracy provided by their heuristic. The reason behind this evaluation approach is that if we affect the clustering accuracy we cannot determine whether it is an increase or decrease in accuracy without doing further research. Therefore our address clustering approach will not affect the clustering accuracy and in Chapter 5 we have demonstrated that the clustering accuracy provided by BlockSci[2] is identical to the accuracy provided by our approach.

1.8 Conclusion

This chapter laid the foundations for the dissertation. It introduced the research problem and research questions and hypotheses. Then the research was justified, definitions were presented, the methodology was briefly described and justified, the dissertation was outlined, and the limitations were given. On these foundations, the dissertation can proceed with a detailed description of the research.

Chapter 2

Literature review

2.1 Bitcoin as a currency

Satoshi Nakamoto explains the need for a peer to peer electronic cash system for the digital currencies and transactions. They have proposed a solution for the double spending problem without involving any trusted third party and have addressed fundamental mechanics of Bitcoin [1] and Blockchain with dividing all the information into 12 main sections. Their solution starts with cryptographic digital signatures. They have defined an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and with the public key of the next owner. Payee needs to use their private key to verify the chain of ownership. The core of the Bitcoin is the blockchain which provides a solution to the double-spending problem. Bitcoin protocol claims because of the decentralized nature of cryptocurrency it provides anonymity and privacy to user transactions.

2.2 Privacy of Bitcoin

Even though Bitcoin [1] claims to provide privacy and anonymity to user transactions there have been some researches [5, 17, 18] on that area to prove otherwise.

Androulaki, Elli et alii [17, 19] explored the privacy implications of Bitcoin. Their findings show that the current measures adopted by Bitcoin are not enough to protect the privacy of users if Bitcoin were to be used as a digital currency in realistic settings. More specially, in a small controlled environment, clustering techniques were found suitable to unveil the profiles of Bitcoin users, even if these users try to enhance their privacy by manually creating new addresses.

The main problem, when it comes to anonymity, is that the history of all coins are publicly available and anyone can see the flow of bitcoins from address to address. [17] points out some common anonymization techniques which can be used to prevent threats to user anonymity such as 1)Randomly sending coins to new addresses generated just for this purpose. The coins are still part of the owner's balance, but it is very difficult for an outsider to prove that the owner sent the coins to himself instead of another person. However, the transaction chain still has the owner's identity in it. 2) Using a mixer, that takes the coins of many different people, mix them up, and send similar amounts back to those peoples' addresses. If the mixer keeps no logs of who gets which coins, investigations are unfeasible.

CoinShuffle [20] is such mixing service which concentrates on unlinkability, verifiability and robustness of Bitcoin transactions. [20] proposes a mixing service with 3 phases as depicted in Figure 2.1. The first phase is the ***announcement*** where every participant generates a fresh ephemeral encryption-decryption key pair and broadcasts the resulting public encryption key. Then each and every participant creates a new Bitcoin address and ***shuffle*** those addresses in an oblivious manner, similar to a decryption mix network [21]. The final phase is ***transaction verification*** where each participant can individually verify if his output address is indeed in the list. If this is true, every participant deterministically creates a mixing transaction that spends coins from all input addresses and sends them to the shuffled list of output addresses. Every participant signs the transaction with her Bitcoin signing key and broadcasts the signature. Upon receiving signatures from all other participants, every participant is able to create a fully-signed version of the mixing transaction. This transaction is valid and can be submitted to the Bitcoin network.

Even Though [17, 20] suggest that using mixing services can prevent threats to user anonymity Moser et al. [18] have proven otherwise. Moser et al. perform an active analysis using reverse-engineering to understand the mode of operation of three mixing services: Bitcoin Fog, BitLaundry and SharedCoin. They perform mix procedures for each mix service for small bitcoin values using as a destination addresses one or multiple new generated ones. Then, they visualize the transaction graph of the addresses involved in the mixing. They conclude that while in Bitcoin Fog and SharedCoin it is hard to relate input and output transactions, for the Bitcoin Fog, they found a clear structure that allows understanding of how the service works and

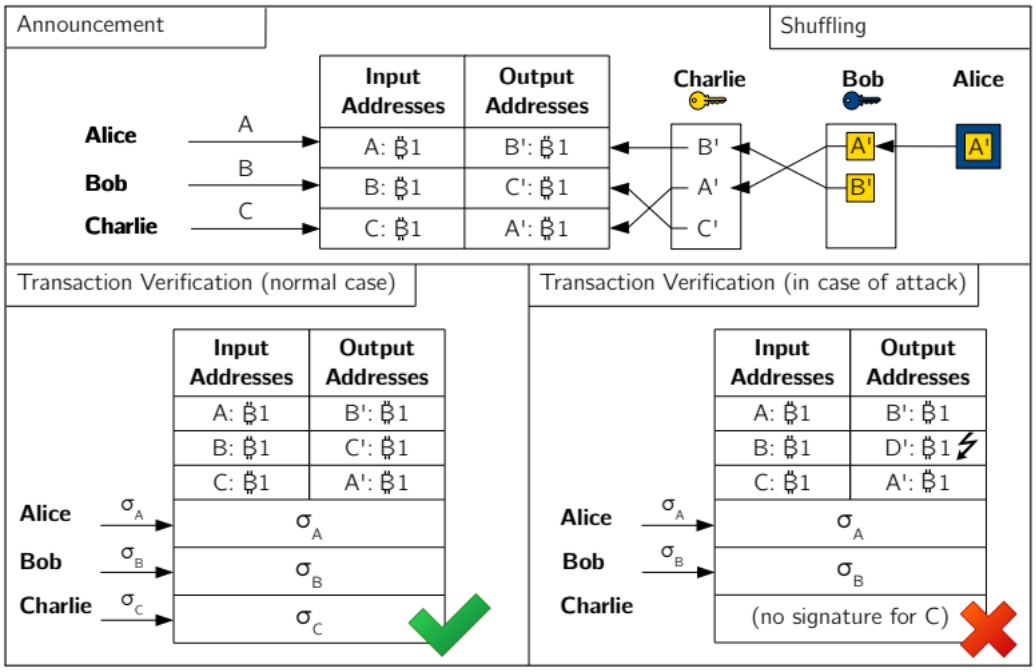


Figure 2.1: Overview of CoinShuffle [20]

may help an attacker to detect the output transactions.

The significance of [18] is that they provide possible vulnerabilities and weaknesses of popular mixing services. By considering these facts on [18] we can come to the conclusion; even though mixing services are considered to be an effective approach to protect user anonymity their functions depend on the implementation. Based on their finding different mixing services can have different vulnerabilities depending on their implementation.

In their study in analysing the anonymity provided by the Bitcoin protocol against the promised anonymity Sarah Meiklejohn et al. [5] identify certain idioms of use present in the concrete Bitcoin network implementations that erode the anonymity of the users who engage in them. In this study, they describe a re-identification attack wherein they open accounts and make purchases from a broad range of known Bitcoin merchants and service providers. Since one endpoint of the transaction is known it is possible to positively label the public key on the other end as belonging to the service. To amplify the results they have used address clustering. In this research, they introduce two address clustering heuristics which can be applied on Bitcoin transactions to form clusters. The first heuristic, treat different public keys

used as inputs to a transaction as being controlled by the same user. However, this heuristic is inherent to Bitcoin protocol and also mentioned in Satoshi Nakamoto's paper. The second is new and based on so-called change addresses; in contrast to the first, it exploits a current idiom of use in the Bitcoin network rather than an inherent property. One other important finding of their research is that they have figured out due to some reasons such as coinjoin transactions and mixing services clusters formed by each above heuristics are not 100% accurate.

Significance of [5] is that they provide a complete analysis of how clustering works and possible weaknesses of each approach. In addition to that they explain the flow which needs to be followed when verifying the accuracy of clustering heuristics. If we are to change the clustering accuracy of Bitcoin clustering algorithm we cannot verify whether it is an increase or decrease in accuracy without going through the same process that these researchers have followed.

2.3 Bitcoin address clustering

The impact of Bitcoin address clustering on user anonymity and privacy has been known for a while now [22, 5, 23, 24, 25]. In their study, Fergal et al. [22] showed that the passive analysis of public Bitcoin information can lead to serious information leakage. To show that they constructed two graphs representing transactions and users from Bitcoin's blockchain data and annotated the graphs with auxiliary data(user accounts from BitcoinTalk, Twitter etc.). The authors used visual content discovery and flow analysis techniques to investigate Bitcoin theft. Alternatively, Fleder et al. [24] explored the level of anonymity in the Bitcoin network. The authors anotated addresses in the transaction graph with user accounts collected from BitcoinTalk in order to show that users can be linked to transactions through their public Bitcoin addresses. These studies show the value of address clustering in Bitcoin privacy research and law enforcement.

Harry A. Kalodner, Steven Goldfeder, Alishah Chator, Malte Möser and Arvind Narayanan proposed an approach to link bitcoin addresses in their paper ***BlockSci: Design and applications of a blockchain analysis platform***[2]. They use address linking heuristics proposed by Meiklejohn et al. [5]: (1)inputs spent to the same transaction are controlled by the same entity and (2)change addresses are not

reused with adding an exception to heuristic 1: it isn't applicable to CoinJoin transactions. In order to accurately detect CoinJoin transactions, they use the algorithm described in Goldfeder at al. [26]. These heuristics create links (edges) in a graph of addresses. By iterating over all transactions and applying the union-find algorithm on the address graph, we can generate clusters of addresses. They do not attempt to be comprehensive, resulting in false negatives (i.e.,missed edges, resulting in more clusters than truly exist). More perniciously, most of the heuristics are also subject to false negatives (i.e., spurious edges), which can lead to "cluster collapse".

Basically, [2] do not attempt to analyze the accuracy provided by each heuristic. They propose a method to simply cluster addresses without giving much attention to accuracy or performance. However address linking is inherently imperfect, and ground truth is difficult to obtain on a large scale since it requires interacting with service providers. Many other heuristics are possible, including those that account for the behavior of specific wallets. Therefore in our study, we do not attempt to improve existing heuristics to provide a more accurate result.

However BlockSci needs all relevant blockchain data files mapped to the memory when doing an analytic task such as address clustering. This approach drastically increase time consumption when the RAM capacity is lower (below than 64GB) than a minimum threshold. One of the main drawbacks of this approach is that it requires more and more resources to analyze blockchain without a performance hit as Bitcoin blockchain grows.

Bitiodine[25] is another Bitcoin analytics platform proposed by Michele Spagnuolo et al. It provides minimal analytics capabilities such as blockchain parsing, addresses clustering and answering simple queries with regards to Bitcoin data. BitIodine adheres to the same design principles used by BlockSci[2] except, instead of using files to store parsed blockchain data, BitIodine uses relational schema. When it comes to clustering algorithm proposed by BitIodine we have observed that they have concentrated more on clustering accuracy, but not much on speeding up the clustering process.

Massimo Bartoletti et al. have proposed a general framework to create general-purpose analytic on the blockchains of Bitcoin and Ethereum[27]. The work-flow supported by their tool consists of two steps: (i) they construct a view of the blockchain,

also containing the needed external data, and save it in a database; (ii) they analyze the view by using the query language of the DBMS. This can be considered as an improvement compared to other tools such as BlockSci since they heavily rely on physical memory in different analytic tasks(including address clustering).

By considering the size of blockchain it can be seen as a good design decision and [27] have been able to reduce the time taken by various analytic tasks on low memory devices. This library supports the most commonly used external data, e.g. exchange rates, address tags, protocol identifiers, and can be easily extended by linking the relevant data sources. However, their research doesn't provide any blockchain address clustering tool which can be used to group addresses which are coming from the same wallet.

A drawback of [27] is that the database schema is fixed, hence it is not possible to use it for analytics which requires external data. While the other tools store results in secondary memory, blockparser and BlockSci[2] keep all the data in RAM. Although this speeds up the execution, it demands "big memory servers", since the size of the blockchains of both Bitcoin and Ethereum has largely surpassed the amount of RAM available on consumer hardware.

As a means of addressing both address clustering heuristics; coin spending (CS) and one-time spending (OTC), Dmitry Ermilov, Maxim Panov and Yury Yanovich propose [7] to use off-chain information as votes for address separation and to consider it together with blockchain information during the clustering model construction step[2]. In this study, a new Bitcoin address clustering algorithm is proposed. Its difference from the existing ones is two-fold. Firstly, it uses for clustering not only blockchain information but also off-chain (tags, publicly available addresses etc.) information from the Internet. Secondly, they treat certain off-chain data types as votes against address union in the clustering process. Such approach allows to avoid a significant part of erroneous cluster merges suggested by blockchain based heuristics. Numerical experiments show that the proposed approach provides reasonable clustering results outperforming approaches based solely on blockchain data in terms of cluster homogeneity. CS and OTC heuristics and the set of negative pairs L may contain erroneous information. To deal with this situation, they propose a probabilistic framework which allows specifying the confidence of different sources of data.

In [9], researchers have focussed on improving the clustering accuracy without giving much attention to overall resource consumption by the approach. However, this approach provides a better outcome(in terms of accuracy) compared to BlockSci[2, 7] address linking approach as it uses off chain information(tags etc.) as well.

Although the motivations for above clustering approaches [2] seem sound, they haven't take a considerable effort to optimize the performance of clustering model. Therefore our study focuses on improving the overall address linking performance without affecting the accuracy provided by existing address clustering heuristics

2.4 Union Find Algorithms

From a theoretical point of view using the Union-Find algorithm is close to optimal. However, for very large problem instances such as those that appear in scientific computing, this might still be too slow or it might even be that the problem is too large to fit in the memory of one processor. In our research, we have to deal with Bitcoin address space which consists of more than hundred million addresses and still growing. Hence, designing parallel algorithms is necessary to keep up with the very large problem instances that appear in scientific computing.

The first such eort was by Cybenco et al.[28] who presented an algorithm using the Union-Find algorithm for computing the connected components of a graph and gave implementations both for shared memory and distributed memory computers. The distributed memory algorithm duplicates the vertex set and then partitions the edge set among the processors. Each processor then computes a spanning forest using its local edges. In log p steps, where p is the number of processors, these forests are then merged until one processor has the complete solution. However, the experimental results from this algorithm were not promising and showed that for a fixed size problem the running time increased with the number of processors used.

The main disadvantage of [28] approach is that they are focussing more on distributed memory architectures. When running this algorithm on shared memory architectures it involves some additional overhead when merging forests generated by each processor. In addition to that [18] doesn't provide a solution to handle larger data-sets on low memory devices. Therefore this approach not much suited to our problem which is to improve the speed of creating disjoint sets.

Richard J. Anderson and Heather Woll have developed a wait-free parallel algorithm model for union find algorithms[29]. This is the address linking algorithm used by the BlockSci platform. One important result from this research is that Richard and Heather developed a solution which gives the wait-free property to data structures of union-find algorithms, meaning that each thread continues to make progress on its operations, independent of the speeds of the other threads. In this model, efficiency is best measured in terms of the total number of instructions used to perform a sequence of data structure operations, the work performed by the processors. Even though these solutions apply to a much more general adversary model that has been considered by other authors these set of algorithms provide the almost same performance as sequential union find the algorithm in the worst case(sequential machines).

The Union algorithm that they have used in this research is ranked union algorithm where each record maintains a rank, and links are made from a record of smaller rank to a record of larger rank so when it comes to Bitcoin address clustering it involves millions of unique addresses(as of 9th March 2017 there were 211,789,876 transactions which cover 244,030,115 unique addresses). Since union find algorithm implemented in research [29] uses an array data structure and each data element is mapped to a certain index [13] can be used to maintain a in memory data structure for Bitcoin address clusters.

After evaluating all the costs and benefits of [28, 29] we have come to the conclusion that [29] is ideal in terms of memory consumption and speed. Therefore when creating disjoint sets we will be using [13].

2.5 Membership Queries

Bloom filters are probabilistic data structures primarily focus on optimizing memory access patterns by probabilistically answering membership queries. Previous studies on membership queries focus on optimizing bloom filters by reducing memory accesses and hash operations. Fan et al. proposed the Cuckoo filter[30] and observed that it is more efficient in terms of space and time compared to standard bloom filters. This improvement comes at the cost of probabilistically failing when inserting a new element. Kirsch et al. proposed to use two hash functions $h1(.)$ and $h_2(.)$ to simulate k hash functions $(h_1(.)+i*h_2(.))\%m$, where $(1 \leq i \leq k)$ in order to reduce the number

of hash operations. However, this reduction of hash operations comes with increased FPR[31]. To reduce the number of memory accesses, Qiao et al. proposed to confine the output of the k hash functions within a certain number of machine words, which reduces the number of memory accesses during membership queries; but the cost again is increased false positive rate[32]. In contrast, Shift bloom filters[33] reduce the number of hash operations and memory access by about half while keeping false positive rate about the same as standard bloom filters.

Chapter 3

Design

3.1 Research methodology

Our primary focus is to optimize Bitcoin address clustering by preserving clustering accuracy. To achieve this goal design science with constructive research approach was carried out which involved in the design and development of Bitcoin address clustering approach which provides better performance in terms of speed and memory consumption with compared to existing clustering approaches adopted by major Bitcoin analytics platforms.

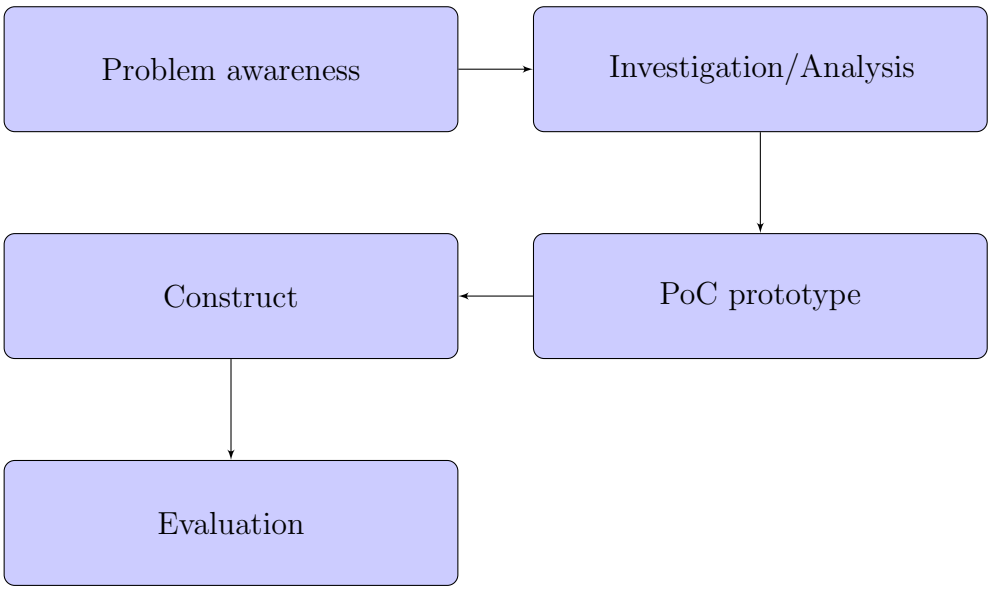


Figure 3.1: Phases aligned with mixed methodology (Design science and constructive)

Figure 3.1 shows aligned the phases of design science methodology and constructive methodology according to the purpose of this research. In design science research,

new scientific knowledge can be generated by means of constructing an artefact, and the core of this approach is a problem-solving process used to develop the artefact. Artefacts can be in the form of constructs, models, methods, instantiations, or better theories and are developed to enable a better understanding of the development, implementation, and use of information systems [34]. Below is a brief overview of the steps in design science methodology aligned with our study.

Problem awareness

As described in chapter one available Bitcoin address clustering approaches concentrate more on accuracy without giving much attention to speed(especially on machines with low resources). It has become a major problem in Bitcoin analytics tasks and the problem becomes larger as Bitcoin blockchain grows. Hence in our study, we try to find a more optimized approach to address clustering which doesn't affect the clustering accuracy.

Solution design and implementation

To design an appropriate model to address our problem we used the API set provided by BlockSci Bitcoin analytics platform. BlockSci is one of the most used tool in Bitcoin analytics and it provides almost all the service such as blockchain parsers, address cluster, graph analysis etc. Therefore it was not required to design and implement parsers and other services from scratch. Instead, we gave attention to the search for possible bottleneck associated with BlockSci that causes poor performance and finally, we were able to generalize those limitations to all major Bitcoin analytics platforms to come up with a global solution to the poor performance of Bitcoin address clustering/linking. Detailed description on design approach is presented in section 3.3.

Evaluation

Twofold evaluation will be carried out in order to evaluate the performance of our approach and accuracy being provided by the approach. At the end of the evaluation, we had to reassess the viability of each approach to determine whether we have arrived at the expected solution. By following this evaluation method we had to reject our initial address clustering model developed with probabilistic data structures as it didn't provide the accuracy as same as existing models (BlockSci & GraphSense clustering model). A detailed description of the evaluation plan and evaluation criteria are mentioned in Chapter 5.

3.2 Design analysis

One of the main question in the research design is to identify the limitations of existing address clustering models that cause its poor performance in terms of speed and memory consumption. In order to observe possible limitations, we analysed clustering models provided by major Bitcoin analytics platforms including GraphSense[13], BlckSci[2] and BitIodine[25].

3.2.1 BlockSci clustering model

BlockSci[2] has its own parser which is capable of parsing the raw blockchain and write transaction, address and indices to separate files. When clusterer tool which is capable of linking Bitcoin addresses is executed it reads each file as chunks of data in order to come up with the required information to link addresses. The cause for the performance(as depicted in Table 1.1 & Table 1.2) lies within this file based approach as we explained with details in Chapter one.

3.2.2 GraphSense clustering model

Unlike BlockSci GraphSense doesn't have its own parser. Instead, it uses BlockSci parser to parse the Blockchain. Significant of their model is that GraphSense doesn't rely on data files. It again read parsed data from files, denormalizes them and inserted into Cassandra which is a distributed key-value store. This approach enables querying of data with a slight performance gain. GraphSense has its own address clustering model which differentiates from BlockSci only by key-value store based approach used by it. As explained in chapter one this model suffers from memory not being sufficient to hold the whole blockchain problem. As it requires an unnecessary number of IO operations to complete clustering this model takes lots of time to complete on machine with comparative low resources.

3.2.3 BitIodine clustering model

BitIodine uses the same approach used in GraphSense except it doesn't uses a relational schema instead of distributed computing to improve performance. Therefore BitIodine inherits all performance issues faced by BlockSci and GraphSense.

3.3 Research approach

The research design has to address the following flow of activities.

1. Parse the Bitcoin blockchain and index blocks, transactions and addresses
2. Store parsed blockchain data in separate files to be used in analytic tasks
3. Load all Bitcoin addresses and transactions in the memory
4. Apply the combination of two clustering heuristics (multi-input heuristic & change address heuristic) to create clusters from transactions

We can reuse several tools developed by BlockSci[2] to parse the Bitcoin blockchain and process relevant information such as transactions and addresses(BlockSci covers point 1 & 2 above). Since BlockSci is a major Bitcoin analytics platform which has been used in many other Bitcoin-related applications, here onwards we'll be using BlockSci to evaluate/compare the performance of our approach.

3.4 BlockSci clustering deep dive

The figure 3.2 represents the abstract flow of BlockSci address clustering. In our design, we will only be handling load transactions from files, address linking and save cluster information and the rest will be handled by Blocksci[2] platform.

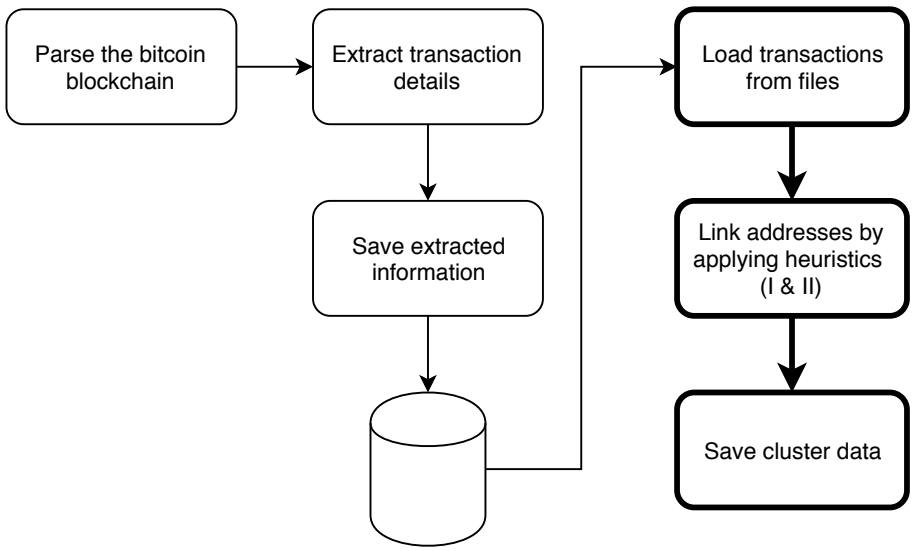


Figure 3.2: Abstract flow of BlockSci address clustering

After further analyzing BlockSci platform we were able to isolate the address linking algorithm(Algorithm 1) designed by its developers.

Algorithm 1 Address Linking Algorithm - Parallel

```
1:  procedure CreateClusters
2:      clusters := Vector()                          ▷ Initialize vector to store clusters
3:      transactions := OpenTransactions()            ▷ Open parsed blockchain files
4:      workers := createWorkers()                    ▷ Create a list of parallel workers
5:      for worker in workers do
6:          assign(worker, transactions, clusters)    ▷ Assign transactions to each worker
7:          Execute(worker, clusters)
8:      return clusters
9:
10:
11: procedure Execute(worker, clusters)
12:     transactions := worker.getTransactions()
13:     for transaction in transactions do
14:         cluster := ProcessTransaction(transaction)
15:         clusters.union(cluster)
16:     return clusters
17:
18:
19: procedure ProcessTransaction(transaction)
20:     if transaction is CoinJoin then               ▷ Ignore CoinJoin transactions
21:         return
22:     cluster := ApplyMultiInputHeuristic(transaction)
23:     changeAddress := GetChangeAddress(transaction)
24:     if changeAddress not None then
25:         cluster.union(changeAddress)
26:     return cluster
```

Our initial assumption was BlockSci's poor performance lies within the union-find algorithm being used in address clustering. They have used **union by rank algorithm**. Because of following reasons, we decided to replace union by rank algorithm with wait-free parallel algorithm model for union find algorithms[29] proposed by Richard J. Anderson and Heather Woll.

1. Blockchain contains hundreds millions of bitcoin addresses and for each of them, the algorithm needs to find if that particular address is already available and if so make the union. This takes much time as it executed in a sequential manner.

2. Data structures being used to implement the union-find algorithm is not optimized to handle concurrent operations.

3.4.1 Replacement of BlockSci union-find algorithm

After analyzing all the costs and benefits of union-find algorithms that have been reviewed in the literature review section we have decided to use [29] which provides a wait-free parallel version of the union by rank algorithm. Research [29] makes use of "Lock-free parallel disjoint set data structure" proposed by W. Jacob which provides an implementation of the lock-free parallel disjoint set data structure. Algorithm 2 and Algorithm 3 represents the union find algorithms proposed by Richard Anderson and Heather Woll[29]

Algorithm 2 Find algorithm

```
1: procedure FIND(x)
2:     while x ≠ A[x]→next do
3:         t := A[x]→next
4:         Compare&Swap(A[x]→next; t; A[t]→next)
5:         x := A[t]→next
6:     return x
```

Algorithm 3 Union algorithm

```
1: procedure UNION(x, y)
2:     TryAgain :
3:     x = Find(x)
4:     y = Find(y)
5:     if x = y then
6:         return
7:     xr := A[x]→rank ; yr := A[y]→rank ;
8:     if xr > yr or (xr = yr and x > y) then
9:         Swap(x; y); Swap(xr; yr);
10:    if UpdateRoot(x; xr; y; xr) = FAILURE then
11:        go to TryAgain;
12:    if xr = yr then
13:        UpdateRoot(y; yr; y; yr + 1);
```

Even after replacing the union-find algorithm used in Blocksci[2] we couldn't achieve considerable performance gain. Further evaluation results will be discussed in chapter five.

3.4.2 Design sequential clustering algorithm

BlockSci clustering algorithm implements parallel workers and distributes transactions among them. Even with parallel workers, it takes days to complete the execution if it is executed on machines with lesser specifications(especially memory). In order to analyze the behaviour and potential limitations of the parallel approach we designed a sequential version of the BlockSci clustering algorithm as depicted in Algorithm 4.

Algorithm 4 Address Linking Algorithm - Sequential

```
1:  procedure CreateClusters(x)
2:      transactions := OpenTransactions()          ▷ Open parsed blockchain files
3:      clusters := Vector()                        ▷ Initialize vector to store clusters
4:      for transaction in transactions do
5:          cluster := ProcessTransaction(transaction)
6:          clusters.union(cluster)                 ▷ Merge clusters with common addresses
7:      return clusters
8:
9:
10: procedure ProcessTransaction(transaction)
11:     if transaction is CoinJoin then             ▷ Ignore CoinJoin transactions
12:         return
13:     cluster := ApplyMultiInputHeuristic(transaction)
14:     changeAddress := GetChangeAddress(transaction)
15:     if changeAddress not None then
16:         cluster.union(changeAddress)
17:     return cluster
```

3.4.3 Final thoughts on BlockSci clustering

In our initial design, we tried to optimize the BlockSci address clustering model without doing any major model modification. In order to optimize the existing clustering model we tried the following modifications :

1. Replaced the existing union-find algorithm with
2. Designed the sequential version of BlockSci clustering algorithm

As per evaluation results of these modifications presented in chapter five none of them provides a significant performance improvement.

Therefore we had to come up with a robust and optimized address clustering algorithm which makes use of available memory efficiently to minimize the execution time. The latter part of this chapter presents two such approaches one which uses probabilistic data structures and partial file mapping approach.

3.5 Probabilistic approach to address clustering

As explained earlier most machines are not capable of holding the whole blockchain in memory at a given time. However, when BlockSci data files are opened, the whole file is mapped to the memory and just a chunk of it brought into the memory buffer if the file is too large for the memory. When the required information is not in the chunk brought into the memory then the whole chunk or part of it needs to be replaced with another portion of the file. This finding the part of the file that contains required information is quicker as the whole file is mapped to the memory. But there occurs an IO operation when reading the disk and bringing data into the memory.

Algorithm 5 Transaction Processing

```
1: procedure ProcessTransaction(transaction)
2:     if transaction is CoinJoin then                ▷ Ignore CoinJoin transactions
3:         return
4:     cluster := ApplyMultiInputHeuristic(transaction)
5:     changeAddress := GetChangeAddress(transaction)
6:     if changeAddress not None then
7:         cluster.union(changeAddress)
8:     return cluster
```

At line 5 of Algorithm 5 it finds for the change address. In other words algorithm checks if outputs of the transaction contain a newly generated address which has not appeared before in the blockchain. In the worst case for each output there occurs a one IO operation which needs to check if that output address has existed before by searching in primary memory as well as the secondary memory. The problem gets severe if we use a parallel algorithm for address clustering on a low memory machine. When executed in parallel each parallel worker tries to bring data from secondary memory to primary memory either by replacing existing data or filling free space. Since threads use the same address space as the parent process this cost several IO operations per output address as each worker tries to replace other workers

data causing the same data to be brought into the primary memory multiple times. This is the primary reason behind the ineffective nature of major Bitcoin analytics platforms.

3.5.1 Proposed solution

To overcome the mentioned problem and reduce the IO overhead we propose a probabilistic approach which uses BloomFilters to determine if an address has already appeared in the blockchain.

Bloom filter(Figure 3.3) is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set. False positive matches are possible, but false negatives are not – in other words, a query returns either "possibly in set" or "definitely not in set". Elements can be added to the set, but not removed

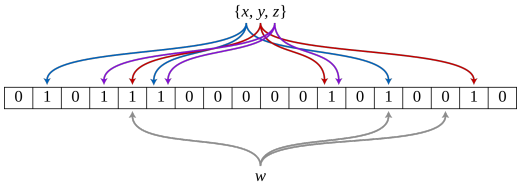


Figure 3.3: An example of a Bloom filter, representing the set x, y, z. The colored arrows show the positions in the bit array that each set element is mapped to. The element w is not in the set x, y, z, because it hashes to one bit-array position containing 0. For this figure, m = 18 and k = 3.

We used **shifting bloom filter framework** proposed in [33] to redesign our transaction processing algorithm as the number of memory accesses of shift bloom filter is smaller than most of the other variants of bloom filters[35, 36] as presented in Figure 3.4. In addition to that the correction rate of ShBF is 1.6 times and 1.79 times of that of Spectral BF[37] and CM Sketches[38], respectively.

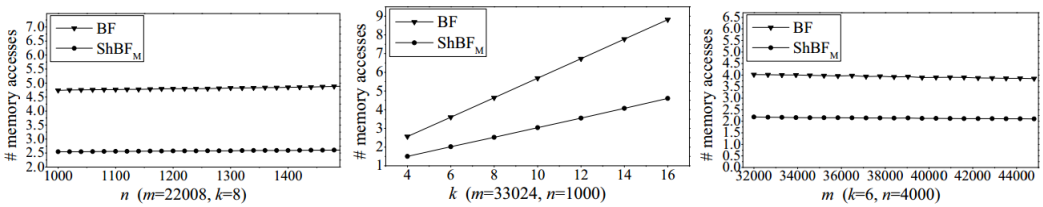


Figure 3.4: Comparison of number of memory accesses per query of ShBFM and BF

Using shift bloom filters we redesigned the transaction processing algorithm(algorithm 6) so that data doesn't have to be brought from secondary device more frequently. Evaluation results in chapter five provide further evidence to this claim.

Algorithm 6 Transaction Processing Using Bloom Filters

```
1:  procedure ProcessTransaction(transaction)
2:      if transaction is CoinJoin then                ▷ Ignore CoinJoin transactions
3:          return
4:      cluster := ApplyMultiInputHeuristic(transaction)
5:      changeAddress := GetChangeAddress(transaction)
6:      if changeAddress not None then
7:          cluster.union(changeAddress)
8:      return cluster
9:
10:
11: procedure GetChangeAddress(transaction)
12:     addresses := OpenAddresses()                   ▷ Open parsed blockchain files
13:     for address in transaction.getAddresses() do
14:         isAddressExist := BloomFilter.has(address)
15:         if not isAddressExist then                 ▷ Referring to a new address
16:             return address
17:         else                    ▷ May be a false positive. Need to check actual file.
18:             isAddressExist := addresses.get(address)
19:         if not isAddressExist then                 ▷ Address actually desn't exist
20:             return address
21:     return None
```

Using probabilistic data structures in address clustering did a greater impact on the running time of the model and has explained in detail in chapter five. However, bloom filter's performance relies on the false positive rate denoted by equation 3.1. Further details in optimizing system parameters have been specified in [33].

$$FPR = (1-p)^{k/2}(1-p+\frac{1}{w-1}p^2)^{k/2} \qquad \text{(equation 3.1)}$$

$$p = e^{\frac{-nk}{m}} \qquad \text{(equation 3.2)}$$

where:

n = # of elements of a Bloom Filter
k = # of hash functions of a Bloom Filter
m = size of a Bloom Filter

w = the maximum value of offset(.) for membership query of a single set

3.6 Optimization using partial memory mapping

As discussed in the literature review of BlockSci[2] the main problem associated with the BlockSci platform is that it requires larger memory to provide good performance. Our analysis on the BlockSci platform shows that the main reason for this limitation is it dealing with larger file size(some files are larger than 40GB and keeps growing with the blockchain).

When a large file is opened, some of it will be in RAM and some of it will be in virtual memory (an extension of physical memory which lives in Storage and is managed by the OS) as presented in figure 3.5. The operating system manages RAM and Storage. When the OS does anything with any data (be it file data or data in your program's execution space) the OS determines where chunks of memory (called pages) containing that data will be placed, when they will be swapped from storage to RAM/vice-versa, and how large a file may be based on the filesystem(s) the OS implements. The system can only do one thing per CPU core at a time, so at any given instant in time the program opening your file may not be scheduled to execute. Any file you're using is liable to be moved to/from storage and memory by the OS many times per second without you even noticing. However, the system is designed to treat virtual memory as an extension of RAM, meaning that it is not "saving" your file simply because it is making a temporary copy of some of its pages.

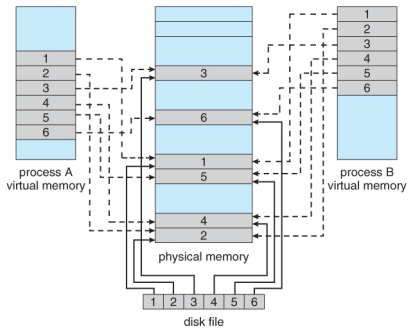


Figure 3.5: Memory mapped files

In the memory mapping[39], the entire file is mapped into the process address space such that the file is treated as an extension of virtual address space of the process.

Here, when we need a byte from the mapped file, the block (Having the byte) is directly copied to memory. If the process is not having enough frames then one of the frames is released to accommodate this new block. Hence, we may need at the worst two I/O operations (Figure 3.6). When the removed block is dirty, it needs another I/O to update dirty frame to disk.

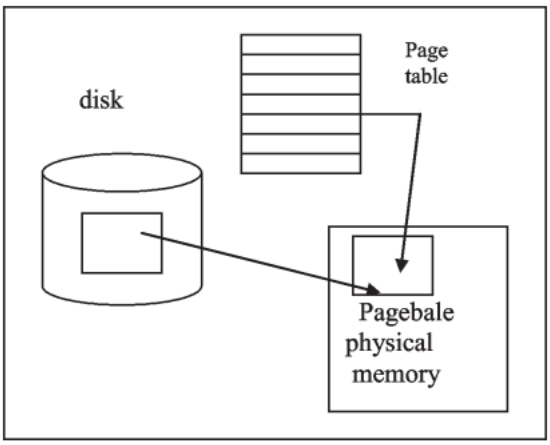


Figure 3.6: Memory mapping

As we explained earlier when executing existing clustering algorithms it performs several I/O operations which adds extra overhead as the whole file is mapped to the memory. This problem worsens as Bitcoin blockchain grows in size and more importantly when referring to previous blocks/addresses when processing transactions.

3.6.1 Proposed solution

To overcome this problem, instead of mapping the whole file into the memory at once we have decided to use a buffer to hold a part of the transaction data file along with a buffer replacement policy implementation. In short, instead of mapping the whole file, we map parts of the file to memory and completely process them before bringing forth the next part. Using this approach we can utilize the memory and time consumption of the existing address clustering model. More importantly, this allows the algorithm to be run on machines with lesser specifications(memory).

By combining partial file mapping approach with probabilistic clustering model explained in section 3.5 we can reduce most challenges such as blockchain being huge, not enough memory to hold the whole blockchain at once, IO overhead etc.

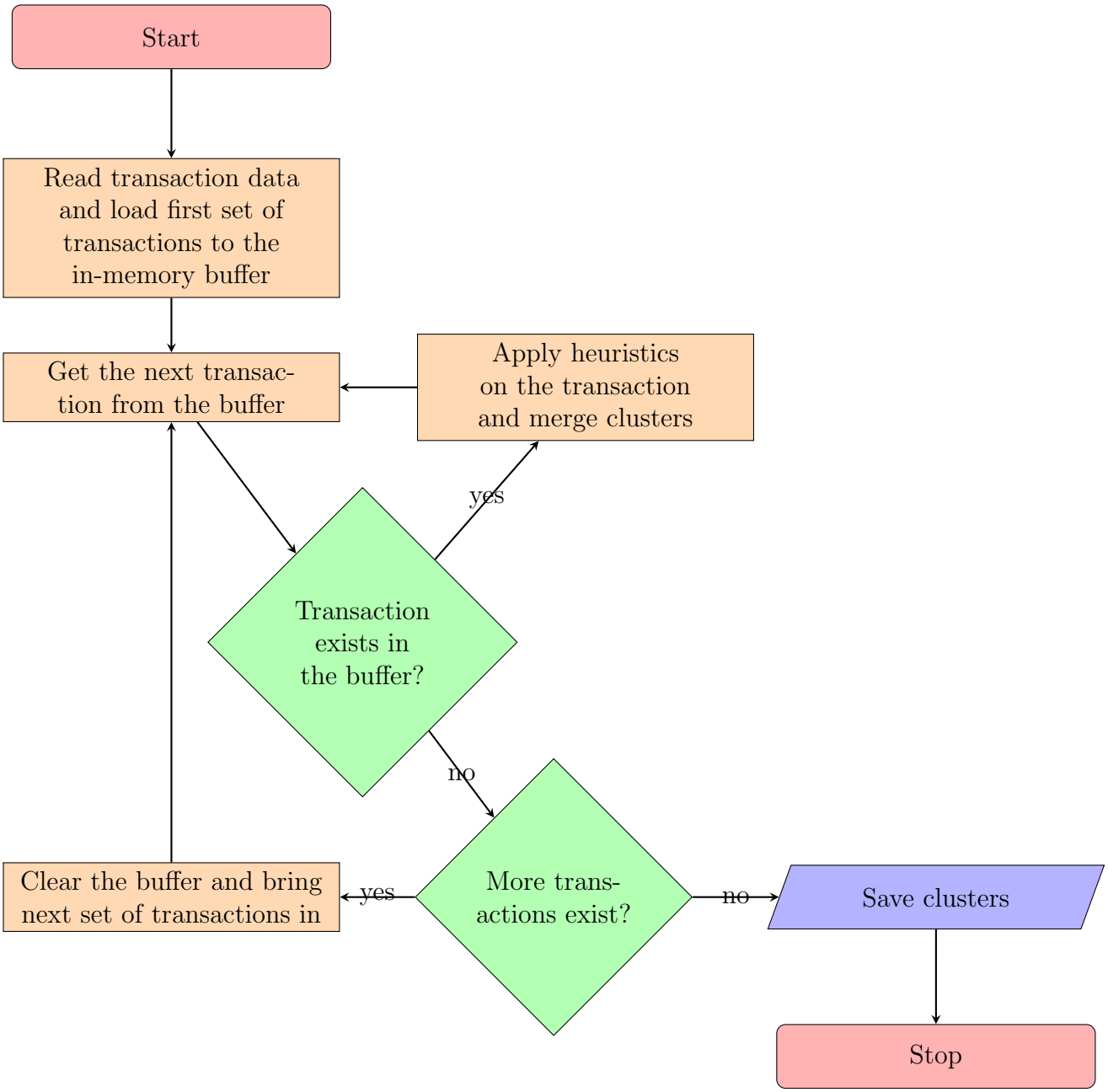


Figure 3.7: Address clustering with partial memory mapping

With design methodology showed in Figure 3.7, we were able to achieve significant performance(in terms of speed) increase compared to existing clustering models adopted by major Bitcoin analytics platforms such as BitIodine[25], BlockSci[2] and GraphSense[13]. Further evaluation details will be discussed in chapter five.

Initially, we divided the transactions file into fixed-size chunks and map each of them into the memory. After extracting all transactions in the memory mapped file chunk

we processed all the transactions by applying address linking heuristics. In order to make it more efficient, we used an ShBF[33] to answer queries on change address availability as discussed in the probabilistic model of address clustering. This is a sequential algorithm that doesn't make use of concurrent processing of the Bitcoin blockchain.

To efficiently use the available processing power we designed the parallel version of this algorithm. Instead of processing a single file chunk at once we used multiple threads to process sequence of file chunks at once. One of the critical constraints we observed in this approach is that the total size of all those file chunks should be lower than the available memory. Otherwise, it will affect the clustering speed negatively as it causes a higher number of page faults and I/O operations. We have evaluated this design idea by allocating of $\frac{1}{3}$ and $\frac{2}{3}$ of available memory to contain transactions file chunks. Chapter 05 further describes about evaluation results we obtained as a result of those design decisions.

3.7 Formal definition of heuristics

As we explained earlier our primary focus is on speed up Bitcoin address clustering without affecting clustering accuracy and we used BlockSci platform to compare the performance of our approach. Since BlockSci has used multi-input heuristic and change address heuristic we will be using same heuristics as that makes it easier to determine the improvements in our clustering algorithm with regards to BlockSci clustering algorithm.

3.7.1 First heuristic: multi-input transactions grouping

If a user wishes to perform a payment with bitcoins and the payment amount exceeds the value of each Bitcoin address owned by user's wallet then wallet has to combine several addresses owned by user to make up the amount which need to be payed in order to avoid performing multiple transactions to complete the payment, enduring losses in terms of transaction fees. Since the wallet should have the private key of each Bitcoin address in order to use them in transactions(as an input) we can safely assume that those addresses belong to the same wallet, thus to the same user.

But there is an exception to this heuristic where addresses from multiple users are used as input to a single transaction. These types of transactions are very rare and in

our study, we assume the effect of these type of transactions are negligible and since we're not focussing on accuracy component of address clustering this exception will not be an issue in our research.

More formally, let $\tau_i(S_i \rightarrow R_i) \in T$ be a transaction, and $S_i = \{a_1, a_2, \ldots, a_{n_{S_i}}\}$ the set of input addresses. Let also $|S_i| = n_{S_i}$ be the cardinality of the set. If $n_{S_i} > 1$, then all input addresses belong to the same (previously known or unknown) user: owns $(a_i) \approx u_k \forall i \in S_i$.

3.7.2 Second heuristic: change address guessing

The second heuristic is associated with determining the change address of a transaction if there is one. One of the design constraints of Bitcoin protocol is that it forces each transaction to spend, as output, the whole input. This means that the **unspent** output of a transaction must be used as input for a new transaction, which will deliver **change** back to the user. As a measure of safeguarding the anonymity of the user most wallets always generate a new Bitcoin address to collect change amount (strictly depends on the implementation of the wallet that some wallets may use a single address to make payments and collect change). Change address tries to determine change address by determining if an output address is a newly generated one. In other words if an output address has not appeared in the blockchain previously there is a greater chance that this address is a change address which belongs to the same user/wallet who owns input addresses.

More formally, let $\tau_i(S_i \rightarrow R_i) \in T$ be a transaction, and $S_i = \{a_1, a_2, \ldots, a_{n_{S_i}}\}$ the set of input addresses (with $|R_i| = n_{R_i}$ being the cardinality of the set), and let us consider $T|_{lastblock(\tau_i)}$, that is, the set T limited to the last block at the time of transaction τ_i. If $n_{R_i} = 2$, then the output addresses are a1 and a2. If $a1 \notin T|_{lastblock(\tau_i)}$ and $a2 \in T|_{lastblock(\tau_i)}$, then a1 is the shadow address, and belongs to the same user u_k who owns the input address(es): owns$(a_i) \approx u_k$.

In addition to deterministic change address determination technique mentioned above, a bug in the official Bitcoin client allowed us to further improve change heuristic method.

```
// Insert change txn at random position
int list_begin = wtxNew.vout.begin();
int n_of_payees = wtxNew.vout.size();
vector<CTxOut>::iterator position = list_begin + GetRandInt(
   ↪ n_of_payees);
wtxNew.vout.insert(position, CTxOut(nChange, scriptChange));
```

As shown in the above code segment when the Bitcoin client chooses in which slot to put the shadow address, it passes to `GetRandInt` the number of payees. Because of off-by-one error, in the common case of one payee, `GetRandInt` will always return 0, and the change always ends up in the first output.

This bug has been fixed on 30 January 2013 and for transactions occurred before that date, just 6.8% have the change address provably in the second slot of two-outputs transactions. Due to this reason, we can relax change address heuristic so that we can consider first output as the change address if a transaction contains only two outputs and that particular transaction occurred before the said date.

Chapter 4

Implementation

This chapter elaborates the implementation details of the proposed solutions.

4.1 Software Tools

Since BlockSci's core infrastructure is written in C++ in order to modify the core platform we had to use the same language. Python bindings and the Jupyter notebook interface has been used in analytics tasks. Valgrind was used for memory debugging, memory leak detection, and profiling of the BlockSci source code. Hardware configurations are simulated by using Google Cloud Platform (GCP) virtual machines. Docker was used to deploying modified BlockSci on a GCP instance.

4.2 BlockSci Clustering

In this section, we will talk about modifications that we performed on BlockSci core in order to utilize memory and time consumption of BlockSci clustering approach.

4.2.1 Replacement of BlockSci union-find algorithm

As explained in chapter three our initial assumption was that the poor performance of BlockSci clustering was lies within its union-find algorithm implementation being used. Therefore we replaced the union by rank algorithm which was being used by BlockSci[2] by **wait-free parallel algorithms** proposed in [29].

Find algorithm was implemented using path-halving such that as we traverse a path to root, we make each node we visit point to its grandparent. This method halves the distance of each node to the root and this is a heuristic based approach

where compare and swap technique is used to assign the new value to the next field. This solves the problem of two or more threads concurrently updating the same pointer.

```
uint32_t find(uint32_t id) const {
    while (id != parent(id)) {
        uint64_t value = mData[id];
        uint32_t new_parent = parent((uint32_t) value);
        uint64_t new_value =
        (value & 0xFFFFFFFF00000000ULL) | new_parent;
            /* Try to update parent (may fail, that's ok) */
        if (value != new_value)
            mData[id].compare_exchange_weak(value, new_value
              ↪ );
        id = new_parent;
    }
    return id;
}
```

Union algorithm is the same as the union by rank algorithm where each element maintains a rank and links are made from an element of smaller rank to an element of larger rank. If two elements of equal ranks are linked then the rank of new root will be incremented. The height of the tree is determined by the logarithmic in the number of vertices.

Several difficulties arise in a concurrent implementation of Union. The key idea that prevents cycles is to perform links in a direction consistent with our ordering on records. If x and y are roots with $\boldsymbol{x} < \boldsymbol{y}$, then we link from x to y. A subtler problem that arises is to ensure that different threads have consistent views of the data. We must prevent one thread from viewing $\boldsymbol{x} < \boldsymbol{y}$ and another viewing $\boldsymbol{y} < \boldsymbol{x}$. We do this by aborting a link from x to y if the rank of x changes between when it was identified as the lesser of the two roots, and when the assignment is to take place.

Parallel workers may try to increment the rank simultaneously. Therefore our implementation guard against this issue by using UpdateRoot, updating the rank only succeeds if a worker is the first to update the rank of the new root, and if the new root is still root. We get adjacent nodes with the same rank if the new root ceases to be the root before its rank is updated.

```
uint32_t unite(uint32_t id1, uint32_t id2) {
        for (;;) {
            id1 = find(id1);
            id2 = find(id2);

            if (id1 == id2)
                return id1;

            uint32_t r1 = rank(id1), r2 = rank(id2);

            if (r1 > r2 || (r1 == r2 && id1 < id2)) {
                std::swap(r1, r2);
                std::swap(id1, id2);
            }

            uint64_t oldEntry = ((uint64_t) r1 << 32) | id1;
            uint64_t newEntry = ((uint64_t) r1 << 32) | id2;

            if (!mData[id1].compare_exchange_strong(oldEntry
               ↪ , newEntry))
                continue;

            if (r1 == r2) {
                oldEntry = ((uint64_t) r2 << 32) | id2;
                newEntry = ((uint64_t) (r2+1) << 32) | id2;
                /* Try to update the rank (may fail, that's
                   ↪ ok) */
                mData[id2].compare_exchange_weak(oldEntry,
                   ↪ newEntry);
            }

            break;
        }
        return id2;
    }
```

4.2.2 Sequential clustering algorithm

To further analyse the performance issues in BlckSci[2] we designed and implemented the sequential version of BlockSci clustering algorithm. Instead of creating multiple workers that process transactions parallelly our implementation uses one thread/-worker which process all transactions by itself. Following code segment shows the

modifications that we performed on BlockSci clustering algorithm in order to execute it sequentially.

```
AddressDisjointSets ds(totalScriptCount, std::move(
    ↪ addressStarts));

uint32_t totalTxCount = 0;
for (auto &block : chain.blocks()) {
    totalTxCount += block.size();
}

uint32_t txNum = 0;
for (auto &block : chain.blocks()) {
    RANGES_FOR(auto tx, block) {
        auto pairs = processTransaction(tx, changeHeuristic)
            ↪ ;
        for (auto &pair : pairs) {
            ds.link_addresses(pair.first, pair.second);
        }
        txNum++;
    }
}
```

4.3 Probabilistic approach to address clustering

Initially, we tried replacing ranked union algorithm used by BlockSci with wait-free algorithms and modified the algorithm to work in a sequential manner in order to analyse possible causes to the poor performance. Since none of them provided satisfactory results we moved to a more robust approach which uses probabilistic data structures in our case it is a variant of bloom filters called **shift bloom filters**[33].

4.3.1 Implementation of shift bloom filter

Construction phase of shift bloom filters follow three steps :

1. Construct an array of m bits, where each bit is initialized to 0
2. To store the existence information of an element e of set S, we calculate $\frac{k}{2}$ hash values $h_1(e)\%m, h_2(e)\%m, \ldots, h_{k/2}(e)\%m$ and the offset values for the element e of set S as the auxiliary information for each element, namely $o(e) = h_{\frac{k}{2}+1}(e)\%(\overline{w} - 1) + 1$

3. Set the $\frac{k}{2}$ bits $B[h_1(e)\%m], \ldots, B[h_{\frac{k}{2}}(e)\%m]$ to 1 and the other $\frac{k}{2}$ bits $B[h_1(e)\%m + o(e)], \ldots, B[h_{\frac{k}{2}}(e)\%m + o(e)]$ to 1

Following code segment shows the implementation details of ShBF construction phase.

```
k_unrounded = K_OPT_SHBF_M * ((float)m / n) / 2;
k_floor = (int)k_unrounded;
k = ((k_unrounded - k_floor) >= 0.5) ? (k_floor + 1) : (
    ↪ k_floor);
if (k == 0) { k = 1; }

B_size = m + (W - 1);
B_size = ((B_size % 8) == 0) ? (B_size / 8) : ((B_size / 8)
    ↪ + 1);

shbf_size = B_size + sizeof(ShBF);

shbf = palloc0(shbf_size);
shbf->B_length = B_size;
```

Query phase of the ShBF decides the existence of a given element within the ShBF. Given a query x:

1. Read the two bits $B[h_1(x)\%m]$ and $B[h_1(x)\%m + o(x)]$. If both bits are 1, then read bits $B[h_2(x)\%m]$ and $B[h_2(x)\%m + o(x)]$; otherwise, output $x \notin S$ and the query process terminates.

2. If $\forall i, 1 \leq i \leq \frac{k}{2}$, $B[h_i(x)\%m]$ and $B[h_i(x)\%m + o(x)]$ are 1, then output $x \in S$.

```
MurmurHash3_x64_128(e, strlen(e), MURMUR_HASH_SEED, &hva);
offset_hash = hva[0];
offset_value = (offset_hash % (shbf_m->w - 1)) + 1;
for (i = 1; i <= shbf_m->k; i++) {

    i_hash = hva[0] + i*hva[1];
    first_index = i_hash % shbf_m->m;
    second_index = first_index + offset_value;

    if (get_bit_ShBF(shbf_m, first_index) != 1 ||
        get_bit_ShBF(shbf_m, second_index) != 1)
        return 0;
}
return 1;
```

Insertion of elements in ShBF behaves the same way as standard bloom filters(replacing each bit by a counter). To insert an element x into the shift bloom filter, instead of setting k bits to 1, we increment each of the corresponding k counters by 1; In other words we increment both $C[h_i(x)\%m]$ and $C[h_i(x)\%m + o(x)]$ by 1 $\forall i, 1 \leq i \leq \frac{k}{2}$

We have not implemented the delete operation of shift bloom filter as we're not using delete operation to delete an address.

```
MurmurHash3_x64_128(e, strlen(e), MURMUR_HASH_SEED, &hva);

offset_hash = hva[0];
offset_value = (offset_hash % (shbf_m->w - 1)) + 1;

for (i = 1; i <= shbf_m->k; i++) {

    i_hash = hva[0] + i*hva[1];

    first_index = i_hash % shbf_m->m;
    second_index = first_index + offset_value;

    set_bit_ShBF(shbf_m, first_index);
    set_bit_ShBF(shbf_m, second_index);
}
```

4.3.2 Transactions processing using bloom filters

As explained in chapter three we implemented transaction processing with the use of shift bloom filters. As the first step, we instantiated a new ShBF that holds all the addresses from Bitcoin blockchain. Following code, segment describes implementation details.

```
ShBF* shbf;

void construct_shbf() {
    uint64_t address_count = 0;
    for(auto address_type : blocksci.address_type.types)
        address_count += chain.address_count(address_type);

    shbf = new_ShBF(BLOOM_FILTER_SIZE, address_count);
}
```

Instead of trying to find the change address by simply reading memory mapped files each time, using this approach for each address in the transaction we first check if the address exists in the bloom filter, Absence of address in the ShBF means address hasn't appeared in the blockchain before. But if the address exists in ShBF it may be a false positive, hence need to check with the actual data file. After processing each transaction we insert all addresses included in the transaction to the ShBF as shown below.

```
for(auto output : tx.outputs) {
    int address_exists = query_ShBF(shbf, output.address.
       ↪ address_string)
    insert_ShBF(shbf, output.address.address_string)
    if(!address_exists) {
        change = output.address;
        break;
    }
}
for(auto intput : tx.inputs)
    insert_ShBF(shbf, input.address.address_string)
if(change == null)
    change = changeHeuristic.uniqueChange(tx)
if (change)
    pairsToUnion.emplace_back(change->getAddress(),
       ↪ firstAddress);
```

4.4 Implementation of partial memory mapped files

The objective of memory mapping files is to increase I/O performance. Memory mapping a file creates a pointer to a segment in virtual memory and the actual loading is performed by the Operating System one page at a time. For large files, this is much faster than using traditional methods in C such as `fopen/fread/fwrite`. We used Boost memory mapping library to partially memory map blockchain files and process them.

```
file->open(path, boost::iostreams::mapped_file::mapmode::
   ↪ readwrite);
const_data = file->const_data();
dataPtr = file->data();
```

Instead of mapping the whole BlockSci transactions file(as shown in above code segment) which is larger than 60GB, we mapped part of the file to the memory and completely processed each part before fetching another part from the disk. Instead of using C++ Boost library we used **MIO** a cross-platform header-only library for memory mapped file IO as MIO provides an easier interface and less more complicated than Boot. Initially, we divided transactions into fixed size file chunks and map them into the memory as shown in the below code segment.

```
std::error_code error;

for(i = 0; i < FILE_SIZE; i+=CHUNK_SIZE){
        mio::mmap_sink transaction_mmap = mio::
           ↪ make_mmap_sink(path, 0, mio::map_entire_file,
           ↪ error);
        if (error) { return handle_error(error); }

        // Iterate through the mapped region
        for (auto& transaction : transaction_mmap) {
        process_transaction(transaction);
        }

        // Remove the mapping, after which rw_mmap will be
           ↪ in a default
    // constructed state
    transaction_mmap.unmap();
}
```

Next, we implemented the parallel version of this algorithm in order to utilize processor to further improve clustering speed. Instead of choosing file chunks which are $\frac{2}{3}$ of available memory in size, we choose several file chunks that can fit into $\frac{2}{3}$ of available memery. Then we map the set of chunks to the memory at once and process them parallelly. Evaluation results of this approach will be discussed in the next chapter.

Chapter 5

Results and Evaluation

Since BlockSci is a standard tool used in Bitcoin analytics tasks we decided to use BlockSci clustering algorithm as the basis for our address clustering model. As explained earlier one of the main problems with BlockSci address clustering was it doesn't perform well on memory constrained devices. Because of this reason, BlockSci takes more time to complete its clustering process as available memory becomes lesser as depicted in Figure 5.1.

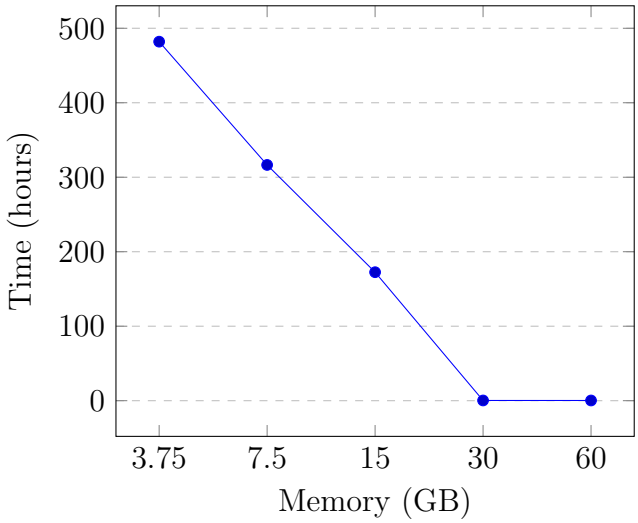


Figure 5.1: BlockSci clustering time on devices with varying memory. **n1-standard-8** GCP instances with 8 VCPUs have been used with different memeory configurations

To further analyse this performance issue we replaced the ranked union algorithm used in BlockSci[2] with wait-free union find algorithms proposed by Richard Anderson and Heather Woll[29]. Results we obtained from this approach can be seen in Table 5.1. Replacement of ranked union algorithm caused a significant improvement in BlockSci address clustering. Even though in low memory GCP instances clustering time re-

n1-standard-8 GCP instances with 8 VCPUs			
Memory: 7.5GB	Memory: 15GB	Memory: 30GB	Memory: 60GB
245.5 hrs	143.5 hrs	0.28 hrs	0.18 hrs

Table 5.1: Clustering time with wait-free union algorithms. VCPU count remains constant throughout the experiment

duced by several hours we can't see much reduction in clustering time in high memory instances such as 30GB and 60GB GCP instances. However, this was expected as I/O overhead is an additive component and reducing number of I/O operations should not linearly reduce the clustering speed.

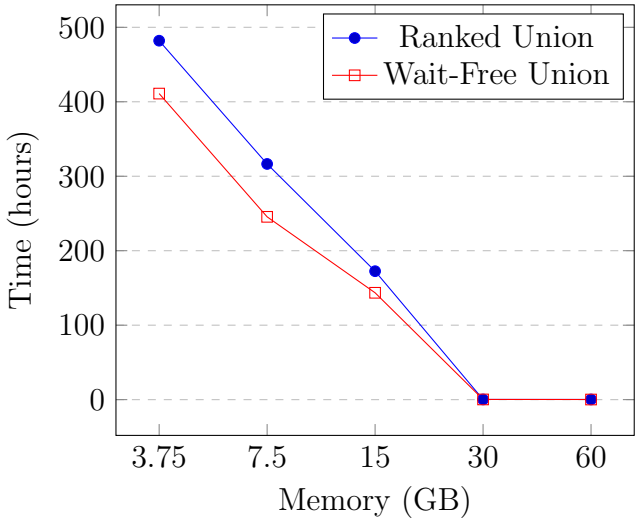


Figure 5.2: BlockSci clustering time before and after replacing union algorithms on devices with varying memory. **n1-standard-8** GCP instances with 8 VCPUs have been used with different memeory configurations

.

Even though the results we obtained from above experiment comparatively better(as shown in Figure 5.2 they were not at a satisfactory level as there is a huge gap (in days) in clustering time across high and low memory instances. In order to further observe performance issues, we implemented the sequential version of BlockSci clustering algorithm. Results we obtained from sequential clustering algorithm implementation can be seen in Table 5.2.

n1-standard-8 GCP instances with 8 VCPUs			
Memory: 7.5GB	Memory: 15GB	Memory: 30GB	Memory: 60GB
3.25 hrs	3 hrs	1.75 hrs	0.38 hrs

Table 5.2: Clustering time of sequential version of BlockSci clustering algorithm. All experiments were carried out on n1-standard-8 GCP instances with 8 VCPUs. VCPU count remained constant throughout the experiment.

This experiment provided a promising result with compared to original BlockSci clustering algorithm and modified clustering algorithm which uses wait-free parallel union algorithms. The sequential version of the clustering algorithm doesn't provide the same performance(as original BlockSci clustering algorithm and the modified version of it which uses wait-free union algorithms) in high memory instances. But this is acceptable as parallel version has several workers working on Bitcoin transaction data while the sequential version of the algorithm only has one worker. However, when it comes to memory constrainted GCP instances (such as 16GB, 8GB etc.) we can see a huge reduction in clustering time as shown in Figure 5.3.

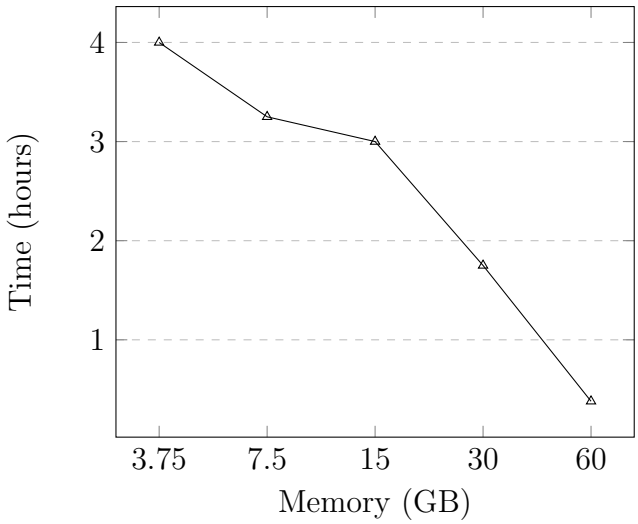


Figure 5.3: Clustering time of sequential version of BlockSci clustering algorithm on devices with varying memory. **n1-standard-8** GCP instances with 8 VCPUs have been used with different memeory configurations

.

Above observations provided further evidence to our claim that the poor performance of BlockSci address clustering algorithm lies within the inefficient use of memory. When the number of parallel workers active on a memory constrained device getting high there may occur a higher number of page faults which require data to be brought

from a secondary device. As the Bitcoin blockchain is a large data structure which is 170+Gb in size, the above problems become worse which results in a higher number of I/O operations per transaction.

With regards to above BlockSci clustering analysis results, we could come to the conclusion that the poor performance (in terms of speed) of Bitcoin address clustering lies within memory constraints of devices. As explained in chapter three we had to come up with a more robust solution to the above problem. Our next approach was using probabilistic data structures to optimize memory access patterns of the clustering algorithm in order to reduce page faults and cache line replacements.

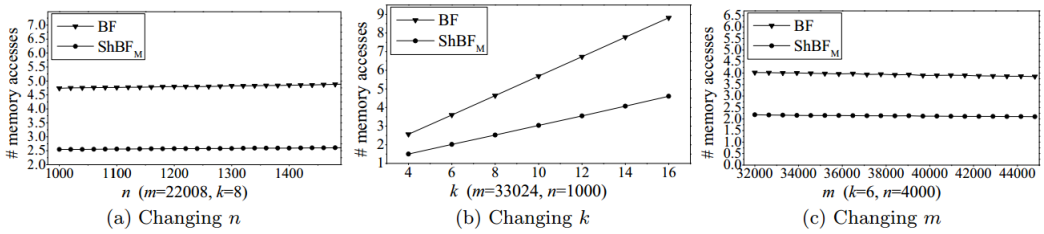
5.1 Probabilistic approach to address clustering

In order to reduce page faults and memory accesses we needed a data structure which has summarising capabilities. Since the size of Bitcoin address space is larger we couldn't use a simple hash table as it will reduce the space allowable to the clustering algorithm. To address this problem we used Shift bloom filters; a variant of bloom filter which is optimized for reduced memory access, FPA(given by equation 5.1) and increased query speed.

$$FPR = (1-p)^{k/2}(1-p+\frac{1}{w-1}p^2)^{k/2} \qquad \text{(equation 5.1)}$$

$$p = e^{\frac{-nk}{m}} \qquad \text{(equation 5.2)}$$

We have evaluated the performance comparison of ShBF aginst BF(Figure 5.4) and observed that ShBF requires a fewer number of memory accesses per query. Changing of the size of a bloom filter(m) and the number of elements of the bloom filter doesn't affect memory accesses per query. But the number of memory accesses required for a query increase with the number of hash functions being deployed(k).



(a) Changing n (b) Changing k (c) Changing m

Figure 5.4: Comparison of number of memory accesses per query of ShBFM and BF[33]

5.1.1 Evaluation of probabilistic clustering

We compared the performance(speed) of the newly designed address clustering algorithm by changing the number of hash functions used in shift bloom filters. As shown in Figure 5.5 clustering time reduces as the number of hash functions being used in ShBF getting high in numbers. Even though a higher number of hash calculations cause increased memory access it also reduces the false positive rate. This reduces the number of disks accesses and page faults mostly caused by parallel threads. Therefore even with increased memory access, we can expect a reduction in clustering time as we increase the number of hash calculations.

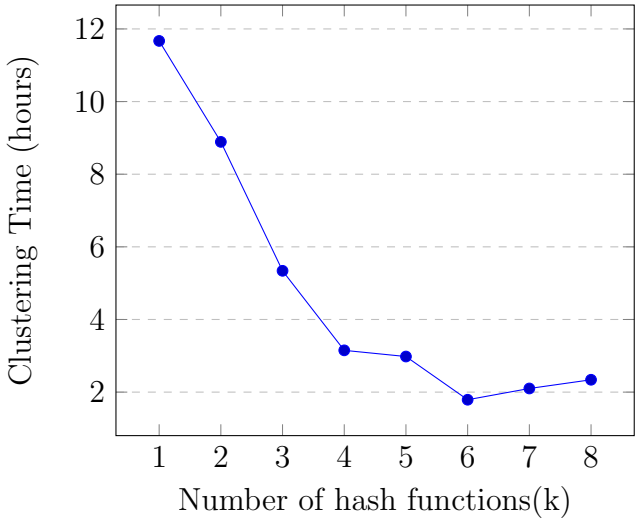


Figure 5.5: Clustering time of probabilistic clustering approach. **n1-standard-8** GCP instances with 8 VCPUs and 15Gb memory have been used with different number of hash functions(k) in ShBF

.

This reduction persists until we reach $k = 6$. However, when we further increase the number of hash calculation we could observe an increase in clustering time. Here we assume that this is caused by increasing number of memory accesses has a higher overhead when compared to page faults and disk accesses.

The reason behind this design of probabilistic clustering approach was to use summarization and caching in order to efficiently answer clustering questions. As explained earlier the main reason behind this poor performance of address clustering was there is a higher number of I/O operations followed by a higher number of page faults. By using ShBF we were able to improve address clustering to some extent. However,

the false positive rate of ShBF directly affects the clustering performance as shown in Fgiure 5.6. In this approach, if a membership query to ShBF returns false that means the address actually doesn't exist. But if the query return true that may be a false positive, hence we have to confirm it with actual data which may cause a page fault.

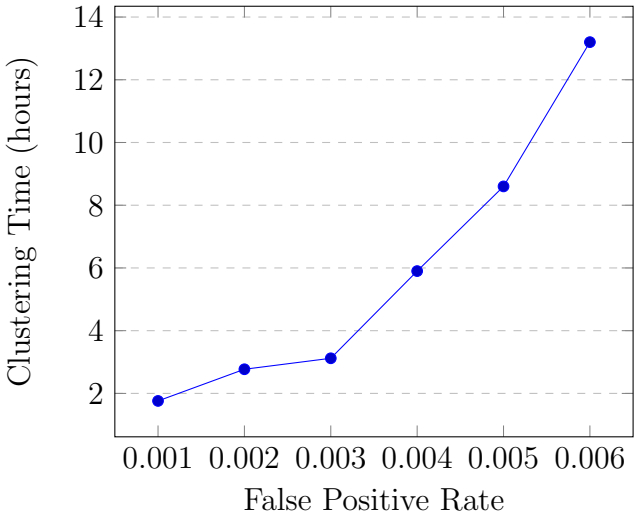


Figure 5.6: Relationship between clustering time in probabilistic clustering and false positive rate. **n1-standard-8** GCP instances with 8 VCPUs and 16Gb memory have been used with different FPR values for ShBF

.

5.1.2 Evaluation of clustering accuracy

As we explained in previous chapters our main focus is on improving the Bitcoin address clustering speed without affecting the accuracy being provided. In order to verify that our probabilistic clustering approach adheres to the above constraints, we followed a two-fold evaluation plan.

1. Since we used BlockSci clustering tool as the basis for our research we compared clusters formed by BlockSci against clusters formed by our algorithm. This was easily compared as BlockSci and our clustering algorithms use the same data structure(vector) to store cluster information. In addition to that, both algorithms use the same union algorithm and access Bitcoin transaction data in the same order. Therefore by simply comparing the 2 vectors formed by algorithms, we observed that they have identical data. Therefore we can come to the conclusion that BlockSci clustering tool and our probabilistic address clustering model provides identical clustering information.

2. As an optional evaluation step to further prove accuracy evaluation results we obtained by comparing cluster vectors we plotted the graph(Figure 5.7) of the number of clusters formed at a given time at the execution of clustering algorithms. **In order to compare the number of clusters formed at a given time, we had to execute both algorithms with only one worker(sequentially)**. The reason behind this is that parallel workers may process transaction data non -deterministically, hence cluster formation can be random.

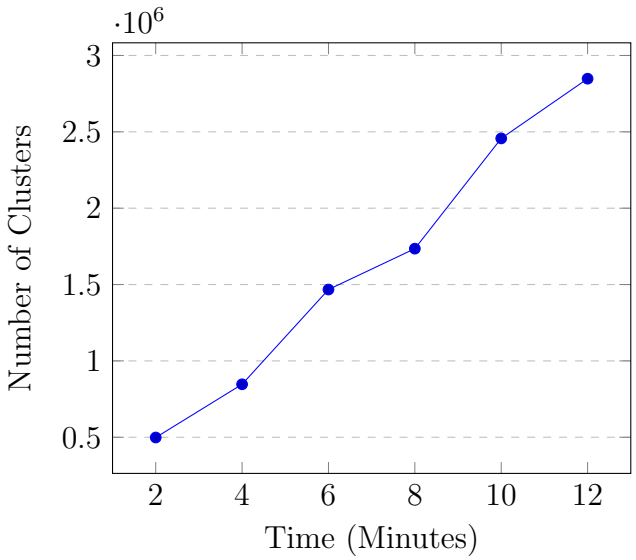


Figure 5.7: Cluster formation at algorithm execution(BlockSci and probabilistic clustering algorithms). **n1-standard-8** GCP instances with 8 VCPUs and 60Gb memeory have been used to execute both BlockSci and probabilistic clustering algorithms.
.

Both algorithms provide identical plots as shown in Figure 5.7. That further proves that accuracy provided by both Blocksci and probabilistic algorithms are identical.

By analyzing all accuracy-related evaluation results we can come to the conclusion that clustering accuracy of our algorithm is same as the accuracy provided by BlockSci clustering tool. Here we don't try to find the actual accuracy percentage provided as we used BlockSci clustering tool as the basis for our study and the accuracy of our model should only be compared with the accuracy of BlockSci clustering model.

Algorithm \ Memory(GB)	7.5	15	30	60
BlockSci Clustering(Parallel)	245.5 hrs	143.5 hrs	0.28 hrd	0.18 hrs
BlockSci Clustering(Serial)	3.25 hrs	3 hrs	1.75 hrs	0.38 hrs
Probabilistic Clustering Model	4.38 hrs	3.15 hrs	1.26 hrs	0.29 hrs

Table 5.3: Clustering time comparison actoss different clustering algorithms. All experiments were carried out on n1-standard-8 GCP instances with 8 VCPUs. VCPU count remained constant throughout the experiment.

5.1.3 Comparison of probailistic clustering

Initially, we evaluated the performance of BlockSci clustering algorithm and its sequential version. In this section, we are going to compare the performance of above methods against our address clustering model which uses Shift bloom filters summarize and cache Bitcoin addresses in order to reduce disk accesses along with page faults. As shown in Table 5.3 serial version of BlockSci clustering algorithm and the clustering algorithm developed using probabilistic data structures(ShBF) has similar performance results(Figure 5.8). Both of these algorithms provide better results on memory constrained GCP instances(7.5GB and 15GB). However, parallel BlockSci clustering algorithm provides better results in high memory GCP instances. This further highlight the poor performance of BlockSci clustering algorithm and prove our claim that this performance issue is caused by the higher number of I/O operations resulted by the larger size of Bitcoin blockchain.

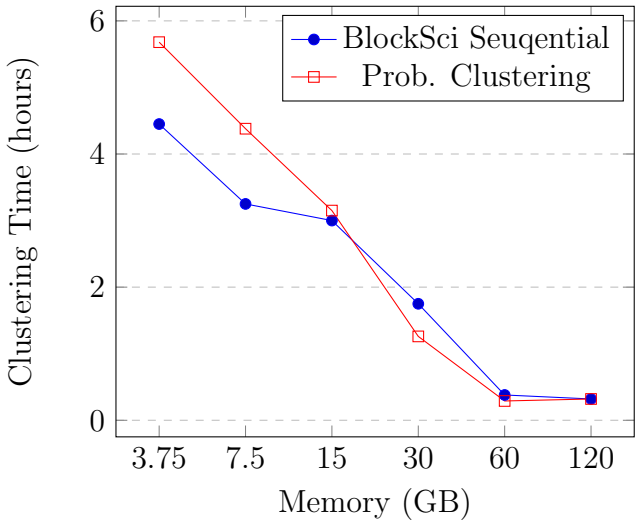


Figure 5.8: Performance comparison of sequential and probabilistic clustering models. All experiments were carried out on n1-standard-8 GCP instances with 8 VCPUs.

.

5.2 Evaluation of partial memory mapping

To further reduce page faults and unnecessary cache line replacement in order to improve Bitcoin address clustering we developed a clustering model which uses probabilistic data structures along with memory mapped files. This approach uses bloom filters as the same way it was used in probabilistic address clustering model. Initially, we divided transactions file into chunks of fixed size and map each of them to the memory one at a time. Figure 5.9 shows time taken by the algorithm when using fixed size file chunks on GCP instances with different memory configurations.

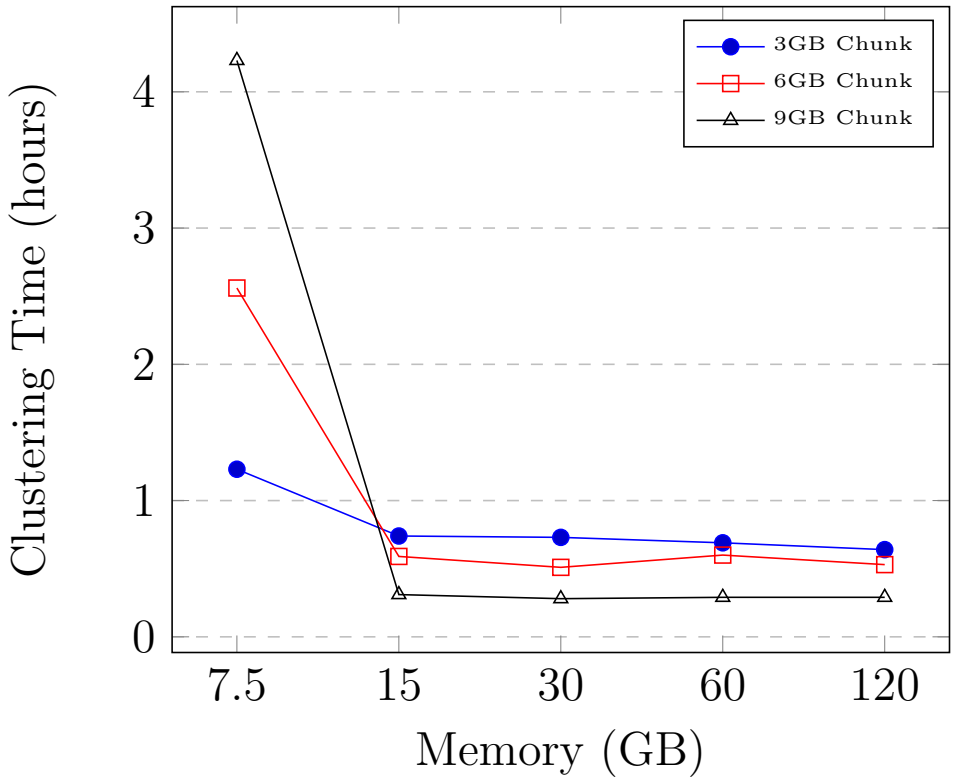


Figure 5.9: Clustering by partial memory mapping time comparison across different chunk sizes. All experiments were carried out on n1-standard-8 GCP instances with 8 VCPUs.

.

The execution time of the clustering algorithm reduced when chunk size getting bigger. As long as the chunk size remains smaller than the available memory clustering time remains almost a constant no matter the actual size of the memory being used. On the other hand clustering, time increases when the chunk size becomes bigger than the available memory. Based on these observations we decided that the size of the

file chunk should be a variable and it must be chosen based on the size of available memory. To evaluate this idea we conducted the experiment shown in Figure 5.10.

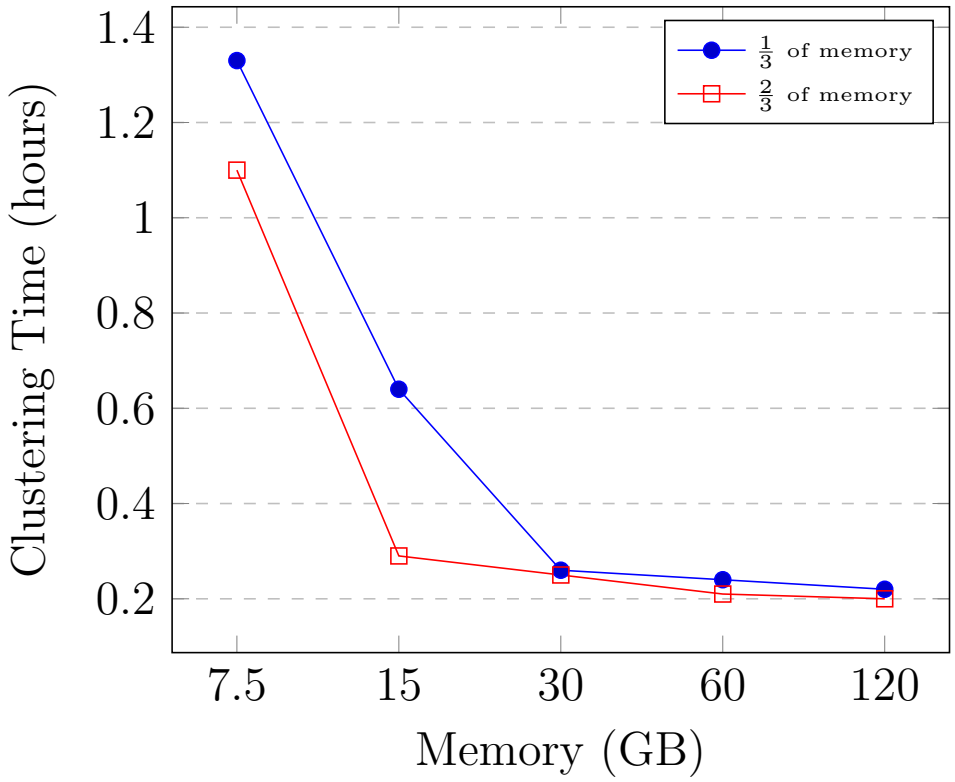


Figure 5.10: Clustering by partial memory mapping time comparison across variable chunk sizes. All experiments were carried out on n1-standard-8 GCP instances with 8 VCPUs.

.

Then we used variable size file chunks mapping in order to better utilise the available memory. As seen on Figure 5.10 this approach provides much better results when compared to fixed file chunks mapping method. Even in low memory instances, the clustering time has reduced with the increased size of file regions mapped to the memory. However, clustering time doesn't reduce in a linear manner when available memory becomes higher. This should be caused by the sequential nature of the algorithm. Even though with the reduced number of I/O algorithm has to process transactions one after the other.

To better utilize the processor we then implemented a parallel version of this algorithm which uses multiple threads to process transaction file portions. We used

Clustering time on n1-standard-8 GCP instances with 8 VCPUs			
Memory: 7.5GB	Memory: 15GB	Memory: 30GB	Memory: 60GB
0.54 hrs	0.19 hrs	0.15 hrs	0.13 hrs

Table 5.4: Clustering time of parallel clustering with file mapping. All experiments were carried out on n1-standard-8 GCP instances with 8 VCPUs. VCPU count remained constant throughout the experiment.

file chunk size as $\frac{2}{3}$ of the available memory. Table 5.4 shows the results we obtained from this approach.

5.2.1 Evaluation of clustering accuracy

We followed the same approach that we discussed in Section 5.1.2. We compared the sequential version of this algorithm against the BlockSci clustering algorithm(sequential) and obtained identical results to 5.1.2. To further prove this claim we plotted the graph(Figure 5.11)of the number of clusters formed at a given time at the execution of clustering algorithms. of clusters formed at a given time, both algorithms with only one worker (sequentially). The reason behind this is that parallel workers may process transaction data nondeterministically, hence cluster formation can be random

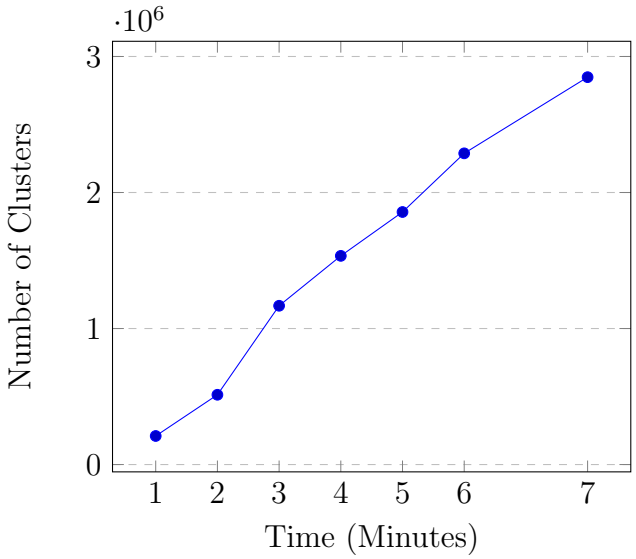


Figure 5.11: Cluster formation at algorithm execution(BlockSci and memory mapping clustering algorithms). **n1-standard-8** GCP instances with 8 VCPUs and 60Gb memeory have been used to execute both BlockSci and memory mapping clustering algorithms.

.

5.3 Comparison of clustering algorithms

Initially, we modified and evaluated BlockSci clustering algorithm in order to find performance problems associated with it. Then we evaluated two different approaches.

1. Using probabilistic data structures to summarize and cache blockchain data to improve Bitcoin address clustering

2. Using memory mapped files and probabilistic data structures to improve Bitcoin address clustering

Figure 5.12 shows a comparison of latter two approaches. There we can observe that parallel version of address clustering with memory mapping has the highest performance in terms of speed and it can be seen as a superior version of the lsutering algorithm that only uses probabilistic data structures.

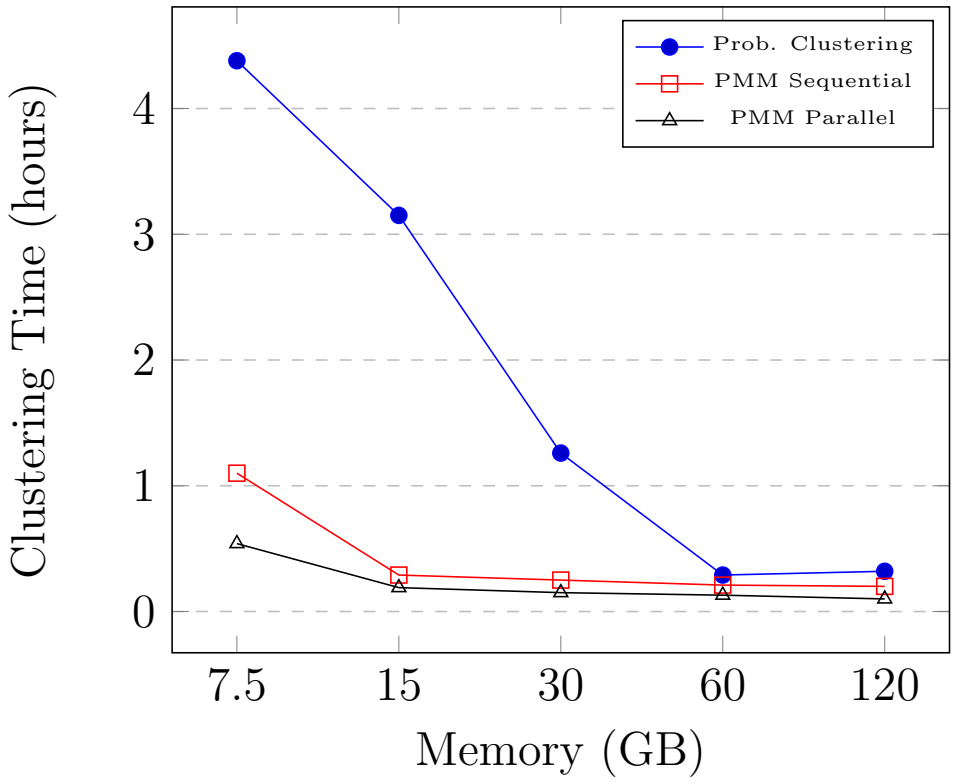


Figure 5.12: Comparison of different clustering approaches. All experiments were carried out on n1-standard-8 GCP instances with 8 VCPUs.

.

Chapter 6

Conclusion

6.1 Introduction

This chapter includes a review of the research aims and objectives, research problem, limitations of the current work and implications for further research.

6.2 Conclusions about research questions

Our primary objective was to design and implement a solution to improve Bitcoin address clustering. In order to achieve this goal we first evaluated existing address clustering approaches used by major Bitcoin analytics platforms such as BitIodine[25], GraphSense[13] and BlockSci[2]. By doing so we observed that the main problem lies within such platforms is that they haven't used the available memory efficiently. Therefore when handling a massive data structure like Bitcoin blockchain there may occur a huge number of unnecessary page faults which in turns results in vast amount of I/O operations which exponentially grows in number when available memory becomes lesser. In order to address this problem we proposed and evaluated two approaches which enable summarizing and caching of blockchain data. The first approach was to use a probabilistic data structure(ShBF) to improve clustering speed. As we illustrated in chapter 5 we were able to reduce the execution time of the clustering algorithm significantly with the use of ShBF[33]. To further improve the clustering speed we used partial file mapping where we mapped parts of the transactions file and processed them individually in order to reduce page faults caused by the algorithm. We implemented both parallel and sequential version of this algorithm and observed that it provides better results than any other approach we used so far. More importantly, we could see a significant improvement in clustering speed even on memory constrained devices(instances with 3.5GB, 7.5GB etc.). By considering the

above facts we can come to a conclusion that is possible to improve the speed of Bitcoin address clustering on memory constrained devices by summarizing and caching blockchain data.

Another objective of this research was to observe whether the scalability of Bitcoin address clustering can be improved without negatively affecting clustering accuracy and speed. We analyzed this problem while designing the memory mapping approach to Bitcoin address clustering. There we used partial file mapping where chunk size that is mapped to the memory can be changed according to the available memory. By doing that we were able to develop an efficient and scalable clustering solution that works efficiently on memory constrained devices as well as high memory devices. Furthermore, in chapter 5 we evaluated that this method is not just scalable, it significantly improves the clustering speed so that our address clustering model outperforms all clustering approaches adopted by major Bitcoin analytics platforms without affecting clustering accuracy.

6.3 Conclusions about research problem

One of the major problems with Bitcoin address clustering is that execution takes days on memory constrained devices. Initially, we analysed clustering approaches adopted by major Bitcoin analytics platforms and observed that none of them provides proper attention to improve the clustering speed and scalability across memory constrained devices. Further evaluation showed us that this poor performance of address clustering has been caused by the I/O overhead occurred at the time of accessing various positions of blockchain randomly. As available memory becomes lesser execution of clustering algorithm takes longer as the number of page faults increase in number. To address this problem we developed two approaches, one which uses probabilistic data structures and the other one uses a combination of probabilistic data structure and partial file mapping. Through the evaluation of these approaches, we showed that they provide a feasible solution to the existing problem we address in our research.

6.4 Limitations

In our study, we didn't concentrate onmeasuring the actual accuracy provided by our model. Instead, we stick to the accuracy model embedded with BlockSci clustering tool. By doing so we determined that the 2 approaches proposed by us have the same

accuracy as BlockSci. The reason behind this was; if we are to calculate the accuracy of clusters we have to follow a lengthy process which involves using different wallets, participation in mining pools, exchanges etc. As our study was primarily about clustering speed improvement those things were out of our context. More importantly, BlockSci is a major Bitcoin analytics platform currently used in various Bitcoin analytics related services[9]. Therefore we can rely on it to provide a reasonable clustering accuracy.

6.5 Implications for further research

The evaluation results showcased when using partial memory mapping in address clustering there are parameters such as the number of hash calculations, size of the bloom filter that we can optimize in order to observe a better clustering performance by reducing FPR. Further research should be carried out in order to determine such optimizations which further improve address clustering speed. Evaluations results showcased that partial memory mapping clustering approach outperforms all other existing clustering approaches adopted by major Bitcoin analytics platforms with variable size file chunks. However, chunk size selection process can be further improved so that chunk size is decided dynamically at the program execution time. It allows utilization of available memory as an efficient caching device in order to further improve clustering performance.

In our study, we only focussed on improving clustering speed. We may conduct further research in order to improve clustering accuracy without negatively affecting the execution time of address clustering algorithm.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf."

[2] H. A. Kalodner, S. Goldfeder, A. Chator, M. Möser, and A. Narayanan, "Blocksci: Design and applications of a blockchain analysis platform," *CoRR*, vol. abs/1709.02489, 2017. [Online]. Available: http://arxiv.org/abs/1709.02489

[3] F. Reid and M. Harrigan, *An Analysis of Anonymity in the Bitcoin System*. New York, NY: Springer New York, 2013, pp. 197–223. [Online]. Available: https://doi.org/10.1007/978-1-4614-4139-7_10

[4] D. Nick, "Master thesis data-driven de-anonymization in bitcoin," 2015.

[5] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. Mccoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: Characterizing payments among men with no names."

[6] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, Sep. 1999. [Online]. Available: http://doi.acm.org/10.1145/331499.331504

[7] D. Ermilov, M. Panov, and Y. Yanovich, "Automatic bitcoin address clustering," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec 2017, pp. 461–466.

[8] M. d. Castillo, "Bitcoin remains most popular digital currency on dark web," https://www.coindesk.com/bitcoin-remains-most-popular-digital-currency-on-dark-web/, 2016, [online; accessed 01-July-2018].

[9] Y. Boshmaf, H. A. Jawaheri, and M. A. Sabah, "Blocktag: Design and applications of a tagging system for blockchain analysis," *CoRR*, vol. abs/1809.06044, 2018.

[10] M. Vasek and T. Moore, "There's no free lunch, even using bitcoin: Tracking the popularity and profits of virtual currency scams," in *International conference on financial cryptography and data security*. Springer, 2015, pp. 44–61.

[11] ——, "Analyzing the bitcoin ponzi scheme ecosystem," in *Financial Cryptography*, 2018.

[12] T. Moore and N. Christin, "Beware the middleman: Empirical analysis of bitcoin-exchange risk," in *International Conference on Financial Cryptography and Data Security*. Springer, 2013, pp. 25–33.

[13] B. Haslhofer, R. Karl, and E. Filtz, "O bitcoin where art thou? insight into large-scale transaction graphs." in *SEMANTiCS (Posters, Demos)*, 2016.

[14] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1773912.1773922

[15] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863103.1863113

[16] N. Nethercote and J. Seward, "Valgrind: A framework for heavyweight dynamic binary instrumentation," in *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '07. New York, NY, USA: ACM, 2007, pp. 89–100. [Online]. Available: http://doi.acm.org/10.1145/1250734.1250746

[17] E. Androulaki, G. O. Karame, M. Roeschlin, and T. Scherer, "Evaluating user privacy in bitcoin."

[18] M. Moser, R. Bohme, and D. Breuker, "An inquiry into money laundering tools in the bitcoin ecosystem," 09 2013, pp. 1–14.

[19] J. Bohr and M. Bashir, "Who uses bitcoin? an exploration of the bitcoin community," in *2014 Twelfth Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2014, pp. 94–101.

[20] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *Computer Security - ESORICS 2014*, M. Kutyłowski and J. Vaidya, Eds. Cham: Springer International Publishing, 2014, pp. 345–364.

[21] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *COMMUNICATIONS OF THE ACM*, vol. 24, pp. 84–88, 1981.

[22] F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," in *Security and privacy in social networks*. Springer, 2013, pp. 197–223.

[23] J. DuPont and A. C. Squicciarini, "Toward de-anonymizing bitcoin by mapping users location," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. ACM, 2015, pp. 139–141.

[24] M. Fleder, M. S. Kester, and S. Pillai, "Bitcoin transaction graph analysis," *arXiv preprint arXiv:1502.01657*, 2015.

[25] M. Spagnuolo, F. Maggi, and S. Zanero, "Bitiodine: Extracting intelligence from the bitcoin network," in *Financial Cryptography*, 2014.

[26] S. Goldfeder, H. A. Kalodner, D. Reisman, and A. Narayanan, "When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies," *CoRR*, vol. abs/1708.04748, 2017. [Online]. Available: http://arxiv.org/abs/1708.04748

[27] M. Bartoletti, S. Lande, L. Pompianu, and A. Bracciali, "A general framework for blockchain analytics," in *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, ser. SERIAL '17. New York, NY, USA: ACM, 2017, pp. 7:1–7:6. [Online]. Available: http://doi.acm.org/10.1145/3152824.3152831

[28] G. Cybenko, T. G. Allen, and J. E. Polito, "Practical parallel union-find algorithms for transitive closure and clustering," *International Journal of Parallel Programming*, vol. 17, no. 5, pp. 403–423, Oct 1988. [Online]. Available: https://doi.org/10.1007/BF01383882

[29] R. J. Anderson and H. Woll, "Wait-free parallel algorithms for the union-find problem," in *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, ser. STOC '91. New York, NY, USA: ACM, 1991, pp. 370–380. [Online]. Available: http://doi.acm.org/10.1145/103418.103458

[30] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '14. New York, NY, USA: ACM, 2014, pp. 75–88. [Online]. Available: http://doi.acm.org/10.1145/2674005.2674994

[31] A. Kirsch and M. Mitzenmacher, "Less hashing, same performance: Building a better bloom filter," *Random Struct. Algorithms*, vol. 33, no. 2, pp. 187–218, Sep. 2008. [Online]. Available: http://dx.doi.org/10.1002/rsa.v33:2

[32] Y. Qiao, T. Li, and S. Chen, "One memory access bloom filters and their generalization," in *2011 Proceedings IEEE INFOCOM*, April 2011, pp. 1745–1753.

[33] T. Yang, A. X. Liu, M. Shahzad, Y. Zhong, Q. Fu, Z. Li, G. Xie, and X. Li, "A shifting bloom filter framework for set queries," *Proc. VLDB Endow.*, vol. 9, no. 5, pp. 408–419, Jan. 2016. [Online]. Available: http://dx.doi.org/10.14778/2876473.2876476

[34] V. K. Vaishnavi and W. L. Kuechler, "Design Science Research in Information Systems," *Ais*, pp. 1–45, 2004. [Online]. Available: http://www.desrist.org/design-research-in-information-systems/

[35] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970. [Online]. Available: http://doi.acm.org/10.1145/362686.362692

[36] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, Jun. 2000. [Online]. Available: http://dx.doi.org/10.1109/90.851975

[37] S. Cohen and Y. Matias, "Spectral bloom filters," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '03. New York, NY, USA: ACM, 2003, pp. 241–252. [Online]. Available: http://doi.acm.org/10.1145/872757.872787

[38] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.jalgor.2003.12.001

[39] S. Tirumal rao, E. V Prasad, and N. Bala Venkateswarlu, "Performance evaluation of memory mapped files with data mining algorithms," *International Journal of Information Technology and Knowledge Management*, vol. 2, pp. 365–370, 08 0002.

Appendix A

Diagrams

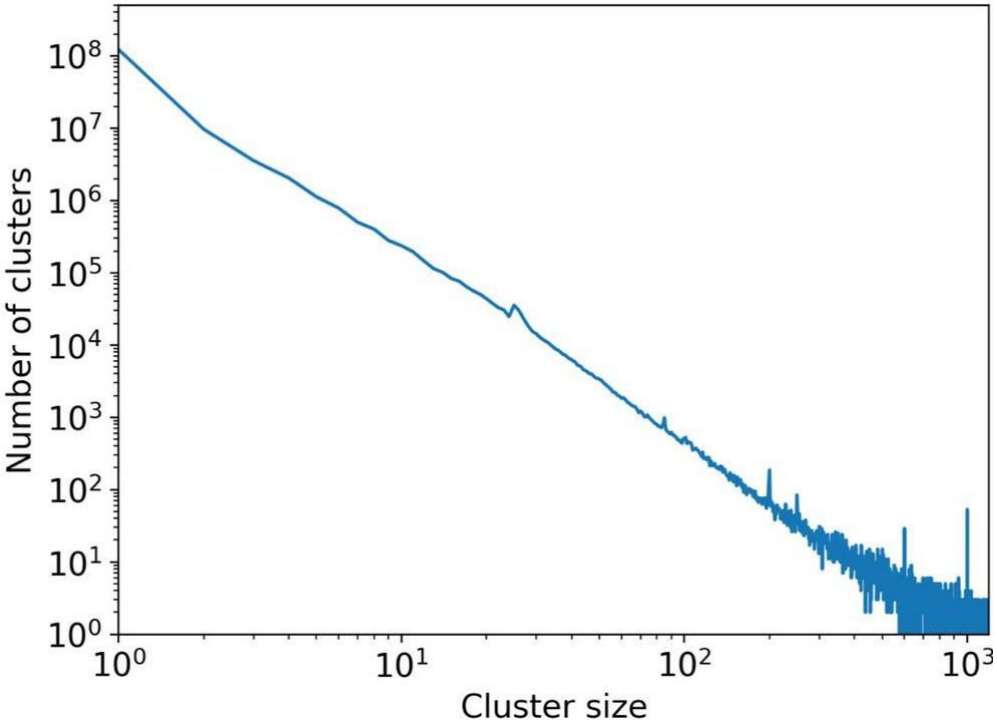


Figure A.1: Distribution of sizes of address clusters in Bitcoin after applying address linking heuristics. Size 1-2000 are shown here but there are many clusters that are much larger[2]

Appendix B

Code Listings

Listing B.1: Constructor for ShBF

```
1  ShBF* new_ShBF_M(int m, int n) {
2      ShBF* shbf;
3      int k;
4      int k_floor;
5      float k_unrounded;
6      int B_size;
7      Size shbf_size;
8
9      k_unrounded = K_OPT_SHBF_M * ((float)m / n) / 2;
10     k_floor = (int)k_unrounded;
11     k = ((k_unrounded - k_floor) >= 0.5) ? (k_floor + 1) : (
           ↪ k_floor);
12     if (k == 0) { k = 1; }
13     B_size = m + (W - 1);
14     B_size = ((B_size % 8) == 0) ? (B_size / 8) : ((B_size /
           ↪ 8) + 1);
15     shbf_size = B_size + sizeof(ShBF);
16
17     shbf = palloc0(shbf_size);
18     shbf->B_length = B_size;
19     shbf->m = m;
20     shbf->n = n;
21     shbf->k = k;
22     shbf->w = W;
23
24     SET_VARSIZE(shbf, shbf_size);
25
26     return shbf;
27 }
```

Listing B.2: Inserts an element into ShBF

```
1  void insert_ShBF_M(ShBF* shbf_m, char* e) {
2      uint64_t hva[2] = {0, 0};
3      uint64_t offset_hash;
4      uint64_t i_hash;
5      int offset_value;
6      int first_index;
7      int second_index;
8      int i;
9
10     MurmurHash3_x64_128(e, strlen(e), MURMUR_HASH_SEED, &hva)
           ↪ ;
11
12     offset_hash = hva[0];
13     offset_value = (offset_hash % (shbf_m->w - 1)) + 1;
14
15     for (i = 1; i <= shbf_m->k; i++) {
16
17         i_hash = hva[0] + i*hva[1];
18
19         first_index = i_hash % shbf_m->m;
20         second_index = first_index + offset_value;
21
22         set_bit_ShBF(shbf_m, first_index);
23         set_bit_ShBF(shbf_m, second_index);
24     }
25 }
```

Listing B.3: Queries ShBF to determine if a given element is present

```
1  int query_ShBF_M(ShBF* shbf_m, char* e) {
2      uint64_t hva[2] = {0, 0};
3      uint64_t offset_hash;
4      uint64_t i_hash;
5      int offset_value;
6      int first_index;
7      int second_index;
8      int i;
9
10     MurmurHash3_x64_128(e, strlen(e), MURMUR_HASH_SEED, &hva)
           ↪ ;
11
12     offset_hash = hva[0];
13     offset_value = (offset_hash % (shbf_m->w - 1)) + 1;
14
```

```
15      for (i = 1; i <= shbf_m->k; i++) {
16
17          i_hash = hva[0] + i*hva[1];
18
19          first_index = i_hash % shbf_m->m;
20          second_index = first_index + offset_value;
21
22          if (get_bit_ShBF(shbf_m, first_index) != 1 ||
23              get_bit_ShBF(shbf_m, second_index) != 1) {
24
25              return 0;
26          }
27      }
28
29      return 1;
30  }
```