PARALLEL ANALYSIS OF BLOCKCHAIN TRANSACTION GRAPHS

by

Baran Kılıç

B.S., Computer Engineering, Boğaziçi University, 2019

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2022

ACKNOWLEDGEMENTS

The research and development work presented in this thesis was carried out as part of the H2020 Infinitech Project and has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No. 856632.

ABSTRACT

PARALLEL ANALYSIS OF BLOCKCHAIN TRANSACTION GRAPHS

Blockchain is more prominent in the finance sector than ever. Stablecoins build a bridge between traditional finance ecosystems and the blockchain ecosystem. Major payment processors adopt cryptocurrency solutions and integrate them into their systems. Blockchain transaction analysis is needed to enforce cryptocurrency regulations, trace fraudulent activities, and create business intelligence solutions. Transaction throughput of blockchains is expected to rise with the transition to proof-of-stake (PoS) consensus mechanism, sharding, and the use of zero-knowledge proofs. New tooling is needed to handle massive transaction graphs. In this thesis, we propose a parallel blockchain transaction graph system for analyzing blockchain transaction graphs. The system utilizes distributed data structures and graph algorithms and is implemented in C++ using message passing interface (MPI). The system constructs the transaction graph from blockchain data using our proposed parallel graph construction algorithm. The transaction graph is then analyzed using our distributed and parallel transaction trace and trace forest algorithms. In addition, we implemented PageRank, connected component calculation, degree distribution calculation algorithms. We collected 12-year Bitcoin and 5-year Ethereum blockchain transaction data as well as some blacklisted blockchain addresses from various websites to test our system. The system is benchmarked using a 16-node high performance computing (HPC) cluster on Amazon Cloud. We report timings obtained for our tests and analysis results like top 10 pageranked addresses, the degree distribution of addresses, trace visualizations. We were able to construct the transactions graph for our Ethereum and Bitcoin transaction data on our cluster in less than 4 minutes and 32 minutes, respectively.

ÖZET

BLOKZİNCİR İŞLEM ÇİZGELERİNİN PARALEL ANALİZİ

Blokzincirinin finans sektöründeki önemi giderek artıyor. Sabitkoinler geleneksel finans ekosistemleri ile blokzincir ekosistemi arasında bir köprü oluşturuyor. Büyük ödeme işlemcileri kripto para çözümlerini benimseyip bu çözümleri sistemlerine entegre ediyor. Blokzincir işlem analizi kripto para düzenlemelerini uygulamak, dolandırıcılık faaliyetlerini izlemek ve iş zekası çözümleri oluşturmak için gereklidir. Blokzincirlerinin işlem hacminin, hisse ispatı (PoS) konsensüs mekanizmasına geçiş ve sıfır bilgi ispatlarının kullanımı ile artması bekleniyor. Büyük işlem çizgelerini işlemek için yeni araçlara ihtiyaç var. Bu tezde, blokzincir işlem çizgelerinin analizini yapmak için paralel bir blokzincir işlem çizge sistemi öneriyoruz. Sistem dağıtık veri yapılarını ve dağıtık çizge algoritmalarını kullanıyor ve mesaj aktarma arayüzü (MPI) kullanılarak C++ programlama dilinde yazıldı. Sistem, önerilen paralel çizge oluşturma algoritmamızı kullanarak blokzincir verilerinden işlem çizgesini oluşturur. İşlem çizgesi daha sonra dağıtık ve paralel işlem izleme ve izleme ormanı algoritmalarımız kullanılarak çözümlenir. Ayrıca PageRank, bağlantılı bileşen hesaplama, derece dağılımı hesaplama algoritmalarını da kodlayıp sistemimize ekledik. Sistemimizi test etmek için 12 yıllık Bitcoin ve 5 yıllık Ethereum blokzinciri işlem verilerini ve çeşitli web sitelerinden bazı kara listeye alınmış blokzincir adreslerini topladık. Sistem, Amazon Bulut üzerinde 16 düğümlü yüksek başarımlı hesaplama (HPC) kümesi kullanılarak değerlendirildi. Testlerimiz için elde edilen zamanlamaları ve en iyi 10 pagerank adresi, adreslerin derece dağılımı, izleme görselleştirmeleri gibi analiz sonuçlarını raporladık. Kümemizde Ethereum ve Bitcoin işlem verilerimiz için işlem çizgesini sırasıyla 4 dakika ve 32 dakikadan daha kısa sürede oluşturabildik.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

A^p	Set of addresses to be traversed on process p
B	Set of blacklisted addresses
C	Total number of parent node and depth pairs to be sent
C^p	Set of unvisited remote addresses on process p
D^p	Distance of addresses on process p from blacklisted address
e	A transaction $e \in E$ and $e = (s, t, b)$ where s is the sender (tail of edge), t is the recipient (head of edge), and b is the block number of transaction
E	All transfer transactions. $E = E_{\text{ETH}} \cup E_t$ for Ethereum or $E = E_{\text{BTC}}$ for Bitcoin
E'^p	Transfer transactions of trace subgraph on process p
E^p	Transfer transactions on process p
E_{BTC}	Transfers between accounts and transaction IDs. $E_{\text{BTC}} = \{<a, t>: a \in V_a \wedge t \in V_t\} \cup \{<t, a>: t \in V_t \wedge a \in V_a\}$
E_{ETH}	All Ethereum blockchain transactions with zero or more ether payments
E_t	ERC20 token t transfer transactions, $t \in T$
F	Set of starting addresses for graph traversal
F^p	Trace forest on process p
$G(V, E)$	Transaction graph
$G^p(V^p, E^p)$	Transaction subgraph on process p
$ID(v)$	Global ID of address v, $ID(v) \in [0, \|V\| - 1]$
M^p	Set of remote addresses that are already visited on process p
p	ID of current process (0-indexed)
P	Number of processes
Q	Set of queried addresses
R^p	Set of parent node and depth pairs received from other processes (Subsection 5.4.1)

R^p	Set of addresses reachable from F on process p (Subsection 5.4.2)
Rev	Traversal direction. Outcoming edges if true. Incoming edges if false.
S^p	Set of parent node and depth pairs on process p to be sent to remote processes (Subsection 5.4.1)
S^p	Set of addresses to be visited on process p (Subsection 5.4.2)
T	Set of major ERC20 tokens tracked, $T = \{\text{USDT,PAX,TRYB}, \ldots\}$, full list given in Table 4.1
T_e	Block range start
T_s	Block range end
TCN	Total number of remote addresses to be visited in all processes
V	All blockchain addresses. $V = V_c \cup V_s$ for Ethereum or $V = V_a \cup V_t$ for Bitcoin
V'	Set of addresses of trace subgraph
V'^p	Set of addresses of trace subgraph on process p
V^p	Blockchain addresses on process p
V_a	Set of Bitcoin addresses
V_c	Set of Ethereum externally owned account addresses
V_s	Set of Ethereum smart contract addresses
V_t	Set of transaction IDs (TXIDs)

LIST OF ACRONYMS/ABBREVIATIONS

AML	Anti Money Laundering
API	Application Programming Interface
CBDC	Central Bank Digital Currency
CFT	Counter Financing of Terrorism
CPU	Central Processing Unit
CSR	Compressed Sparse Row
EBS	Elastic Block Store
ERC	Ethereum Request For Comments
FATF	Financial Action Task Force
GB	Gigabyte
Gbps	Gigabits per second
GiB	Gibibyte
HPC	High Performance Computing
HTML	HyperText Markup Language
ID	Identity
JSON	JavaScript Object Notation
KYC	Know Your Customer
Mbps	Megabits per second
MPI	Message Passing Interface
NFS	Network File System
OFAC	Office of Foreign Assets Control
PoS	Proof of Stake
PoW	Proof of Work
RLP	Recursive Length Prefix
RPC	Remote Procedure Call
SDN	Specially Designated Nationals
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol

TXID	Transaction Identity
UTXO	Unspent Transaction Output
VASP	Virtual Asset Service Provider

1. INTRODUCTION

Blockchain is a type of distributed ledger technology where the records of transactions are stored as an immutable sequence of blocks. Building on cryptographic technologies, blockchains can provide programmable transaction services directly to the masses in a trustless, speedy, and low-cost manner by removing middleman organizations that operate in a classical centralized fashion. Blockchains also keep ownership records of cryptocurrencies and tokens representing various assets such as company shares, stablecoins (tokenized forms of fiat currencies), media, and works of art.

Blockchain has numerous applications in various sectors. It can be used to monitor supply chains, securely share medical data, automate insurance operations, collect royalties, verify official documents, create digital identities, create decentralized finance services, manage the distribution and trade of energy, and many more. Although blockchain has use cases in many fields, it is commonly known for cryptocurrencies because it makes it easy to transfer and trade crypto-assets worldwide.

Blockchain is more prominent in the finance sector than ever. Here is some recent news about blockchain adoption in the finance sector. Large payment networks such as Mastercard [3], Visa [4], PayPal [5] announce that they are now accepting cryptocurrencies as a payment method. Mastercard and Bakkt partner to offer loyalty rewards in cryptocurrencies [6]. El Salvador becomes the first country to adopt Bitcoin as national currency [7]. Countries around the world are researching or piloting central bank digital currencies (CBDCs). Some even launched it. Visa is researching to make stablecoins and CBDCs interoperable [8]. Coinbase, the largest cryptocurrency exchange in the United States, is directly listed in the Nasdaq stock exchange [9], allowing cryptocurrencies to gain exposure and legality.

New technologies require new tooling. When the Internet first appeared, there was a need for web page search services. This need has led to the development of search engine companies. There is a similar need in the case of blockchain networks, and that is the need to get information on addresses, tokens, and transactions on the blockchain. This need has led to the development of block explorer services offered by companies like Etherscan, which is currently the most famous explorer service for Ethereum. Block explorer services provide basic information on individual addresses, tokens, and transactions such as amounts, balances, and times of transactions. Blockchain transactions, however, form a directed graph, and there is a need to analyze this graph.

ERC20 token contracts act like bridges between the public Ethereum blockchain ecosystem and the traditional finance ecosystems. Valuable blockchain assets such as cryptocurrencies can be exchanged with stablecoins, and stablecoins can, in turn, be redeemed as fiat money in the traditional finance ecosystem. Therefore, assets obtained fraudulently can go through various transfers on the blockchain and end up as stable coins in different jurisdictions. As a result, a company that accepts stablecoins may be paid by stablecoins that can be traced to addresses involved in fraudulent activities. Holding stable coins that originated from fraudulent or sanctioned addresses can be risky for the company.

Blockchain transaction analysis is necessary to enforce cryptocurrency regulations, trace fraudulent activities like ransomware, get provenance information about tokens representing products, and create business intelligence solutions. Financial Action Task Force (FATF) has published guidance [10] in which it requires that *"Virtual Asset Service Providers (VASPs) be regulated for anti-money laundering and combating the financing of terrorism (AML/CFT) purposes, licensed or registered, and subject to effective systems for monitoring or supervision."* The Office of Foreign Assets Control (OFAC) in the United States compiles a specially designated nationals (SDN) list [11] and prohibits U.S. citizens from dealing with them. This list also contains cryptocurrency addresses. Recently, fraud-related activities such as hacks, scams, and ransomware have increased. For example, in the recent Colonial Pipeline ransomware incident [12, 13], hackers demanded and got paid approximately 75 bitcoins after taking part of the company's infrastructure out of operation. In 2021, Poly Network [14], DAO Maker [15], Liquid Exchange [16], Coinbase [17] and pNetwork [18] are hacked. These developments are necessitating the development of tools and services that will help to analyze transactions on blockchains.

A blockchain transaction analysis system has to be fast, parallel, and scalable. The system must be fast to take swift action against fraudulent activities while the new transactions are executed. It also needs to be parallel and scalable to handle the ever-increasing blockchain transaction data. Such a system can simply be scaled out by using a larger number of cluster nodes. In addition to the increasing numbers of transactions, the blockchain transaction throughput is expected to rise. The current Proof-of-Work (PoW) based Bitcoin and Ethereum blockchains have very low transaction throughput, as shown in Table 1.1. Newer Proof-of-Stake (PoS) based systems like Ethereum2, Avalanche, and Cardano will be able to achieve thousands of transactions per second (tps). Sharding and zero-knowledge proofs will also increase the transaction throughput. Permissioned Hyperledger, which is designed for enterprises, can also achieve thousands of tps. Such a high transaction throughput will enable the transaction graphs to grow to billions and billions in size.

Table 1.1. Transaction throughputs.

Blockchain	**Type**	**Transaction Throughput (tps)**
Bitcoin	permissionless, PoW	7
Ethereum	permissionless, PoW	14-30
Hyperledger Fabric	permissioned	3.5K [19]
Ethereum2	permissionless, PoS	first 2-3K, then 100K [20]
Cardano	permissionless, PoS	1K per stake pool with Hydra [21]
Avalanche	permissionless, PoS	$>$ 4500 [22]

The current tools are not fit to handle or access large amounts of transaction data. Blockchain data is only accessible in sequential form. Each block or transaction can be retrieved from blockchain data using the blockchain client's application programming interface (API). The blockchain data can be extracted and stored in a relational database, but making complex queries to such a database will be slow. By complex queries, we mean the queries like tracing an address back to a blacklisted address or calculating the Pagerank of an address. In addition, graph algorithms cannot run on relational databases. Graph databases like Neo4j can be used instead. However, they are not distributed and only replicate data when run on multiple nodes. Therefore, they will not be able to handle increasing amounts of transactions.

Some of the materials contained in this thesis were previously submitted to “Second International Conference on Blockchain Computing and Applications” and accepted as a conference paper [23]. An extended version of this conference paper is published in the “Cluster Computing” journal [24]. Some of the materials will be published in the “Big Data and Artificial Intelligence in Digital Finance” book as a chapter [25].

1.1. Contributions of the Thesis

The main contributions of this thesis are:

(i) Extract Bitcoin, Ethereum, and popular ERC20 [26] token transaction information,
(ii) Compile a list of blacklisted addresses for Bitcoin and Ethereum blockchains,
(iii) Propose a parallel graph system for blockchain transactions using message passing interface (MPI [27]),
(iv) Develop a parallel scalable graph construction algorithm,
(v) Implement parallel graph traversal for transaction tracing,
(vi) Integrate an existing parallel graph partitioner into the system,
(vii) Analyze transaction graphs using graph traversal for fraudulent activities,
(viii) Benchmark the parallel system with different number of processors and nodes.

1.2. Outline of the Thesis

The rest of this thesis is organized as follows. Chapter 2 gives a background on blockchain and parallel programming to help the readers understand the following chapters of the thesis. Chapter 3 reviews the previous works about Bitcoin and Ethereum transaction graph analysis. Chapter 4 explains the details of the blockchain and blacklist data collection processes. Chapter 5 presents our proposed blockchain transaction graph system and explains its architecture, data structures, and algorithms. Chapter 6 covers the details of the test environment, the benchmark timing results, and the results obtained from the transaction graph analysis. Finally, Chapter 7 concludes the thesis and discusses future work.

2. BACKGROUND

2.1. Blockchain

2.1.1. Token Models

The two common models for tokens in blockchains are unspent transaction output (UTXO) and account models. Bitcoin uses the UTXO model, and Ethereum uses the account model. In the UTXO model, each transaction takes UTXOs as input, consumes them, and outputs new UTXOs. An UTXO is indivisible. For example, one cannot spend 0.4 value from an UTXO with 1 value. However, if you want to spend 0.4 value, you can use the UTXO with 1 value and create two UTXOs with 0.4 and 0.6 values, the UTXO with 0.6 value is returned to you and the other one is sent to the recipient of the transaction. A user's balance is the sum of all UTXOs a user own. Each account has a balance in the account model, increasing or decreasing with transactions. Our system that is presented in Chapter 5 supports both types of models.

2.1.2. Token Types

Ethereum defines several standards to create a common interface between tokens. ERC20 token standard [26] introduces fungible tokens. ERC721 [28] defines a standard for non-fungible tokens. ERC1155 [29] is the standard for multi tokens (both fungible and non-fungible).

In a ERC20 contract, a user can check his balance of tokens, transfer his tokens to another user, allow a specific user to withdraw tokens from his account up to a specified amount (allowence amount), check this allowence amount, and transfer the tokens of a user (if that user allowed him) to another user.

2.2. Parallel Programming

2.2.1. MPI

In message passing interface [27] programs, the copy of the same program is run on multiple processors, and these copies communicate with each other via message passing. The copies of the program are called MPI processes. We will refer to an MPI process simply as a process from now on. Each process has an identifier (ID) called a rank. The ranks are zero-indexed, i.e., they start from zero. MPI employs a distributed memory model (excluding the commands introduced after MPI-1). Therefore, the value of a variable on each process can be different. To refer to the value of a variable on a specific process, we will use a superscript notation. For example, if A is a variable, A^2 is the variable's value on the process with rank 2. This thesis uses the words *local*, *remote*, and *global* to refer to different parts of distributed data structures. When we use the word *local*, we refer to the part of data on the current process. When we use the word *remote*, we refer to the part of data on the other processes. When we use the word *global*, we refer to all of the data as a whole on all processes. The MPI programs are usually run on a cluster of nodes, and a single process is run on each core of a node in a cluster.

2.2.2. Distributed Data Structures

Distributed data structures are utilized when implementing MPI programs. The simplest example is a distributed array. The structure of an example distributed array can be seen in Figure 2.1. An array is divided into multiple parts, and each part is stored on a different process. In this example, there are three processes, and processes 0, 1, and 2 store 3, 2, and 3 elements, respectively. The first element of the process with ID 1 has the value 2. Its index is 0 if you only consider the local part of the array. Its index is 3 if you consider all of the arrays. The conversion between local and global index can be done by storing a value for the global index of the first value in the local array, which is presented as the start index in the example. You get the global index when you add the local index with the start index.

Process ID	0			1		2		
Start index	0			3		5		
Global index	0	1	2	3	4	5	6	7
Local index	0	1	2	0	1	0	1	2
Array values	3	7	4	2	3	8	4	4

Figure 2.1. How a distributed array is stored on three processes.

Distributed compressed sparse row (CSR) format is another distributed data structure. Figure 2.2 illustrates the serial CSR format. Let A be the adjacency matrix of a directed graph, where the rows stand for the tail nodes, the columns stand for the head nodes, and the cell value stands for the weight of the edge. The arrays row_begin, values, and col_indices represent the CSR format. The values array contains non-zero entries in the matrix in row-wise order. The array col_indices gives the corresponding column indices of these values. The array row_begin stores the beginning index of each matrix row in the values. Distributed CSR can be thought of as a combination of the serial CSR and the distributed array concepts.

$$A = \begin{bmatrix} 4 & 0 & 3 \\ 0 & 0 & 8 \\ 7 & 3 & 0 \end{bmatrix} \qquad \begin{aligned} \text{row\_begin} &= \begin{bmatrix} 0 & 2 & 3 & 5 \end{bmatrix} \\ \text{values} &= \begin{bmatrix} 4 & 3 & 8 & 7 & 3 \end{bmatrix} \\ \text{col\_indices} &= \begin{bmatrix} 0 & 2 & 2 & 0 & 1 \end{bmatrix} \end{aligned}$$

Figure 2.2. Compressed sparse row matrix format.

3. LITERATURE REVIEW

3.1. Bitcoin Transaction Graph Analysis

The work [30] is an early analysis of Bitcoin from the privacy perspective. They extracted Bitcoin transactions from January 2009 to July 2011 and analyzed the user and transaction networks. They reported degree distribution, cumulative component size, edge count, density, and average path length for both transaction and user networks. They show that the identity of some users may be disclosed by collecting information from Bitcoin Faucet, voluntary disclosures, TCP/IP layer information.

In the work [31], Bitcoin blockchain transactions from January 2009 to May 2012 are analyzed. They downloaded the first 180 000 blocks as HTML files from a bitcoin explorer. Since Bitcoin encourages its users to create a new address to receive payments, one user may have multiple addresses. This paper merges the addresses using the Union-Find algorithm [32] and calls it an entity. They report various statistics about Bitcoin, such as the distribution of the addresses per entity, the distribution of the current balance, maximum balance ever seen, the distribution of transactions. They analyzed the subgraph of one large transaction more closely.

BlockSci [33] proposes a blockchain analysis platform for Bitcoin. This platform uses an analytical in-memory database and works on a single node. The source code of this platform is available at [34].

3.2. Ethereum Transaction Graph Analysis

The work reported in [35] presents the statistical properties of the Ethereum transaction graph. They analyzed more than 680 000 transactions from the blocks between 200 000 and 300 000, representing an early stage of Ethereum, and 610 000 transactions extracted from blocks 3 000 000 and 3 200 000, representing a later state of Ethereum. They show that some statistics, such as transaction volume and degree distribution, are heavy-tailed, and the transaction graph has a bow-tie structure.

In the work [36], a part of Ethereum transactions is loaded into the Neo4j graph database and analyzed. They found addresses related to transactions from the Gatecoin hack. They downloaded transaction information from an Ethereum blockchain explorer.

The work presented in [37] analyzes Ethereum transactions between July 2015 and August 2017. They analyze three types of transactions: user-to-user, user-to-smart contract, and smart contract deployment. They report the number of transactions and transferred ether amount over time. The statistics about indegree, outdegree, betweenness centrality, left eigenvector centrality of transaction graph are presented.

Chen et al. [38] analyze Ethereum transactions between July 2015 and October 2018. They construct money flow, smart contract creation, and smart contract invocation graphs. The degree distribution, clustering coefficient, degree correlation, node importance, assortativity coefficient, and Pearson coefficient metrics are calculated for the three graphs in their work. They also analyze the evolution of the graphs. They propose new approaches for attack forensics, anomaly detection, and deanonymization.

Victor and Lüders [39] analyze the top 1000 ERC20 token transactions of the first 6.3 million Ethereum blocks. They provide information about degree distribution, clustering coefficients, degree assortativity, and network activity of token transaction networks. For 64000 ERC20 tokens, they provide only an overview and show that the networks follow either a star or hub-spoke pattern.

Somin et al. [40] analyze ERC20 token transactions between February 2016 and February 2018. They collected 18517 token addresses. They report buying popularity, selling popularity of tokens. They report indegree, outdegree of ERC20 token transaction graph. They show that the degrees of nodes in the graph follow a power-law distribution. In a more recent work [41], they add 4 more months of ERC20 token transactions to their data set and show that ERC20 token transaction graph development over time can be modeled as an underdamped harmonic oscillator.

XBlock-ETH [42] extracts the first 8.1 million blocks of Ethereum. They prepare six data sets from the extracted raw data: block and transaction, internal ether transaction, contract information, contract calls, ERC20 token transactions, and ERC721 token transactions. The data sets are available at [43]. The data sets are updated to the first 9 million blocks of Ethereum as of writing this thesis.

4. DATA COLLECTION

We collected the transaction data and blacklisted addresses of Bitcoin and Ethereum blockchains to test our system. We choose Bitcoin and Ethereum because they are the most popular two blockchains and have the largest market capitalization [44] as of writing this thesis. In addition, Bitcoin and Ethereum are the most prominent representatives of the two important blockchain categories, namely blockchain 1.0 and 2.0. Bitcoin is the first blockchain and started the blockchain 1.0 phase. The introduction of smart contracts started blockchain 2.0, and Ethereum is one of the most important blockchains in this category. We also collected the transaction data of major tokens and stablecoins that are implemented as ERC20 tokens. The tokens are listed in Table 4.1.

Table 4.1. List of symbols of 40 major ERC20 tokens.

BNB	DAI	HPT	LEO	OMG	renBTC	SUSD	VEN
BTCB	EURS	HT	LINK	PAX	REP	SXP	WBTC
BUSD	HDG	imBTC	MKR	pBTC	SAI	TRYb	XAUt
CRO	HEDG	INB	MOF	PLUS	sBTC	USDC	XIN
cWBTC	HOT	INO	OKB	QCAD	SNX	USDT	ZRX

4.1. Blockchain Data Collection

The blockchain transaction data can be collected in various ways, such as syncing a node, accessing a node, and using already extracted data. Each of the listed methods has different advantages and disadvantages.

Syncing a node can be considered the most official way of collecting blockchain data. A full node has to be run and synced to download and verify the blockchain blocks. This method requires high bandwidth and large disk space, and the syncing process can take days if not weeks. For example, the Go Ethereum client needs more than six terabytes of disk size for the full node archive sync of the Ethereum mainnet [45]. Depending on the blockchain and client, this method may also require a high processing power because the downloaded block must be verified. The advantage of using this method over the other methods is that you can get the transaction trace data. For example, you can run the methods prefixed with debug (e.g. debug_traceTransaction) in Go Ethereum client.

The second method for collecting blockchain data is accessing a node. For example, both Infura [46] and Cloudflare [47] provide a gateway for Ethereum. In addition, another website [48] provides access to many other blockchains including Bitcoin, and Ethereum. The advantage of such a gateway is that Ethereum can be interacted with using the JSON RPC API without syncing a node. This method may require you to pay for the gateway services. At the time of writing this thesis, the Cloudflare Ethereum gateway was free, and the Infura Ethereum gateway has a free tier for a limited number of requests per day. The disadvantage of this method is that you need to pay extra for accessing trace data, or the service may not even provide the trace data. The trace data is only necessary to examine the internal Ethereum transactions.

A third method for collecting blockchain data is using the already extracted data. This method requires you to trust the provider of the data. There are various extracted blockchain data that are available on the internet [43,49,50]. The most comprehensive data is provided by Google BigQuery [49]. The tools that Google uses to extract the blockchain data are available online [51].

The Ethereum data used in this thesis is collected from the Cloudflare Ethereum gateway. We choose this method because it is easier than syncing a node. The dataset we created for Ethereum is available at [52]. The Bitcoin data used in this thesis is collected from Google BigQuery. We chose this method because there was no Bitcoin gateway at the time we collected the data and syncing a Bitcoin node and extracting the Bitcoin transaction data from a synced node is a time-consuming process, especially finding the address from an input UTXO because you need to go back to the block that the UTXO is output for each UTXO input to a transaction. How the Bitcoin and Ethereum data are collected is explained in more detail in Subsection 4.1.2 and 4.1.3, respectively. The statistics of the collected data are given in Table 6.1.

4.1.1. Transfer Transaction Schema

The extracted blockchain and ERC20 token transfer transactions are stored in plain text files in a space-separated values format. The blockchain transfer transaction schema used in this thesis is presented in Table 4.2. The sender, recipient, and amount fields are the required fields to represent a transfer transaction. The sender is the sender address of the transaction (called *from* in Ethereum). The recipient is the recipient of the transaction (called *to* in Ethereum). The amount is the amount of the asset that is transferred. It is stored in hexadecimal form for Ethereum. In addition, there is a need to store the type of the transaction since there are different ERC20 tokens. We used the symbol field for this purpose. The symbols for the assets are taken from the symbol field of the ERC20 token smart contracts. The block number and transaction index fields are used to be able to order the transactions. In addition, they uniquely identify a transaction, and you can find the transaction in a blockchain explorer using this information. The transaction index represents the index of transactions inside the block.

Table 4.2. Transaction schema for Ethereum transactions.

Symbol	**Block No.**	**Tx. Id.**	**Sender**	**Recipient**	**Amount**
ETH	9400000	2	0xeea...bfb	0x179...d72	0x5a0257dae3c19f
USDT	9400000	5	0x8f2...11d	0x6b2...1a6f	0x82a7440

Tx. Id.: transaction index inside the block

In Bitcoin, a transaction has multiple senders (input UTXOs) and recipients (output UTXOs). To convert the Bitcoin transactions to the format represented in

Table 4.2, we treated the transaction IDs (TXID) in Bitcoin as addresses. We created a transaction from input address to TXID with the symbol BTC-IN for each input, and a transaction from TXID to output address with the symbol BTC-OUT for each output. An example of the output of this conversion for a transaction [53] is presented in Table 4.3. The Bitcoin transaction amounts are stored in decimal form.

Table 4.3. Transaction schema for Bitcoin transactions.

Symbol	**Block No.**	**Tx. Id.**	**Sender**	**Recipient**	**Amount**
BTC-IN	618000	1	396...piq	26f...0a3	29983720
BTC-OUT	618000	1	26f...0a3	3Qw...2zi	26956164
BTC-OUT	618000	1	26f...0a3	3DG...qGx	3000000

Tx. Id.: transaction index inside the block

4.1.2. Bitcoin Transaction Data Collection

The Bitcoin transaction data can be collected using *getblockhash*, *getblock*, and *getrawtransaction* functions from the Bitcoin RPC API [54]. *getblockhash* queries the block hash with the block number. The block information can be obtained with this block hash using *getblock*. If you use *verbosity* = 2 with *getblock*, transaction information is embedded into the block. When you use *verbosity* = 1, you can get each transaction by calling *getrawtransaction* with transaction id listed in the *tx* field of the block. The UTXOs can be found under *vout* in the transaction data. The *value* field in each UTXO represents the transaction amount. The address of the UTXO can be found under *scriptPubKey.addresses[0]*.

We used Google BigQuery to collect Bitcoin data. The SQL query we used is available in Figure A.1. The result of the query is exported as compressed JSON files. After downloading the files, they are parsed and converted to the format described in Subsection 4.1.1. We sorted the transactions by block number since the SQL query result returned is unordered. The Bitcoin data in Google BigQuery does not contain the index of transactions in blocks. We had to extract this information from Bitcoin RPC API and add it to our transaction files.

4.1.3. Ethereum Transaction Data Collection

4.1.3.1. Ether Transactions. The block data for Ethereum is downloaded from Cloudflare Ethereum gateway [47] using the *eth_getBlockByNumber* JSON-RPC function [55]. If the second parameter of this function is set to *true*, it also returns the transaction data inside the block. The block object and transaction object format can be seen in Figures 4.1 and 4.2, respectively. Every field is in hexadecimal format. The block number, transaction index, sender address, recipient address, and transaction amount information are parsed from the *blockNumber*,*transactionIndex*, *from*, *to*, and *value* fields, respectively. If the transaction is a contract creation transaction, the field *to* will have the value *null* instead of the recipient address. When parsing the transactions, we replaced this *null* value with *0x0* to mark the transaction as a contract creation transaction.

```
{
  "difficulty":"0xc5d598778bd91",
  "extraData":"0x737061726b706f6f6c2d636e2d6e6f64652d3132",
  "gasLimit":"0x7a121d",
  "gasUsed":"0x79c7bf",
  "hash":"0x6b855d4ca375a859bb00ee9a70b0e51bcefac0b075e5855a39bc9e361005af89",
  "logsBloom":"0x0008484...",
  "miner":"0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c",
  "mixHash":"0x752aa195c7f86065068ee7800d7246d4456a65eda1abdcf7d3f28804bb95ae7f",
  "nonce":"0xda299d681573218d",
  "number":"0x5b8d82",
  "parentHash":"0xa618c3ac198431ead17dd143f3fab8a73da6a3c506764dff84108bd06f72eb82",
  "receiptsRoot":"0x01b161b3548f26e828eb995140ac3727d1378eebcbe0034965ad08b6d404ab5b",
  "sha3Uncles":"0x1b4bc85c0f4b02f6ca8bd1b48e3ae72b64d7131a530d91bb41a679115cc5b07e",
  "size":"0x3ebb",
  "stateRoot":"0x31152daddddd075c01bf7c2c8ecfca2a9e75c02b67d47466226aeb7c1b14d229",
  "timestamp":"0x5b524653",
  "totalDifficulty":"0x12950c319cf9901c8a5",
  "transactions":[ {"...": "..."}, {"...": "..."}],
  "transactionsRoot":"0xd01ba0158ef1b57a183848f1029532ce8c9fbdabd89ce056c056d212e2172a5f",
  "uncles":[
    "0x1867ebdfbd39b6638a792fcf297c515d858198ea7c27fe90d10270e745cbb608"
  ]
}
```

Figure 4.1. Ethereum block object in JSON.

```
{
  "blockHash": "0xfc8c6fc69291422825ebb826f9eed0e360f1da6837281c652542c8074b01ff35",
  "blockNumber": "0xb74404",
  "from": "0x49b21bdfa30333858956342f4028ce72e37eb851",
  "gas": "0x186a0",
  "gasPrice": "0x2540be400",
  "hash": "0xdb627936c0d5f9b97dc2a5a939a416c9ec3ec081f762c03dd1b3dee82ae765d2",
  "input": "0x",
  "nonce": "0x32697",
  "to": "0x4a4aa09ee61939e2e544a940d03cc0f796e2abe8",
  "transactionIndex": "0x0",
  "value": "0xe2717cab4642000",
  "v": "0x25",
  "r": "0x639b3d18ac170188338f8fb4098bedc524ce8c6468a0be76ac6294137b198bff",
  "s": "0x4adf43fdafb198272827d6f1c237fb85194e7de3320ccb744e85386c217a7389"
}
```

Figure 4.2. Ethereum transaction object in JSON.

4.1.3.2. ERC20 Token Transactions.

Until now, the parsing of the Ethereum transfer transactions is explained. We also extracted the transactions of popular ERC20 tokens. There are two ways to extract ERC20 transactions. One way is to listen to the *Transfer* events emitted from the token smart contracts. The other way is to parse the function call data. We choose the latter since the former requires you to be connected to a node and the latter can be extracted from the transaction data we already downloaded. Another reason for choosing the call data parsing is that there may be some ERC20 token contracts that have not implement the ERC20 standard properly and do not emit *Transfer* events.

The function call data can be found in the *input* field of the transaction data. In Figure 4.2, it can be seen that this field is empty because no function is called in this transaction. In Figure 4.3, an ERC20 transfer function call in the *input* field of the transaction can be seen. In Figure 4.4, the decoded version of the input data of the transaction can be seen. The input data's first 4-bytes (8 characters in hexadecimal format) represents the function signature. For *transfer* function, it is *a9059cbb* in hex. For *transferFrom* function, it is *23b872dd* in hex. The next part is the arguments of the function. The first argument has the type *address* and represents the recipient address. An Ethereum address is 20 bytes, but it is encoded as 32 bytes with zero padding in the input data. The second argument is the amount of the transaction. It is also 32 bytes and zero-padded. The sender address is the *from* field in the transaction. The *to* field in the transaction is the contract address whose function is called. The type of token can be found using this address. We manually collected this ERC20 token contract address and token symbol mapping information before parsing the transactions. In Figures 4.5 and 4.6, an example transaction for transferFrom function call can be found. The only difference here is that instead of taking the sender address from the *from* field in the transaction data, we take the sender address from the first argument of the function call in the *input* field.

```
{
  "blockNumber": "0x7a1200",
  "from": "0x6748f50f686bfbca6fe8ad62b22228b87f31ff2b",
  "hash": "0xeaa62fbad7a4bb97f34bda732b2ba5df4bb3c8b593eb80d91cb9a1f2b91d784c",
  "input": "0xa9059cbb000000000000000000000000add72aeef293eebe69a0592baf583fdaa47d8cbe0000000000000000
            000000000000000000000000000000000000000053448740",
  "to": "0xdac17f958d2ee523a2206206994597c13d831ec7",
  "transactionIndex": "0x19",
  "value": "0x0",
  "...": "..."
}
```

Figure 4.3. Ethereum ERC20 transfer transaction example in JSON (unrelated fields are omitted) (input field is splitted into two lines since it did not fit to the page).

```
// Function: transfer(address _to, uint256 _value)
0xa9059cbb // Transfer function signature
000000000000000000000000add72aeef293eebe69a0592baf583fdaa47d8cbe
// Recipient address: 0xadd72aeef293eebe69a0592baf583fdaa47d8cbe
0000000000000000000000000000000000000000000000000000000053448740
// Amount: 0x53448740 or 1397000000
```

Figure 4.4. Ethereum ERC20 transfer input data (splitted and annotated).

```
{
  "blockNumber": "0x8e9623",
  "from": "0xe6cab379cfa1ee999ca7d5fae3ceb2247eca9640",
  "hash": "0x56d3594e13861c1efa58929b98c4bdede6741f0120447d513036d3e3e748980c",
  "input": "0x23b872dd0000000000000000000000006903b4ab7d5a1a01cca0fc9fe1531bc5cbba26e40000000000000000
            0000000057e52d2d195359ae566e45c164d7243596ddf735000000000000000000000000000000000000000000
            000000000000006679c192",
  "to": "0xdac17f958d2ee523a2206206994597c13d831ec7",
  "transactionIndex": "0x2f",
  "value": "0x0",
  "..." : "..."
}
```

Figure 4.5. Ethereum ERC20 transferFrom transaction example in JSON (unrelated fields are omitted) (input field is splitted into several lines).

```
// Function: transferFrom(address _from, address _to, uint256 _value)
0x23b872dd // TransferFrom function signature
0000000000000000000000006903b4ab7d5a1a01cca0fc9fe1531bc5cbba26e4
// Sender address: 0x6903b4ab7d5a1a01cca0fc9fe1531bc5cbba26e4
00000000000000000000000057e52d2d195359ae566e45c164d7243596ddf735
// Recipient address: 0x57e52d2d195359ae566e45c164d7243596ddf735
000000000000000000000000000000000000000000000000000000006679c192
// Amount: 0x6679c192 or 1719255442
```

Figure 4.6. Ethereum ERC20 transferFrom input data (splitted and annotated).

4.1.3.3. Failed Transactions.

Unlike Bitcoin blocks, the blocks in Ethereum can contain failed transactions. An Ethereum transaction may fail because the transaction is reverted. Such a transaction is included in the block because the transaction execution consumes blockchain resources, and when it is included, the user has to pay gas. It is a precaution against the spamming of the blockchain network.

The *status* field of the transaction receipt of a transaction can be checked to find whether a transaction is failed for the transactions after the Byzantium fork [56] (for the blocks whose block number is greater than or equal to 4 370 000). This field has the value 0 for failed transactions and 1 for successful transactions. For example, the transaction in Figure 4.7 is a failed transaction. The transaction receipt can be obtained via *eth_getTransactionReceipt()* function [55].

```
{
  "blockHash": "0xb1fcff633029ee18ab6482b58ff8b6e95dd7c82a954c852157152a7a6d32785e",
  "blockNumber": "0x42ae50",
  "contractAddress": null,
  "cumulativeGasUsed": "0x4f037",
  "from": "0xa1619bda3f5160d19df5f9358f76a3e9bc893ed6",
  "gasUsed": "0xc350",
  "logs": [],
  "logsBloom": "0x00...0",
  "status": "0x0",
  "to": "0x1f573d6fb3f13d689ff844b4ce37794d79a7ff1c",
  "transactionHash": "0xeec4ccd13fe05907f9d732a8ad245bcb7f918217157b89baaa23895c12eb329a",
  "transactionIndex": "0xa"
}
```

Figure 4.7. Ethereum transaction receipt in JSON.

Finding whether a transaction is failed for the blocks before Byzantium fork is more complex. We need to check the transaction trace using *debug_traceTransaction()* function [57] and find if the trace contains any errors.

Checking every transaction receipt and trace would be time-consuming, and we had to connect a node to get the trace information. Therefore, we used the Google BigQuery service to find the block number and transaction index of the failed transactions instead of using *eth_getTransactionReceipt()* and *debug_traceTransaction()*. The SQL queries that we used to get failed transaction information can be seen in Figures A.2 and A.3. We downloaded the results of these queries and input them into our Ethereum transaction parser.

We added the failed transactions to our data with their symbol field prefixed *F-*. We did not omit the failed transactions in case they may provide some extra information for the analysis that we will make.

4.1.3.4. Block Rewards. There are four types of block rewards in Ethereum, namely static block reward, uncle reward, uncle inclusion reward, and transaction fee. The static block reward is the reward that the miner of a block gets for mining the block. The reward amount is hardcoded to the clients [58] and may be updated with the new hard forks. We collected the reward amount and the block range of the reward manually and added this information to our Ethereum transaction parser. We used the same symbol for the static block reward with the ether transactions. We set the sender address as *ETHMAINBLOCK*. The reward recipient is the *miner* field in the block (see Figure 4.1). We set the transaction index as zero and shifted the transaction index of all other transactions by one.

The uncle reward is the reward that the miner of the uncle block gets when that block is included in a block. The reward amount is $(uncle\_block\_number + 8 - block\_number) * block\_reward/8$. We did not include this reward in our data because we only downloaded the blocks for the valid chain. To add this reward, we also need to download the uncle blocks, but to download them, first, we need to know which uncle blocks are included in a block. So, this was a two-step process and would slow down the data collection step.

The uncle inclusion reward is the reward that the miner of a block gets because it includes uncle blocks. The reward amount is $block\_reward/32$ for each included uncle block. We used the same symbol for the uncle inclusion reward with the ether transactions. We set the sender address as *ETHMAINUNCLE*. The recipient of the reward is the *miner* field in the transaction. We set the transaction index as zero. The number of uncle blocks included in the block can be found by finding the length of the *uncles* array field in the block (see Figure 4.1).

We did not include transaction fees in our data since its calculation required the transaction receipt information.

4.1.3.5. Genesis Block Transactions. Genesis block contains all the necessary information to configure a blockchain network. It has a *alloc* field to define the initial balance of the wallets. The genesis block for Ethereum mainnet has 8893 account and balance pairs. These accounts belong to anyone who bought ether in the presale period. We take this information from [59] and add it as the zeroth block transactions. It can also be found in the source code of the Go Ethereum client in recursive length prefix (RLP) encoded form [60].

4.2. Blacklisted Address Data Collection

We collected blacklisted addresses for Bitcoin and Ethereum from various websites manually. The count of the blacklisted addresses is given in Table 6.1(f).

4.2.1. Bitcoin Blacklisted Address Data Collection

4.2.1.1. Data Sources. The blacklisted addresses for Bitcoin are collected from Bitcoin Who's Who blog post [61], Chainalysis blog posts [62–65], Sextortion Spam Bitcoin list [66], COVID-19 themed cryptocurrency scams list [67], Ciphertrace blog [68], Ransomware in the Bitcoin Ecosystem dataset [69, 70], CryptoScamDB [71], and OFAC SDN list [11].

4.2.1.2. Data Cleaning. The dataset [69] contains 11 entries that are not Bitcoin addresses but Bitmessage addresses. Ten of these addresses start with 2, and one address starts with N. Bitcoin mainnet addresses only start with 1, 3, or bc1. We removed them from our blacklist data.

4.2.2. Ethereum Blacklisted Address Data Collection

4.2.2.1. Data Sources. The blacklisted addresses for Ethereum are collected from MyEtherWallet blacklist [72], CryptoScamDB [71] (previously called EtherScamDB), Etherscan Label Word Cloud [73], and USDC banned addresses list [74]. From Etherscan, the addresses with the following labels are collected: Bitpoint Hack, Cryptopia Hack, EtherDelta Hack, Heist, Lendf.me Hack, Phish / Hack, Plus Token Scam, Ponzi, Spam Token, and Upbit Hack.

4.2.2.2. Data Preprocessing. The collected addresses are lowercased since the Ethereum addresses are hexadecimal, and multiple different address strings can represent the same address in the hexadecimal form. For example, "0xa" and "0xA" represent the same value, but they are different strings. After lowercasing, the duplicate addresses are removed.

5. BLOCKCHAIN TRANSACTION GRAPH SYSTEM

5.1. System Architecture

Figure 5.1 depicts the architecture of the blockchain transaction graph system. The ether, ERC20 token, and bitcoin transaction are extracted from Ethereum and Bitcoin blockchains, parsed, and stored as text files, as explained in Section 4.1. Blacklisted addresses are collected from various websites manually, as explained in Section 4.2. The blockchain transactions are input into the HPC cluster, and distributed transaction graph is constructed. This transaction graph is analyzed using graph algorithms.

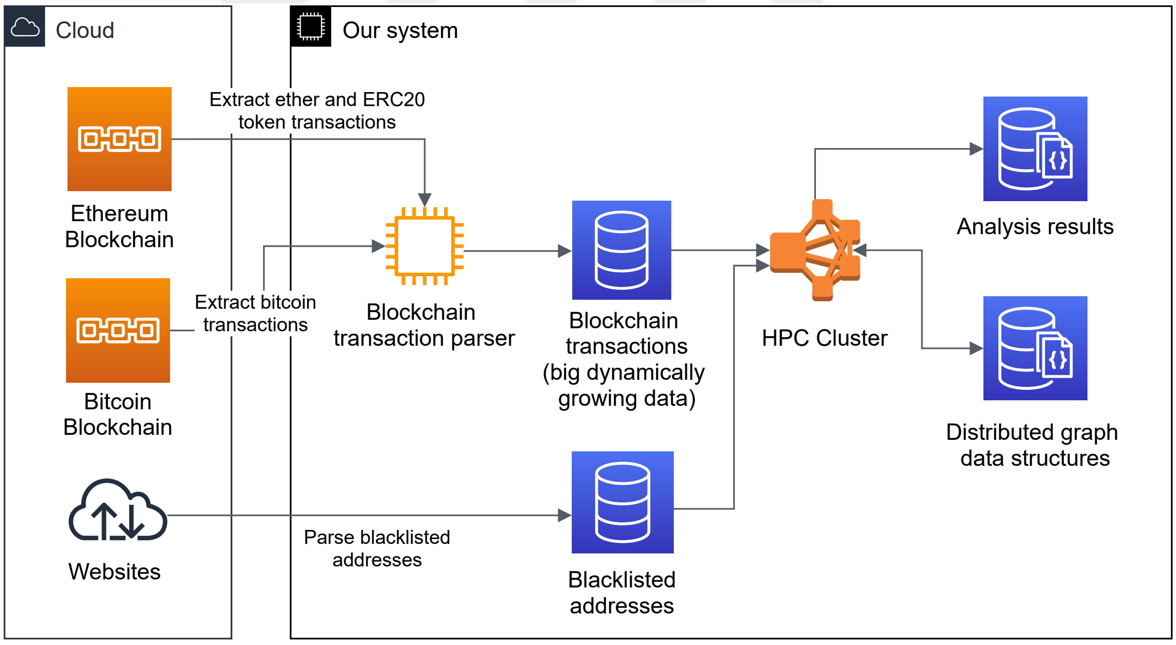

Cloud
Our system
Extract ether and ERC20 token transactions
Ethereum Blockchain
Extract bitcoin transactions
Bitcoin Blockchain
Websites
Blockchain transaction parser
Blockchain transactions (big dynamically growing data)
Parse blacklisted addresses
Blacklisted addresses
HPC Cluster
Analysis results
Distributed graph data structures

Figure 5.1. Blockchain transaction graph system architecture.

Figure 5.2 depicts the software stack of the blockchain transaction graph system. The undermost blue layer illustrates the parallel programming libraries. We used C++ language and the MPI library to implement this system. The second layer of the stack is responsible for the parallel scalable graph construction, sorting, and dynamic distributed graph data structures components of our system. This layer gives service to the third layer, the graph analysis layer that includes graph queries and algorithms.

MPI was chosen to implement the system since it is the de-facto standard communication library that is used on high-performance clusters. In addition, the system can be better optimized since MPI is a low-level library.

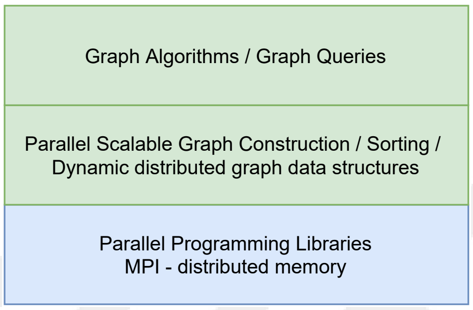

Graph Algorithms / Graph Queries
Parallel Scalable Graph Construction / Sorting / Dynamic distributed graph data structures
Parallel Programming Libraries
MPI - distributed memory

Figure 5.2. Blockchain transaction graph system software stack.

5.2. Data Structures

Distributed data structures are used while implementing the system. We frequently used distributed arrays. Distributed compressed sparse row (CSR) format is used when storing the transaction graph because the transaction graph is sparse. This format is prevalently used when storing graphs. We followed [75] for this format, and you can find its details in it. We also explained the basics of distributed array and CSR in Subsection 2.2.2. We modified the CSR format slightly since there can be multiple edges between nodes, i.e., multiple transactions between a sender and receiver. We stored an array of edge weights (transaction amounts) in each matrix cell instead of storing a single weight value.

5.2.1. Transaction to Graph Mapping

Blockchain transactions can be thought of as a directed graph. The tail of an edge is the address of a transaction sender. The head is the recipient, and the weight of the edge is the amount of transaction. Since Bitcoin and Ethereum have different token models (described in Subsection 2.1.1), we converted them to a common format (see Subsection 4.1.1).

The symbols and definitions of the transaction graph are given in Table 5.1. For Ethereum, we have both ether and ERC20 token transfer transactions of tokens listed in Table 4.1. The union of the set of externally owned account addresses V_c and the set of smart contract addresses V_s makes up the nodes V of the distributed transaction graph. Each Ethereum transaction E_{ETH} and token transfer transaction E_t is an edge E of the transaction graph.

Table 5.1. List of symbols and their meanings for transaction graph.

Symbol	**Meaning**
E	All transfer transactions. $E = E_{\mathrm{ETH}} \cup E_t$ for Ethereum or $E = E_{\mathrm{BTC}}$ for Bitcoin
E_{BTC}	Transfers between accounts and transaction IDs $E_{\mathrm{BTC}} = \{<a,t>: a \in V_a \wedge t \in V_t\} \cup \{<t,a>: t \in V_t \wedge a \in V_a\}$
E_{ETH}	All Ethereum blockchain transactions with zero or more ether payments
E_t	ERC20 token t transfer transactions, $t \in T$
T	Set of major ERC20 tokens tracked, $T = \{\mathrm{USDT,PAX,TRYB},\ldots\}$, full list given in Table 4.1
V	All blockchain addresses. $V = V_c \cup V_s$ for Ethereum or $V = V_a \cup V_t$ for Bitcoin
V_a	Set of Bitcoin addresses
V_c	Set of Ethereum externally owned account addresses
V_s	Set of Ethereum smart contract addresses
V_t	Set of transaction IDs (TXIDs)

For Bitcoin, the union of the set of addresses V_a and the set of transaction IDs V_t makes up the nodes V of the distributed transaction graph. Each input and output

UTXO of a Bitcoin transaction is an edge E of the transaction graph. From now on, when we refer to the nodes, we will only mention the addresses and not the transaction IDs because mentioning both make the descriptions complicated. However, note that the nodes of the transaction graph for Bitcoin also contain the transaction IDs.

Figure 5.3 shows the distributed transaction graph for (a) account-based and (b) UTXO-based blockchain graphs. In Figure 5.3(b), transaction IDs are shown with a square shape to make the graph more understandable. The nodes at the tail of incoming edges to the square shapes are the addresses that own the input UTXOs. The nodes at the head of outcoming edges from the square shapes are the addresses that own the output UTXOs.

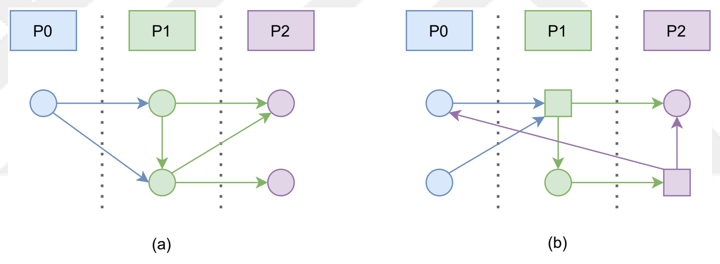


Figure 5.3. Distributed data structures of (a) account-based and (b) UTXO-based blockchain graphs.

5.3. Distributed Transaction Graph Construction

The distributed transaction graph construction algorithm is presented in Figure 5.4 and the symbols used in it are described in Table 5.2. The details of the nodes and edges for Ethereum and Bitcoin graphs are explained in Subsection 5.2.1 and will not be repeated here.

The first phase (lines 2-6) of distributed transaction graph construction involves building the whole node set V of the graph by finding unique addresses and giving them global IDs. We need these global IDs to build the transaction graph. Our blockchain transaction data is stored in multiple files. Each line in these files stores the information of one transaction, such as sender address, receiver address, and transaction amount. The detailed transaction format is given in Subsection 4.1.1. Each process takes a subset of these files as input, reads the files, and creates the set of sender (s) and receiver (t) addresses (line 2). This is the local address set.

Table 5.2. List of symbols and their meanings for graph construction.

Symbol	**Meaning**
E	All transfer transactions
E^p	Transfer transactions on process p
$G(V, E)$	Transaction graph
$G^p(V^p, E^p)$	Transaction subgraph on process p
$ID(v)$	Global ID of address v, $ID(v) \in [0, \lvert V\rvert - 1]$
P	Number of processes
p	ID of current process (0-indexed)
V	All blockchain addresses
V^p	Blockchain addresses on process p

The address set is copied to an array because we cannot sort a set. The address set is sorted using parallel sample sort (line 3) [76]. This parallel sample sort is not local. It sorts globally all addresses in the address array in each process. For better understanding, the global sorting can be thought of as taking the address array of each process, concatenating these arrays, sorting the concatenated array, splitting the sorted array, and putting the addresses back into the address array in each process. The sample sort is not centralized but distributed and works in parallel.

Before sorting the addresses, the addresses in each process were locally unique, but they are not necessarily unique after sorting them. Suppose that the same address is on the first and second processes. These two addresses will be on the same process when the addresses are globally sorted. Therefore, the duplicate addresses need to be removed after the sorting operation. The duplicate addresses are removed by comparing the consecutive addresses and taking only the addresses that are not the same (line 4). Comparing the boundary values, i.e., the last address in one process and the first one in the next process, is unnecessary since sample sort places the same elements on the same processes.

At this point, we have the unique addresses and need to give each address a global ID. The addresses in all processes as a whole stand for our addresses. The ID of the first address in the current process is found by summing the number of addresses in the processes, whose ID ranges from 0 to the current process ID (excluding) (line 5). The ID of an address is the sum of the first address's ID and the address's index in the array (line 6).

In the second phase (lines 7-28), the adjacency list of the graph is constructed from the transactions and global IDs of the transaction addresses. The sender and receiver addresses must be mapped to their corresponding global IDs. Since the address set is distributed, each process has only a part of the addresses. However, we need to know the global ID of all addresses to be able to map them. To solve this problem, each process sends its addresses to the next process and receives the addresses from the previous process in a ring fashion. This send and receive operation is done for the number of process times. At each iteration, the process will have new addresses and will be able to map these addresses to their corresponding global IDs. Two buffers with the size of the largest address set are used to be able to both send the addresses to the next process and receive the addresses from the previous process simultaneously (lines 7-8). The buffer that contains the addresses will alternate with every iteration (line 10). The other buffer only receives the addresses for the next iteration (line 11). Each process iterates the number of process times (line 9) and processes the address information again (line 12). If the sender or receiver address is found in the address set (located in the buffer), the address is given its global ID (lines 13-20). The search for an address in the buffer can be done using binary search since the addresses are in order. The current address set is sent to the next process (line 22). The address set to be used at the next iteration is received from the previous process (line 23).

```
function GenerateGraph($E^p, p$)
    $V^p \leftarrow \bigcup_{(s,t)\in E^p} \{s, t\}$
    $SA^p \leftarrow$ parallelSampleSort($V^p$)                              ▷ sorted array
    $V^p \leftarrow \{SA_i^p \mid i = 1 \lor (i \in [2..|SA^p|] \land SA_i^p \neq SA_{i-1}^p)\}$   ▷ rem. duplicates
    $IdStart \leftarrow \sum_{j=0}^{p-1} |V^j|$                              ▷ by parallel scan
    $ID(v_j) = IdStart + j \quad \forall j \in [0..V^p - 1]$
    $SR_0^p \leftarrow \{(v, ID(v)) \mid v \in V^p\}$                      ▷ send/recv buffer
    $SR_1^p \leftarrow \varnothing$                                          ▷ send/recv buffer
    for $i \in [0..P-1]$ do
        $j \leftarrow i$ mod 2                                           ▷ index of current SR
        $k \leftarrow i + 1$ mod 2                                       ▷ index of SR to recv.
        for each $(s,t) \in E^p$ do
            $sID \leftarrow$ binarySearch($SR_j^p, s$)
            if $sID \neq null$ then
                $ID(s) \leftarrow sID$
            end if
            $tID \leftarrow$ binarySearch($SR_j^p, t$)
            if $tID \neq null$ then
                $ID(t) \leftarrow tID$
            end if
        end for
        Send $SR_j^p$ to process $(i+1)$ mod $P$                  ▷ send SR to next proc.
        Receive $SR_k^p$ from process $(i-1+P)$ mod $P$ ▷ recv. from prev. proc.
    end for
    $IDE^p \leftarrow \{(ID(s), ID(t)) \mid (s,t) \in E^p\}$
    $SIDE^p \leftarrow$ parallelSampleSort($IDE^p$)
    formAdjacencyListofGraph($SIDE^p$)
    return $G^p(V^p, E^p)$
end function
```

Figure 5.4. Distributed transaction graph construction algorithm.

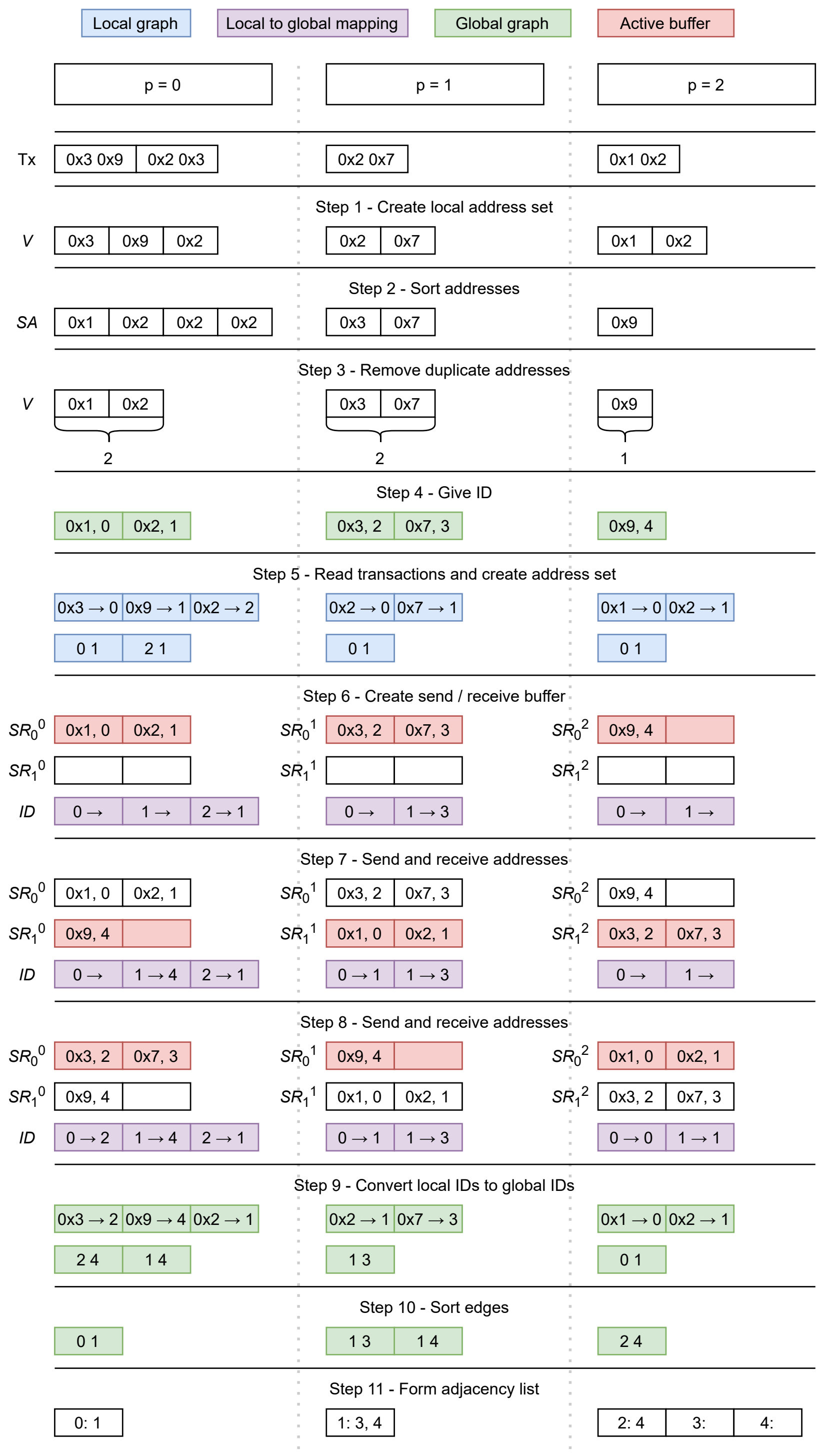


Figure 5.5. Example showing the steps of distributed transaction graph algorithm.

The edges of the graph are converted from local IDs to global IDs (line 25). The edges are sorted according to IDs of address pairs, $(ID(s), ID(t))$, making up the transactions (line 26), and the adjacency list representation of the graph is formed. This adjacency list is the output of the algorithm. Figure 5.5 illustrates the execution of our algorithm on a small example dataset.

In addition to the directed graph, we also constructed the transpose and undirected graphs. We can construct the transpose graph by continuing at line 26, switching the tail and head node IDs, sorting the edges again, and building the adjacency list representation. We can construct the undirected graph by adding the reverse edges, sorting edges, removing duplicates, and building the adjacency list representation.

5.4. Graph Algorithms

Figure 5.6 shows an overview of the functionalities of our system and which graph algorithm uses which graph. The directed, transpose, and undirected graphs are constructed as described in Section 5.3. We do not show the graphs as distributed in Figure 5.6 not to complicate the figure. However, note that our graphs are stored in a distributed form. The constructed graphs are input to parallel graph algorithms of our system.

5.4.1. Distributed Calculation of the Blacklisted Address Trace Forest

When fraudulent activities are carried out on the blockchains, addresses that engage in fraud are usually posted on the Internet by companies or government agencies. We refer to these addresses as blacklisted addresses. We are interested in transactions that originate from these blacklisted addresses. Our system provides the capability to return subgraphs that contain transactions that trace to blacklisted addresses. We have also developed an additional algorithm that will output a more compact and optionally prunable forest of trace trees. Such a compact trace can be used by a web server to provide fast answers to trace queries since the traces are small and precomputed.

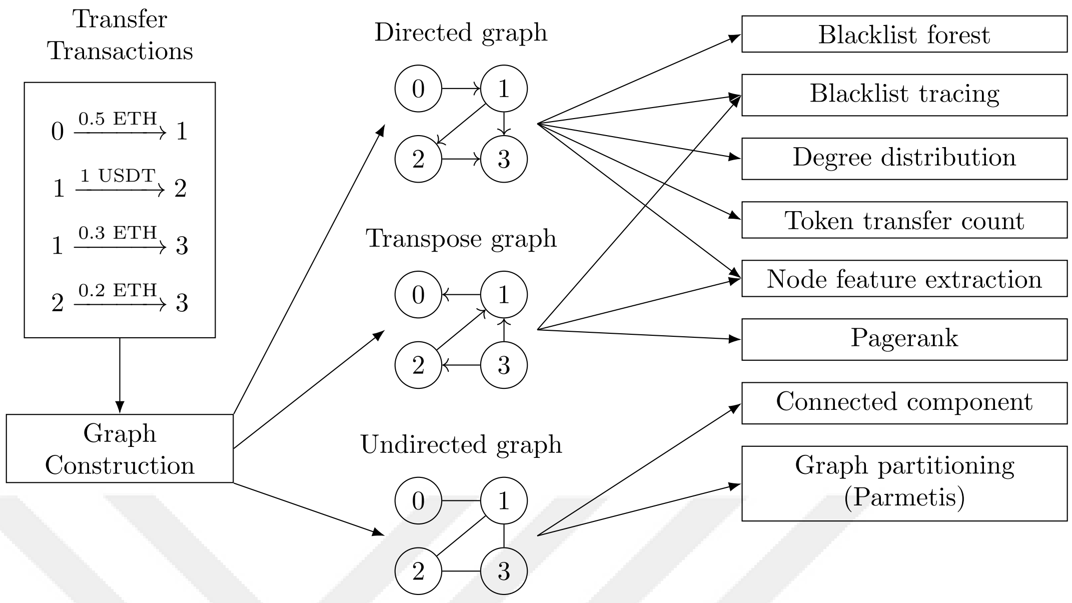


Figure 5.6. Parallel algorithms of the system.

The distributed calculation of the blacklisted addresses trace forest algorithm is presented in Figures 5.7 and 5.8, and the symbols used in it are described in Table 5.3. This algorithm takes distributed transaction graph and the ID of blacklisted nodes (addresses) and outputs a transaction trace forest of trees, whose roots are the blacklisted nodes, in a distributed array format. The algorithm can calculate the full shortest path based forest of trees. It can also calculate the forest of pruned trees up to depth D.

The algorithm traverses the nodes of the graph starting from blacklisted nodes and adds the visited nodes to the trace forest. Since the graph is distributed, not every node can be visited on a process. The algorithm first visits all local nodes and stores the remote nodes that need to be visited. The stored node information is then exchanged between processes. These new nodes are traversed locally again. This traversal and communication cycle continues until no remote node is left to be visited.

Each node in the forest F points to its parent. The root, which are blacklisted nodes in our case, points to itself. The depth D is used to store the distance of the node from the blacklist. It is also equivalent to the depth in the forest. The nodes that need to be visited are stored on a stack (line 2). Each process adds the blacklisted nodes that belong to the current process to the forest as a root and to its stack (lines 6-12). The total number of nodes in the stack of all processes is calculated (line 13). While there are nodes to be visited, the traversal and communication cycle will continue (lines 14-43).

Table 5.3. List of symbols and their meanings for trace forest calculation.

Symbol	**Meaning**
A^p	Set of addresses to be traversed on process p
B	Set of blacklisted addresses
C	Total number of parent node and depth pairs to be sent
D^p	Distance of addresses on process p from blacklisted address
E	All transfer transactions
E^p	Transfer transactions on process p
F^p	Trace forest on process p
$G(V, E)$	Transaction graph
$G^p(V^p, E^p)$	Transaction subgraph on process p
P	Number of processes
p	ID of current process (0-indexed)
R^p	Set of parent node and depth pairs received from other processes
S^p	Set of parent node and depth pairs on process p to be sent to remote processes
V	All blockchain addresses
V^p	Blockchain addresses on process p

In the traversal part (lines 15-28), a node is popped from the top of the stack (line 17) and its edges are visited if there is any node in the stack. If the head node of the edge is a local node (line 19), and if the node is unvisited or it is visited and a shorter path is found, the node is added to the stack, the tail node of the edge is set as its parent in the forest, and its depth is set as one added to the depth of the tail node (lines 47-51). If the head node of the edge is a remote node (line 21), the tail node and depth information is stored in a map S to be sent to the corresponding node in the communication part (lines 22-25). This tail node and depth information in the map is updated if a shorter path is found.

After the traversal part, the number of cut edges (edges whose tail and head are in different processes) is calculated to determine whether the cut edge communication part is needed (line 29). If there are cut edges, the cut edge information is sent and received. For each received node (line 37), if the node is unvisited or it is visited and a shorter path is found, the node is added to the stack, the tail node of the edge is set as its parent in the forest, and its depth is set as one added to the depth of the tail node (line 39).

At the end of the loop, the total number of nodes in the stack of all processes is calculated again (line 42). If there is any node in the stack, the traversal and communication cycle will continue. If not, the algorithm will return the forest. Figure 5.9 illustrates the execution of our algorithm on a small example dataset.

5.4.2. Distributed Blacklisted Address Transaction Tracing

Given a blockchain address of a customer and a set of blacklisted addresses, we want to identify a subgraph that traces transaction activities between the blacklisted addresses and the queried customer address. Two example trace subgraphs are presented in Figure 6.7. Such trace subgraphs are returned by the parallel tracing algorithm that is presented in this section.

The parallel blacklisted address transaction tracing algorithm is given in Figures 5.10 and 5.11. The symbols used in it are described in Table 5.4. The algorithm finds the subgraph between two sets of nodes. We use it to trace the set of queried addresses Q back to the set of blacklisted addresses B. The algorithm first finds the nodes (addresses) reachable from B denoted with RB by depth-first traversal (line 2). Then, it finds the nodes that can reach Q denoted with RQ by traversing the edges in

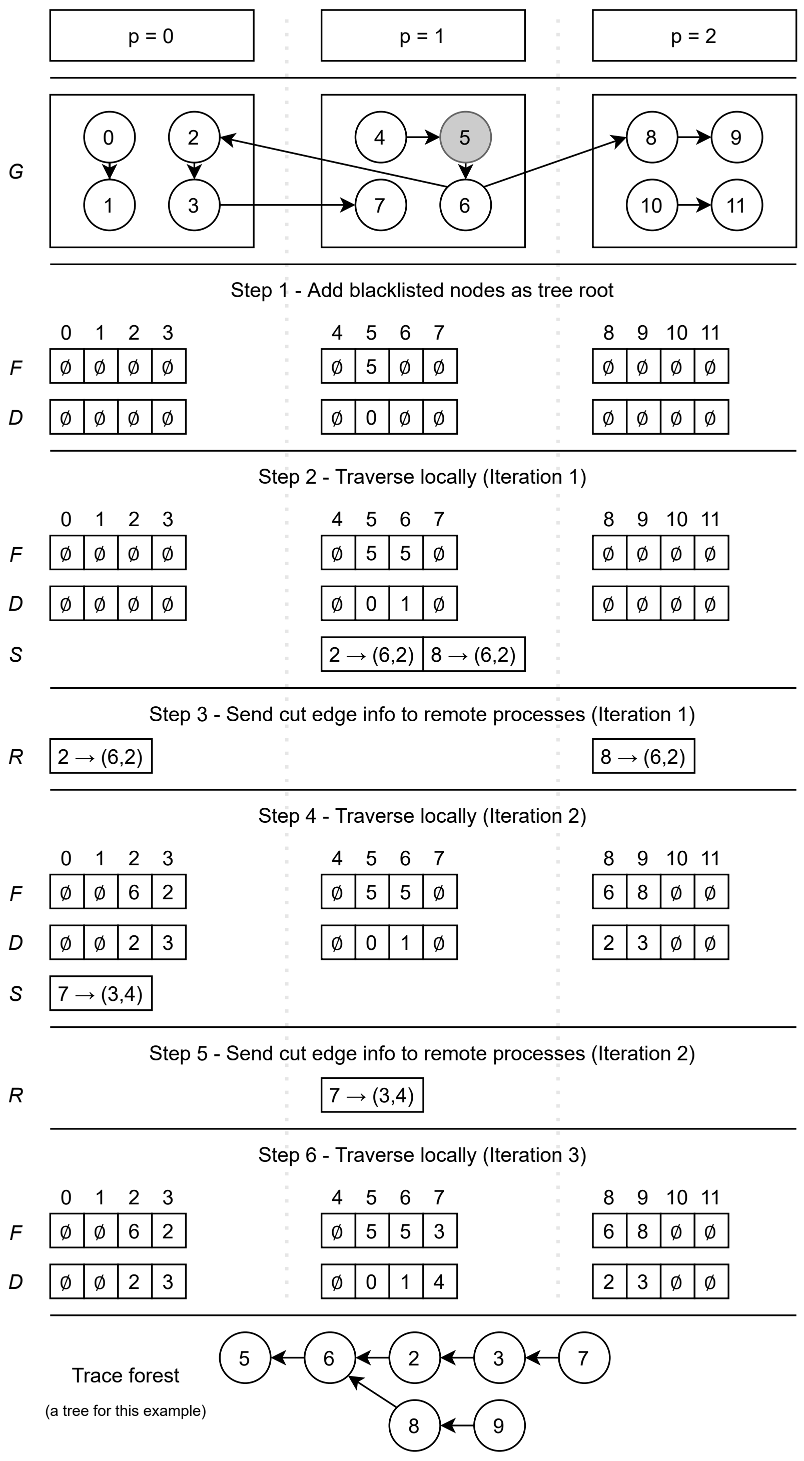
```
function BuildTraceForest($p, B, G^p(V^p, E^p)$)
    $A^p \leftarrow \varnothing$
    for all $v \in V^p$ do                              ▷ initialize the forest
        $F^p(v) \leftarrow \varnothing$
    end for
    for all $b \in B$ do                 ▷ add blacklisted addresses as root nodes
        if $b \in V^p$ then
            $F^p(b) \leftarrow b$                        ▷ root node points to itself
            $D^p(b) \leftarrow 0$
            $A^p \leftarrow A^p \cup \{b\}$                    ▷ add to stack to visit later
        end if
    end for
    $|A| \leftarrow \sum_{i=0}^{P-1} |A^i|$                                  ▷ allreduce
    while $|A| > 0$ do                            ▷ while there is nodes to visit
        $S^p \leftarrow \varnothing$
        while $A^p \neq \varnothing$ do               ▷ until no local node remained to visit
            $A^p \leftarrow A^p \setminus \{s\} \quad \exists s \in A^p$              ▷ pop a node from stack
            for all $(s,t) \in E^p$ do                   ▷ visit the neighbors of s
                if $t \in V^p$ then                              ▷ if local node
                    VisitNode($F^p(t), s, t, D^p(s)+1, D^p(t)$)
                else                                         ▷ if remote node
                    if $(S^p(t) = \varnothing) \vee$
                        $((S^p(t) \neq \varnothing) \wedge (D^p(s)+1) < d \mid (v,d) \in S^p(t)))$ then
                            $S^p(t) \leftarrow (s, D^p(s)+1)$
                    end if
                end if
            end for
        end while
```

Figure 5.7. Distributed calculation of the blacklisted address trace forest algorithm - part 1.

```
29:         $C \leftarrow \sum_{i=0}^{P-1} |S^i|$                                ▷ allreduce
30:         if $C > 0$ then                     ▷ if there is any node info to send/recv
31:             for all $i \in ([0 \mathinner{..} P-1] \setminus \{p\})$ do
32:                 Send $\{t \mid t \in S^p \wedge t \in V^i\}$ to process $i$
33:             end for
34:             for all $i \in ([0 \mathinner{..} P-1] \setminus \{p\})$ do
35:                 Receive $R^p$ from process $i$
36:             end for
37:             for all $t \in R^p$ do
38:                 $(s, d) \leftarrow R^p(t)$
39:                 VisitNode($F^p(s), s, t, d, D^p(t)$)
40:             end for
41:         end if
42:         $|A| \leftarrow \sum_{i=0}^{P-1} |A^i|$                               ▷ allreduce
43:     end while
44:     return $F^p$
45: end function

46: procedure VisitNode($f, s, t, d_1, d_2$)
47:     if $(f = \varnothing) \vee ((f \neq \varnothing) \wedge (d_1 < d_2))$ then        ▷ is null or shorter
48:         $A^p \leftarrow A^p \cup \{t\}$                          ▷ add head node to stack
49:         $F^p(t) \leftarrow s$                     ▷ head node points to tail node in tree
50:         $D^p(t) \leftarrow d_1$
51:     end if
52: end procedure
```

Figure 5.8. Distributed calculation of the blacklisted address trace forest algorithm - part 2.



p = 0
p = 1
p = 2
G
Step 1 - Add blacklisted nodes as tree root
F
D
Step 2 - Traverse locally (Iteration 1)
S
2 → (6,2)
8 → (6,2)
Step 3 - Send cut edge info to remote processes (Iteration 1)
R
Step 4 - Traverse locally (Iteration 2)
7 → (3,4)
Step 5 - Send cut edge info to remote processes (Iteration 2)
Step 6 - Traverse locally (Iteration 3)
Trace forest
(a tree for this example)

Figure 5.9. Example showing the steps of distributed calculation of the blacklisted address trace forest algorithm.

```
 1: function $\textsc{Trace}(p, P, G^p(V^p, E^p), B, Q, T_s, T_e)$
 2:     $RB^p \leftarrow \text{DFS}(p, P, G^p(V^p, E^p), B, T_s, T_e, false)$    ▷ nodes reachable from B
 3:     $RQ^p \leftarrow \text{DFS}(p, P, G^p(V^p, E^p), Q, T_s, T_e, true)$    ▷ nodes that can reach Q
 4:     $V'^p \leftarrow RB^p \cap RQ^p$    ▷ nodes of subgraph
 5:     $V' \leftarrow \bigcup_{i=0}^{P-1} V'^p$    ▷ MPI_Allgatherv
 6:     $E'^p \leftarrow \{e \mid e \in E^p \wedge e = (s, t, b) \wedge s, t \in V' \wedge b \in [T_s \,..\, T_e]\}$    ▷ edges of subgraph
 7:     return $G'^p(V'^p, E'^p)$
 8: end function
 9: function $\text{DFS}(p, P, G^p(V^p, E^p), F, T_s, T_e, Rev)$
10:     $R^p \leftarrow \varnothing$    ▷ nodes reachable from F
11:     $C^p \leftarrow \varnothing$    ▷ remote nodes to be visited
12:     $M^p \leftarrow \varnothing$    ▷ remote nodes that are already visited
13:     $S^p \leftarrow F \cap V^p$    ▷ stack of nodes to visit
14:     $TCN \leftarrow 1$    ▷ total number of remote nodes to be visited in all processes
15:     while $TCN > 0$ do
16:         while $S^p \neq \varnothing$ do    ▷ while there are nodes to visit
17:             $S^p \leftarrow S^p \setminus \{cur\} \quad \exists cur \in S^p$    ▷ pop a node from stack
18:             $R^p \leftarrow R^p \cup \{cur\}$    ▷ mark the node as visited
19:             for all $e \in E^p$ do
20:                 if $Rev$ then
21:                     $(t, s, b) \leftarrow e$
22:                 else
23:                     $(s, t, b) \leftarrow e$
24:                 end if
25:                 if $s = cur \wedge b \in [T_s \,..\, T_e]$ then
26:                     if $t \in V^p \wedge t \notin R^p$ then    ▷ if node is local and not already visited
```

Figure 5.10. Distributed blacklisted address transaction tracing algorithm - part 1.

```
27:                 $R^p \leftarrow R^p \cup \{t\}$          ▷ mark node as reached
28:                 $S^p \leftarrow S^p \cup \{t\}$          ▷ push node to stack
29:             else if $t \notin V^p \wedge t \notin M^p$ then     ▷ if node is remote and not already visited
30:                 $C^p \leftarrow C^p \cup \{t\}$          ▷ save the node send it later
31:             end if
32:         end if
33:     end for
34:     end while
35:     $TCN \leftarrow \sum_{i=0}^{P-1} |C^p|$          ▷ MPI_Allreduce
36:     if $TCN > 0$ then
37:         for all $i \in ([0\,..\,P-1] \setminus \{p\})$ do
38:             Send $\{t \mid t \in C^p \wedge t \in V^i\}$ to process $i$
39:         end for
40:         $CR^p \leftarrow \varnothing$
41:         for all $i \in ([0\,..\,P-1] \setminus \{p\})$ do
42:             Receive $CR_i$ from process $i$
43:             $CR^p \leftarrow CR^p \cup CR_i$
44:         end for
45:         $M^p \leftarrow M^p \cup C^p$ ▷ save sent nodes as visited not to send them again
46:         $C^p \leftarrow \varnothing$          ▷ clear the send nodes
47:         $S^p \leftarrow S^p \cup (CR^p \setminus R^p)$   ▷ push the not visited received nodes to the stack
48:     end if
49:   end while
50:   return $R^p$
51: end function
```

Figure 5.11. Distributed blacklisted address transaction tracing algorithm - part 2.

the reverse direction (line 3). The intersection of RB and RQ gives us the set of nodes in the subgraph V'^p (line 4). Each local set of nodes of the subgraph is exchanged between the processes and we get the global set of nodes V' in all processes (line 5). Each process finds the local edges whose tail and head are in the V' and is between the given block range from T_s to T_e (which also represents the time since all of the transactions in a block have the same timestamp) (line 6). T_s is the block range start and denotes the start time range of trace. T_e is the block range end and denotes the end time range of trace. The algorithm returns the subgraph (line 7).

The depth-first search function takes current process ID p, total process count P, the graph G^p, set of starting addresses F, block start T_s, block end T_s, and traversal direction Rev. It returns the set of nodes R^p that are reachable from the set of starting nodes F. C^p is the set of nodes that needs to be visited and is located on remote processes. M^p is the set of remote nodes that are already visited. S^p is the stack of nodes to be visited. At the start, S^p contains the nodes that are in F and is a local node (line 13). The algorithm first traverses the local reachable nodes (lines 16-34) and then sends the remote nodes to be visited C^p to corresponding processes (lines 36-48). This loop continues until no node is left in the stack of any processes. TCN is the total number of nodes in C^p in all processes. It is initialized with 1 to run the first loop (line 14). The nodes in the stack S^p is traversed until it is empty. At each step, a node *cur* is taken from the stack and marked as visited (lines 17-18). The nodes at the head of the outcoming edges of the node *cur* are traversed if *Rev* is false. The nodes at the tail of the incoming edges of the node *cur* are traversed if *Rev* is true (lines 19-33). If the edge is between the given block time range, it needs to be visited (line 25). If it is a local node and is not already visited, it is added to the stack (line 26). If it is a remote node and not already sent to other processes to be visited, it is added to the set of remote nodes to be visited C^p (line 29). The total number of nodes in C^p in all processes is calculated (line 35). If any nodes need to be send to remote processes and visited in any of the processes, the nodes C^p are exchanged between processes. The C^p is added to M^p not to send them again (line 45). The C^p is cleared. The received nodes are added to the stack S^p.

Table 5.4. List of symbols and their meanings for blacklisted address transaction tracing.

Symbol	**Meaning**
B	Set of blacklisted addresses
C^p	Set of unvisited remote addresses on process p
E	All transfer transactions
E^p	Transfer transactions on process p
E'^p	Transfer transactions of trace subgraph on process p
e	A transaction $e \in E$ and $e = (s, t, b)$ where s is the sender (tail of edge), t is the recipient (head of edge), and b is the block number of transaction
F	Set of starting addresses for graph traversal
$G(V, E)$	Transaction graph
$G^p(V^p, E^p)$	Transaction subgraph on process p
M^p	Set of remote addresses that are already visited on process p
P	Number of processes
p	ID of current process (0-indexed)
Q	Set of queried addresses
R^p	Set of addresses reachable from F on process p
Rev	Traversal direction. Outcoming edges if true. Incoming edges if false.
S^p	Set of addresses to be visited on process p
T_e	Block range start
T_s	Block range end
TCN	Total number of remote addresses to be visited in all processes
V	All blockchain addresses
V'	Set of addresses of trace subgraph
V'^p	Set of addresses of trace subgraph on process p

5.4.3. Parallel Graph Partitioning

Since our system is distributed, we cannot access all the data on the local process. We need to communicate with other processes to access remote data. For example, let two nodes be on different processes and a directed edge connect them. When visiting these two nodes via graph traversal, we need to transfer data from one process to another. Graph partitioning is an optimization where we try to minimize the number of edges that connect nodes in two different partitions while keeping the partitions load balanced (have a roughly equal number of nodes). Graph partitioning decreases the amount of communication required while processing the graph, thus increasing the system's performance.

Initially, we tried to parallel partition the blockchain transaction graphs with PT-SCOTCH 6.0.9 [77] and ParMETIS 4.0.3 [75]. Since PT-SCOTCH was slower than ParMETIS in our experiments, we opted for ParMETIS and integrated it into our system.

5.4.4. Connected Component Calculation

We implemented the parallel connected components algorithm of [78] named FastSV on our blockchain transaction graph system. The algorithm is presented as Algorithm 2 in the paper. We can find the connected component ID of a node and the total connected component count.

5.4.5. Node Feature Extraction

We calculated the following features for each node (address): outdegree, indegree, unique outdegree, unique indegree, total outgoing amount, total incoming amount, net balance, timestamp of the first transaction, timestamp of the last transaction, the difference between first and last transaction timestamps, whether the last transaction is outgoing, last transaction amount, average outgoing amount per transaction, average incoming amount per transaction.

5.4.6. Other Algorithms

In token transfer count, the number of ERC20 tokens is counted. In node degrees and degree distribution, we find the indegree and outdegree of every node and the number of nodes for each indegree and outdegree. In Pagerank, we calculate a ranking value for each node of the graph by using Pagerank.

6. EXPERIMENTS AND RESULTS

6.1. Test Environment Setup

We tested our blockchain transaction graph system on a 16 node (machine instance) cluster on Amazon Elastic Compute Cloud. For Ethereum tests, we used c5.4xlarge machine instances, each of which had 16 virtual central processing units (CPUs) (8 physical cores with hyper-threading), 32 GiB memory, up to 10 Gbps network bandwidth, and 4.750 Mbps storage bandwidth. For Bitcoin tests, we used r5b.4xlarge machine instances, each of which had 16 virtual CPUs, 128 GiB memory, up to 10 Gbps network bandwidth, and 10.000 Mbps storage bandwidth. We launched the cluster using StarCluster [79]. Since the development of StarCluster is stopped, we had to patch its source code by adding the instance types we used in the file *static.py*. StarCluster sets the placement group of machines as *cluster* so that the cloud service packs the instances close together. StarCluster also sets up Network File System (NFS) on the nodes so that the compute nodes can read data from the master node. We created a 450 GB gp3 elastic block storage (EBS) volume for Ethereum data to store that input and output data. We connected this storage to the master node of the cluster. For Bitcoin, we created a storage volume with 1200 GB. We did not use any job scheduler like Slurm. We run the MPI code directly with *mpirun* command.

6.2. Test Dataset

As datasets, we have used roughly twelve years of Bitcoin and five years of Ethereum blockchain data. Our Ethereum dataset also contains major ERC20 token transfer transactions. The details of the datasets are given in Table 6.1. How these datasets are collected is explained in Chapter 4. Our Ethereum dataset is available at [52].

Table 6.1. Bitcoin and Ethereum blockchain dataset statistics.

	Statistic	**Bitcoin**	**Ethereum**
(a)	Blocks	0 - 674 999	0 - 10 199 999
(b)	Time coverage of blocks	3.1.2009 - 17.3.2021	30.7.2015 - 4.6.2020
(c)	No. of transactions	625 570 924	766 899 042
(d)	No. of addresses	800 017 678	78 945 214
(e)	No. of 40 major ERC20 token transfer transactions	N/A	43 371 941
(f)	Number of blacklisted addresses	21 028	5 830
(g)	Uncompressed dataset size	382 GB	81 GB

6.3. Description of Tests

The descriptions of the tests that are carried out are given in Table 6.2. All these tests are programmed in C++ using the MPI libraries. All of the tests are distributed and parallel. In test T1, the transactions are read from text files, the directed, transposed, and undirected transaction graph is constructed, and the constructed graphs are saved to files. The distributed directed transaction graph construction algorithm is presented in Section 5.3. In test T2, we use ParMETIS [75] software in order to partition the undirected transaction graph in parallel. Test T3 runs the parallel pageranking algorithm on the transaction graph. Test T4 computes the node indegrees and outdegrees (number of incoming and outcoming transactions for each address) and degree distribution of the transaction graph. Test T5 computes the total number of transfer transactions of 40 major tokens. In test T6, the number of connected components in the transaction graph is calculated. In test T7, the shortest path based trace forest of blacklisted addresses is built. The algorithm for test T7 is presented in Subsection 5.4.1. In test T8, the following features are calculated for each address: outdegree, indegree, unique outdegree, unique indegree, total outgoing ether, total incoming ether, net ether balance, timestamp of the first transaction, timestamp of the last transaction, the difference between first and last transaction timestamps, whether the last transaction is outgoing, last transaction amount, average outgoing ether per transaction, average incoming ether per transaction. In test T9, the subgraph between a blacklisted node and a query node for a given time range is found. The algorithm for test T9 is presented in Subsection 5.4.2.

Table 6.2. Description of tests.

Test	**Description**
T1	Transaction graph construction
T2	Graph partitioning using ParMETIS [75]
T3	Pageranking on transaction graph
T4	Node degrees and degree distribution of transaction graph
T5	No. of transfer transactions of 40 major ERC20 tokens
T6	Connected component count
T7	Shortest path based blacklisted address trace forest
T8	Extracting node features
T9	Example of trace subgraph of a blacklisted address

6.4. Test Results

We ran the tests two times and took the average of the timings. We ran the test with different numbers of nodes. The Ethereum tests are run on 4, 8, 12, and 16 nodes. We could not run the tests with 1 or 2 nodes because the memory of the instances was insufficient. Since the program uses distributed memory, the memory used per node increases when we decrease the number of nodes. The Bitcoin tests are run on 8, 12, and 16 nodes. Since each node in the cluster has 16 virtual CPUs, we also run the tests with different MPI processes per node, namely 1, 2, 4, 8, 12, and 16 MPI processes per node.

In the timing result figures, we grouped the timings by MPI processes per node. The connected lines show the execution time when we increase the number of nodes and keep the MPI processes per node constant.

The timing result for transaction graph construction is given in Figure 6.1. Since graph construction involves reading transactions from disk and then uses a ring communication (see the algorithm in Figure 5.4) to implement an all-to-all communication in order to construct the graph, we can expect its timing to level off and even increase after the number of nodes is increased. Figure 6.1 shows this happening. In the case of Ethereum, the timing levelled off. For Bitcoin, since the number of addresses is much higher, we see the times decreasing and then increasing due to increased communication cost.

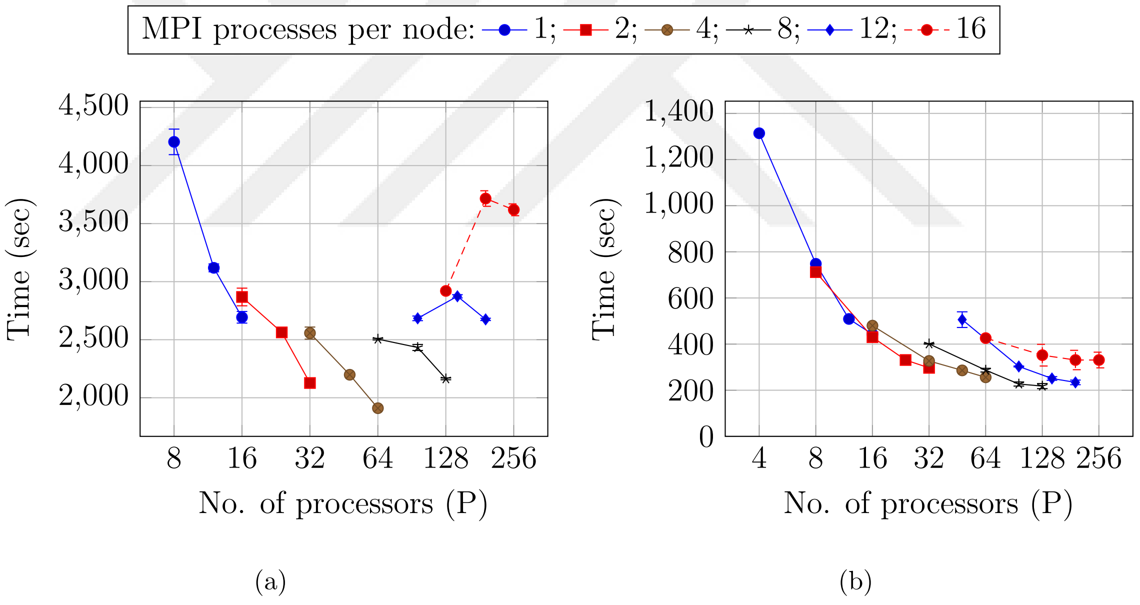


Figure 6.1. Test T1: Graph construction times for (a) Bitcoin and (b) Ethereum on the HPC cluster.

Test T2, which is about graph partitioning, takes hours for Ethereum and gives out of memory error for Bitcoin. Since tests T3-T9 take minutes or seconds for the Ethereum transaction graph and the sum of their timings is smaller than T2, it is not worth applying graph partitioning using ParMETIS.

Figure 6.2 shows the execution times for the Pagerank computation. Note that the graph resides in memory after graph construction, and the pagerank computations are run on the distributed data structures. We see faster processing times when the number of processors is increasing.

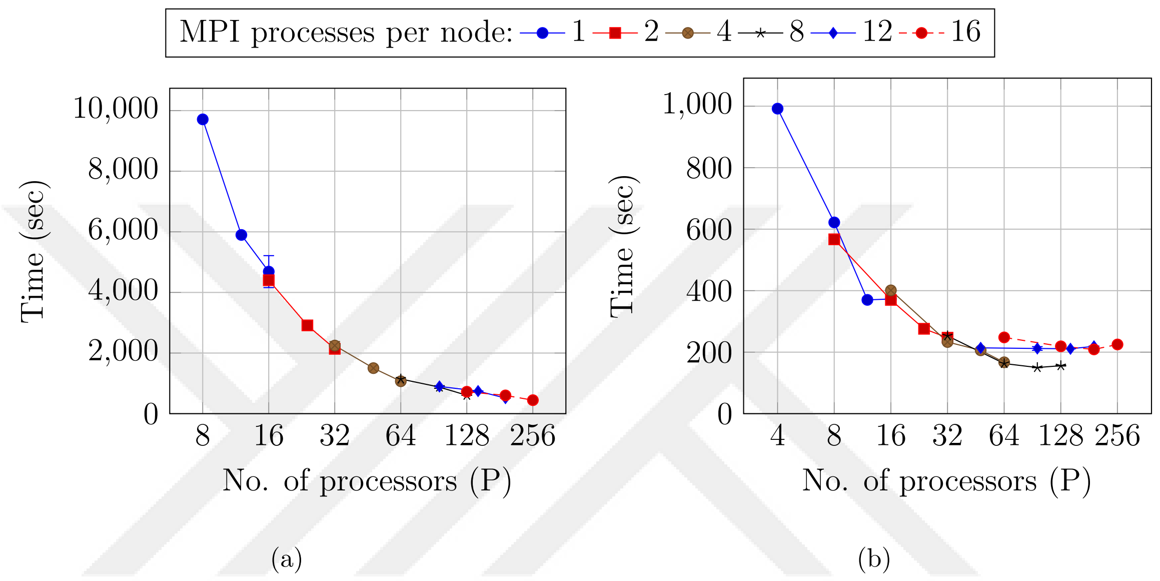


Figure 6.2. Test T3: Pageranking times for (a) Bitcoin and (b) Ethereum on the HPC cluster.

Table 6.3 shows the addresses that ranked in the top 10 in Pagerank. For Bitcoin, the most important addresses belong to exchange companies and popular gambling sites. For Ethereum, the most important addresses belong to exchanges and token contracts. Plus Token, which ranked fourth in our Pagerank calculation for Ethereum, is a ponzi scheme [80] that operated for more than one year and collapsed in 2019.

Table 6.4 shows the total number of distinct addresses at the tail/head of incoming/outcoming transactions to/from k most important addresses. Since Bitcoin is UTXO-based, it is expected that address reuse is seldom, so we have lower percentages. On the other hand, since Ethereum is account-based, address reuse is more frequent, and the distinct address percentages are higher than Bitcoin. About half of the addresses directly transacted with the most important 1000 Ethereum addresses. Most of the exchanges verify the identity of their users as a part Know Your Customer

(KYC) procedure, so the transactions coming from exchanges can be associated with a user and are less risky. The efforts can be concentrated on this small set of important addresses to see if their identity verification procedures are strong. If they are strong, then lower risk scores can be assigned to the addresses who transacted with them.

Table 6.3. Top 10 ranked addresses on Bitcoin and Ethereum transaction graph.

Rank	**Bitcoin**	**Ethereum**
1	1HckjUpRGcrrRAtFaaCAUaGjsPx9oYmLaZ Huobi (exchange)	0x3f5ce5fbfe3e9af3971dd833d26ba9b5c936f0be Binance (exchange)
2	1NDyJtNTjmwk5xPNhjgAMu4HDHigtobu1s Binance (exchange)	0xdac17f958d2ee523a2206206994597c13d831ec7 Tether USDT (token contract)
3	1NxaBCFQwejSZbQfWcYNwgqML5wWoE3rK4 LuckyB.it (gambling)	0x70faa28a6b8d6829a4b1e649d26ec9a2a39ba413 ShapeShift (exchange)
4	1dice8EMZmqKvrGE4Qc9bUFf9PX3xaYDp Satoshi Dice (gambling)	0xf4a2eff88a408ff4c4550148151c33c93442619e Plus Token (ponzi [80])
5	1FoWyxwPXuj4C6abqwhjDWdz6D4PZgYRjA Binance (exchange)	0xac08809df1048b82959d6251fbc9538920bed1fa MSD (contract)
6	1G47mSr3oANXMafVrR8UC4pzV7FEAzo3r9	0xfa52274dd61e1643d2205169732f29114bc240b3 Kraken (exchange)
7	1dice97ECuByXAvqXpaYzSaQuPVvrtmz6 Satoshi Dice (gambling)	0xbcf935d206ca32929e1b887a07ed240f0d8ccd22 Million Money (contract)
8	37Tm3Qz8Zw2VJrheUUhArDAoq58S6YrS3g OKEx (exchange)	0x689c56aef474df92d44a1b70850f808488f9769c KuCoin (exchange)
9	17kb7c9ndg7ioSuzMWEHWECdEVUegNkcGc	0x86fa049857e0209aa7d9e616f7eb3b3b78ecfdb0 EOS (token contract)
10	3CD1QW6fjgTwKq3Pj97nty28WZAVkziNom The Shadow Brokers (hacker group [81])	0x0d8775f648430679a709e98d2b0cb6250d2887ef BAT (token contract)

Table 6.4. Total number and percentage of distinct addresses at the tail/head of incoming/outcoming transactions to/from k most important addresses.

Ranking Range	**No. of Addresses**		**Percentage of Addresses**	
	Bitcoin	**Ethereum**	**Bitcoin**	**Ethereum**
	Incoming Transactions			
1	442 218	2 153 020	0.05	2.73
1-10	8 069 235	12 308 446	1.01	15.59
1-100	18 406 620	24 623 414	2.30	31.19
1-1000	56 795 882	38 110 110	7.10	48.27
1-10000	105 329 533	47 182 270	13.17	59.76
	Outcoming Transactions			
1	1 012 509	1 524 220	0.13	1.93
1-10	12 315 379	1 689 862	1.54	2.14
1-100	21 608 153	5 234 996	2.70	6.63
1-1000	43 235 102	15 698 561	5.40	19.88
1-10000	63 023 109	24 008 947	7.88	30.41

The degree distribution of Ethereum transaction graph nodes is illustrated in Figure 6.3. The majority of Ethereum addresses (about 50 million) created less than three transactions. About 21 million addresses created 3 to 10 transactions. 36049 addresses have created more than 1000 transactions. The indegree distribution follows a similar distribution to outdegree distribution, except that the indegree distribution is more concentrated around one and two.

Figure 6.4 shows the numbers of transfer transactions of ten major ERC20 tokens. USDT token has the largest transfer transaction count among our 40 major ERC20 token list with about 33 million transfer transactions. OMG takes second place and has around 1.8 million transfer transactions. USDC is the third token with about 1.6 million transactions.

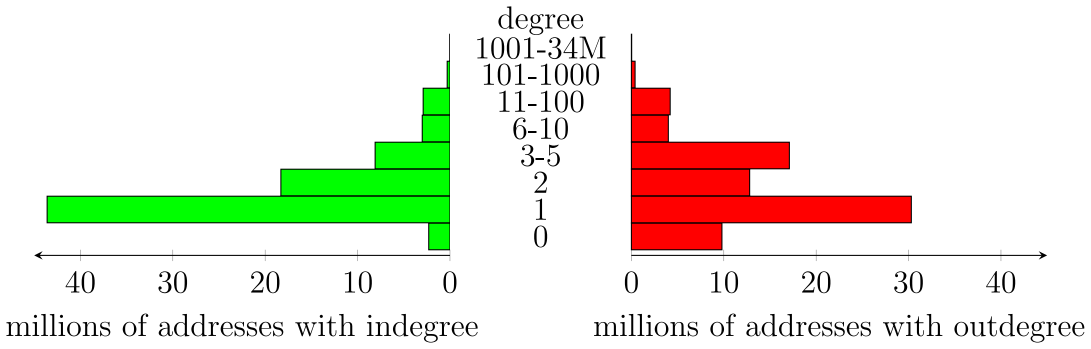


Figure 6.3. Degree distributions of Ethereum transaction graph nodes.

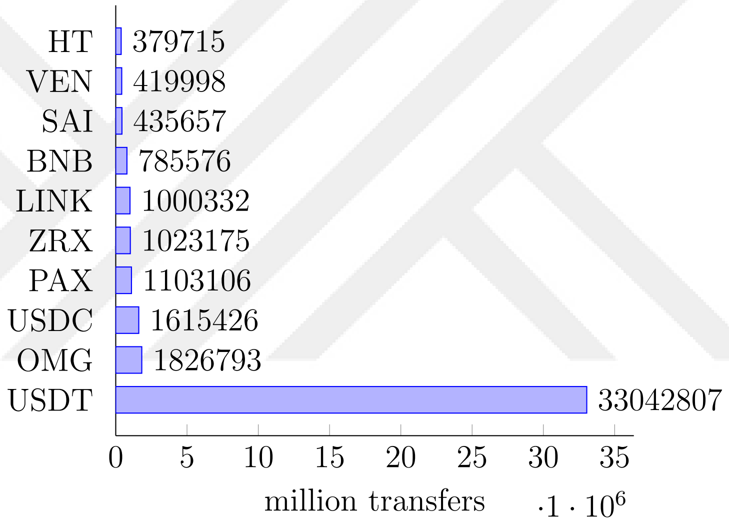


Figure 6.4. Number of transfer transactions of ten major ERC20 tokens.

Figure 6.5 shows the timing results for shortest path based blacklisted address trace forest. Figure 6.6 shows (a) a forest of trees, whose roots are blacklisted blockchain addresses, computed to depth 5 from Ethereum transaction graph and an example tree trace to a blacklisted address of the DragonEx hacker [1] and (b) transaction details of one path on the DragonEx hacker trace tree.

Figure 6.7 shows two example trace subgraphs: (a) for the DragonEx Hacker [1] on the Ethereum blockchain and (b) for the July 15 Twitter Hack [2] on the Bitcoin blockchain. These subgraphs are visualized from the results of our test T9. In Figure 6.7(a), the stolen ethers first go through several addresses, then split into four addresses, and finally are consolidated in a single address (our query address). Since the blockchain addresses are pseudoanonymous, we cannot be one hundred percent sure whether the final address belongs to the hacker or not. In Figure 6.7(b), the stolen bitcoins are transferred from 3 hacker addresses to an address.

Table 6.5 shows the timing results of the tests.

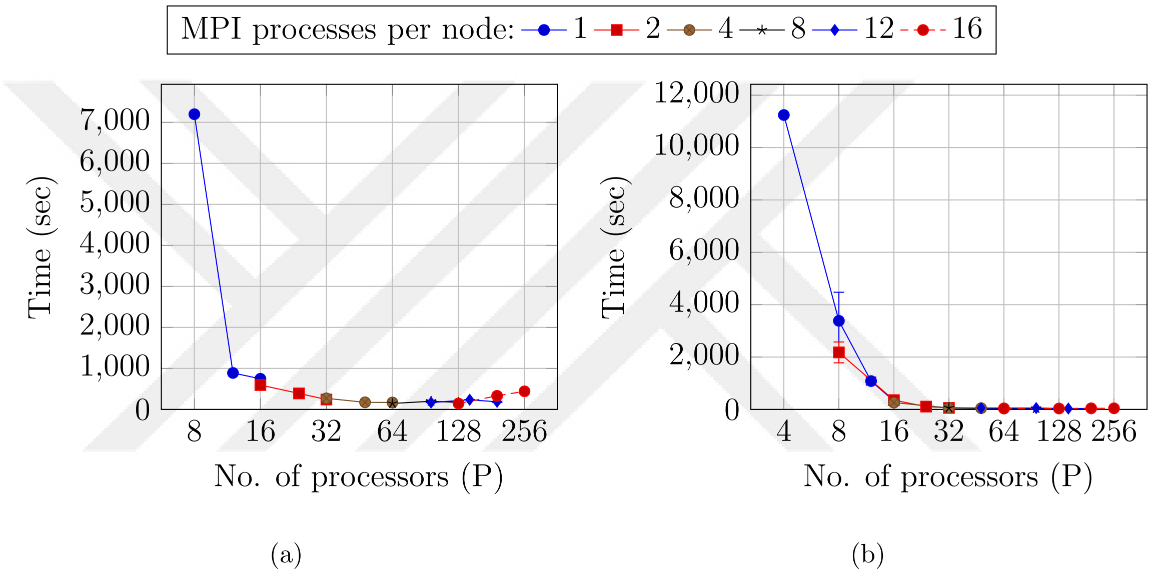


Figure 6.5. Test T7: Shortest path based blacklisted address trace forest times for (a) Bitcoin and (b) Ethereum on the HPC cluster.

Table 6.5. Timing results of tests in seconds.

	Bitcoin			**Ethereum**			
	1 MPI process per node						
Test	**P=8**	**P=12**	**P=16**	**P=4**	**P=8**	**P=12**	**P=16**
T1	4204	3119	2693	1314	748	509	443
T2	-[1]	-[1]	-[1]	-[1]	5768	6056	5718
T3	9711	5897	4687	991	622	370	372
T4	88	76	60	14.5	10.8	8.3	7.5

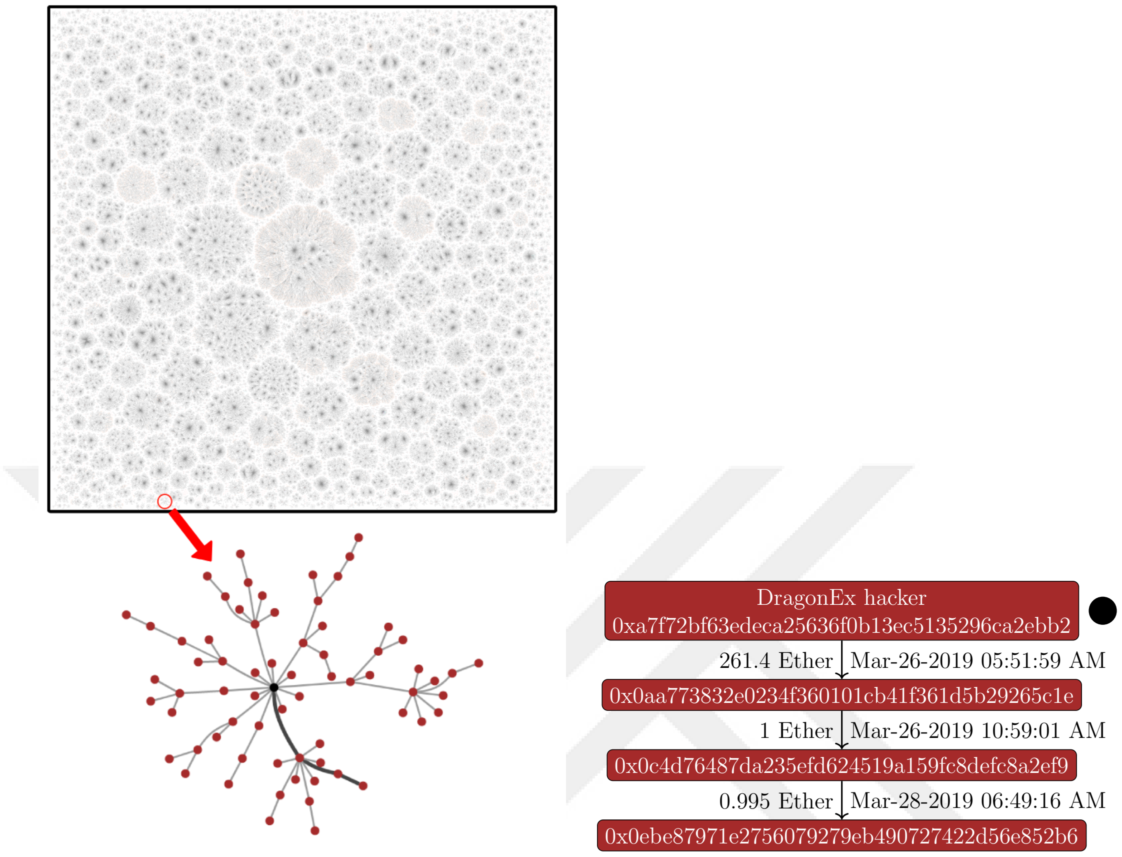
[1]out of memory

Table 6.5. Timing results of tests in seconds. (cont.)

T5	N/A	N/A	N/A	149	85.6	53.4	45.1
T6	16648	10533	8289	1319	855	580	491
T7	7198	889	747	11243	3383	1082	323
T8	67	44	33	116	70.5	39.9	34.1
T9	5	3.4	2.6	4.0	1.9	1.4	1.2
	2 MPI processes per node						
	P=16	**P=24**	**P=32**	**P=8**	**P=16**	**P=24**	**P=32**
T1	2868	2563	2127	712	430	331	297
T3	4403	2913	2139	566	369	276	247
T4	77	78	69	10.2	8.2	7.1	6.8
T5	N/A	N/A	N/A	81	45.3	26.8	20.8
T6	9065	5720	4994	841	487	355	329
T7	594	390	241	2178	355	108	58.2
T8	32	22	16	66.4	32.3	26.3	22.3
T9	2.6	2.1	1.7	1.9	1.1	0.9	0.8
	4 MPI processes per node						
	P=32	**P=48**	**P=64**	**P=16**	**P=32**	**P=48**	**P=64**
T1	2557	2198	1909	480	327	286	256
T3	2247	1502	1069	401	232	206	167
T4	72	71	66	8.3	7.1	11.1	11.0
T5	N/A	N/A	N/A	43	20.7	33.8	21.3
T6	5191	3570	2885	505	309	254	219
T7	270	173	169	260	58.2	46.1	33.5
T8	16	11	9	34.2	21.1	18.7	14.3
T9	1.7	1.2	0.9	1.2	0.7	0.6	0.5
	8 MPI processes per node						
	P=64	**P=96**	**P=128**	**P=32**	**P=64**	**P=96**	**P=128**
T1	2505	2433	2163	401	287	227	218

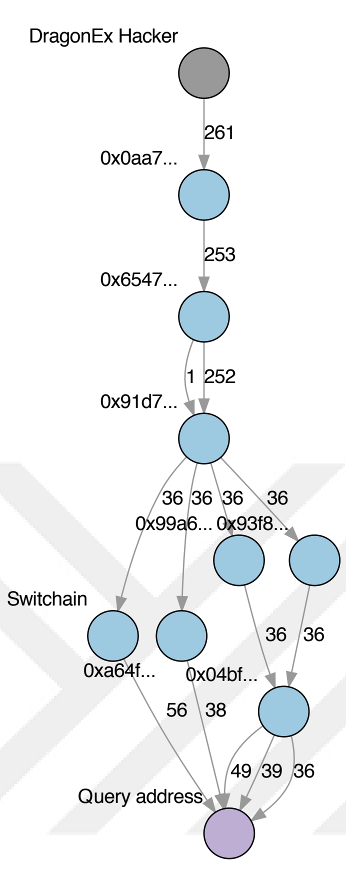
Table 6.5. Timing results of tests in seconds. (cont.)

T3	1143	876	605	252	163	150	156
T4	67	74	68	7.1	10.5	14.5	18.3
T5	N/A	N/A	N/A	20.8	20.2	19.8	19.3
T6	3207	2502	2149	324	224	197	180
T7	145	199	172	57.5	37.2	37.6	32.4
T8	9	6	5	22.3	14.2	12.7	12.7
T9	2.1	0.8	0.7	0.8	0.5	0.5	0.5
	12 MPI processes per node						
	P=96	**P=144**	**P=192**	**P=48**	**P=96**	**P=144**	**P=192**
T1	2684	2874	2675	506	303	251	234
T3	898	745	512	213	212	211	219
T4	85	91	84	14.2	15.9	25.6	35.5
T5	N/A	N/A	N/A	44.4	30.9	25.7	25.3
T6	2855	2395	2037	305	229	177	186
T7	168	236	185	53.7	51.9	33.0	37.6
T8	8	6	4	24.8	15.9	16.2	14.9
T9	1.0	7.7	9.1	0.8	0.6	0.7	1.5
	16 MPI processes per node						
	P=128	**P=192**	**P=256**	**P=64**	**P=128**	**P=192**	**P=256**
T1	2920	3716	3619	426	352	331	331
T3	723	602	445	247	219	209	224
T4	87	96	95	14.2	24.6	38.3	55.0
T5	N/A	N/A	N/A	29.5	28.2	27.7	31.6
T6	2599	2159	1984	284	216	188	184
T7	144	332	441	39.0	34.9	42.3	42.7
T8	7	5	4	20.0	17.1	14.8	16.1
T9	3.5	6.3	0.8	0.9	0.6	0.5	1.5

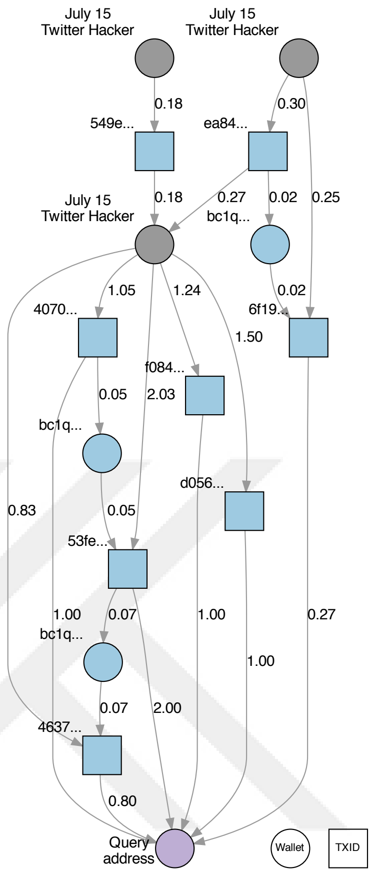

DragonEx hacker
0xa7f72bf63edeca25636f0b13ec5135296ca2ebb2
261.4 Ether
Mar-26-2019 05:51:59 AM
0x0aa773832e0234f360101cb41f361d5b29265c1e
1 Ether
Mar-26-2019 10:59:01 AM
0x0c4d76487da235efd624519a159fc8defc8a2ef9
0.995 Ether
Mar-28-2019 06:49:16 AM
0x0ebe87971e2756079279eb490727422d56e852b6

(a) (b)

Figure 6.6. (a) Blacklisted addresses trace forest for Ethereum data and example tree trace to the blacklisted address of the DragonEx hacker [1] and (b) transaction details of one path on the DragonEx hacker trace tree.



(a)



(b)

Figure 6.7. Trace subgraphs of (a) DragonEx hacker (Ethereum) [1] and (b) July 15 Twitter Hack (Bitcoin) [2].

7. CONCLUSION

In this thesis, we propose a distributed, parallel, and scalable blockchain transaction graph system for analyzing transaction graphs. The system is written in the C++ programming language using the MPI library and supports both account-based and UTXO-based tokens. The system constructs the directed, transpose, and undirected transaction graphs in a parallel and distributed fashion from the parsed blockchain transactions stored as plain text files. We were able to construct these graphs for a 5-year Ethereum transaction data in less than 4 minutes and a 12-year Bitcoin data in less than 32 minutes. After the graph construction, the transaction graphs are input to graph algorithms and analyzed.

We developed distributed and parallel blacklisted address trace forest and transaction tracing algorithms to analyze blockchain transactions. In addition, we implemented Pagerank, connected component calculation, degree distribution, token count calculation algorithms to analyze the transaction graph. We also extracted blockchain address features, such as in/outdegree, unique in/outdegree, in/out average transaction amount. To parallel partition the graph, we used ParMETIS. Graph partitioning was not useful in our case since it gave out of memory error for our Bitcoin data and took a longer time than the total time of our other tests for Ethereum data.

We extracted and parsed bitcoin transactions from Bitcoin blockchain; ether and popular ERC20 token transactions from Ethereum blockchain. For transaction tracing, we collected blacklisted addresses for Bitcoin and Ethereum from public scam databases, various blog posts, previously published academic papers, and a government agency website.

We benchmarked our transaction graph system on a 16 node cluster on Amazon cloud using different graph algorithms and reported the timings obtained for these algorithms. We presented our graph analysis results, such as top 10 pageranked addresses, the degree distribution of addresses, the number of transfer transactions for ERC20 tokens, trace forest visualization.

7.1. Future Work

- A future research topic may be the parallel partitioning of blockchain transaction graphs. Partitioning will be important since the transaction throughput of blockchains is expected to rise. Although well-established partitioners like ParMETIS exist, transaction graph partitioning takes a long time and consumes much memory. Since some users use the blockchain addresses (wallets) in a disposable manner, a heuristic may be developed that excludes the inactive addresses, i.e., nodes of the transaction graph, from the graph partitioning calculations. Working on a subset of nodes will speed up the calculations. In addition, prioritizing the speed over partitioning quality may be helpful because there will always be new transactions in the blockchain, and the transaction graph will be updated or reconstructed regularly.
- Another future research topic may be the analysis of blockchain transaction graphs from a graph motiv perspective.

REFERENCES

1. Khatri, Y., *Singapore-Based Crypto Exchange DragonEx Has Been Hacked*, 2019, https://www.coindesk.com/markets/2019/03/26/singapore-based-crypto-exchange-dragonex-has-been-hacked/, accessed in November 2021.

2. Wikipedia contributors, *2020 Twitter Account Hijacking*, 2021, https://en.wikipedia.org/w/index.php?title=2020_Twitter_account_hijacking&oldid=1021898024, accessed 18 June 2021.

3. Dhamodharan, R., *Why Mastercard is Bringing Crypto onto Its Network*, 2021, https://www.mastercard.com/news/perspectives/2021/why-mastercard-is-bringing-crypto-onto-our-network/, accessed in September 2021.

4. Visa, *Digital Currency Comes to Visa's Settlement Platform*, 2021, https://usa.visa.com/visa-everywhere/blog/bdp/2021/03/26/digital-currency-comes-1616782388876.html, accessed in September 2021.

5. Hunter, T., *Press Release: PayPal Launches Checkout with Crypto*, 2021, https://newsroom.paypal-corp.com/2021-03-30-PayPal-Launches-Checkout-with-Crypto, accessed in September 2021.

6. Post, L. and B. Cozine, *Mastercard and Bakkt Partner to Offer Innovative Crypto and Loyalty Solutions*, 2021, https://www.mastercard.com/news/press/2021/october/mastercard-and-bakkt-partner-to-offer-innovative-crypto-and-loyalty-solutions/, accessed in October 2021.

7. Pérez, S. and C. Ostroff, *El Salvador Becomes First Country to Adopt Bitcoin as National Currency - WSJ*, 2021, https://www.wsj.com/articles/bitcoin-comes-to-el-salvador-first-country-to-adopt-crypto-as-national-currency-11631005200, accessed in September 2021.

8. Christodorescu, M., E. English, W. C. Gu, D. Kreissman, R. Kumaresan, M. Minaei, S. Raghuraman, C. Sheffield, A. Wijeyekoon and M. Zamani, "Universal Payment Channels: An Interoperability Platform for Digital Currencies", *arXiv preprint arXiv:2109.12194*, 2021.

9. Levy, A., *Coinbase Stock Debuts on Nasdaq in Direct Listing*, 2021, `https://www.cnbc.com/2021/04/14/coinbase-to-debut-on-nasdaq-in-direct-listing.html`, accessed in September 2021.

10. Financial Action Task Force, *Guidance for a Risk-Based Approach to Virtual Assets and Virtual Asset Service Providers*, 2019, `https://www.fatf-gafi.org/publications/fatfrecommendations/documents/guidance-rba-virtual-assets.html`, accessed in October 2021.

11. U.S. Department of the Treasury, *Specially Designated Nationals List - Data Formats & Data Schemas*, 2021, `https://home.treasury.gov/policy-issues/financial-sanctions/specially-designated-nationals-list-data-formats-data-schemas`, accessed in May 2021.

12. Bing, C., *Exclusive: U.S. to Give Ransomware Hacks Similar Priority as Terrorism*, 2021, `https://www.reuters.com/technology/exclusive-us-give-ransomware-hacks-similar-priority-terrorism-official-says-2021-06-03/`, accessed in October 2021.

13. U.S. Department of Justice, *Department of Justice Seizes 2.3 Million in Cryptocurrency Paid to the Ransomware Extortionists Darkside*, 2021, `https://www.justice.gov/opa/pr/department-justice-seizes-23-million-cryptocurrency-paid-ransomware-extortionists-darkside`, accessed in October 2021.

14. Poly Network, *Important Notice: We Are Sorry to Announce that #PolyNetwork Was Attacked on @BinanceChain @Ethereum and*

@0xPolygon Assets Had Been Transferred to Hacker's Following Addresses: ETH: 0xC8a65Fadf0e0dDAf421F28FEAb69Bf6E2E589963 BSC: 0x0D6e286A7cfD25E0c01fEe9756765D8033B32C71, 2021, https://twitter.com/PolyNetwork2/status/1425073987164381196, accessed in October 2021.

15. Zaknun, C., *DAO Maker Statement*, 2021, https://medium.com/daomaker/dao-maker-statement-thursday-12th-of-august-2c3bb0d1bb69, accessed in October 2021.

16. Liquid Global, *Important Notice: We Are Sorry to Announce that #LiquidGlobal Warm Wallets Were Compromised, We Are Moving Assets Into the Cold Wallet. We Are Currently Investigating and Will Provide Regular Updates. In the Meantime Deposits and Withdrawals Will Be Suspended.*, 2021, https://twitter.com/Liquid_Global/status/1428176357515612165, accessed in October 2021.

17. DiCamillo, N., *Coinbase Multi-Factor Authentication Hack Affects at Least 6,000 Customers*, 2021, https://www.coindesk.com/business/2021/10/01/coinbase-multi-factor-authentication-hack-affects-at-least-6000-customers/, accessed in October 2021.

18. pNetwork DeFi, *1/N We're Sorry to Inform the Community that An Attacker Was Able to Leverage A Bug In Our Codebase and Attack PBTC on BSC, Stealing 277 BTC (Most of Its Collateral). The Other Bridges Were Not Affected. All Other Funds in the PNetwork Are Safe.*, 2021, https://twitter.com/pNetworkDeFi/status/1439690593211490324, accessed in October 2021.

19. Androulaki, E., A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains", *Proceedings of the Thirteenth EuroSys Conference*, pp. 1–15, ACM, 2018.

20. Buterin, V., *ETH2 Scaling for Data Will Be Available Before ETH2 Scaling for*

General Computation. This Implies that Rollups Will Be the Dominant Scaling Paradigm for at Least A Couple of Years: First ∼2-3k TPS with Eth1 as Data Layer, then ∼100k TPS With Eth2 (Phase 1). Adjust Accordingly., 2020, `https://twitter.com/VitalikButerin/status/1277961594958471168`, accessed in October 2021.

21. Chakravarty, M. M., S. Coretti, M. Fitzi, P. Gazi, P. Kant, A. Kiayias and A. Russell, “Hydra: Fast Isomorphic State Channels”, *International Association for Cryptologic Research Cryptology ePrint Archive*, Vol. 2020, p. 299, 2020.

22. Cusce, C., *The Avalanche Platform — A Tech Primer*, 2020, `https://medium.com/avalancheavax/the-ava-platform-a-tech-primer-7a9b5de57a35`, accessed in October 2021.

23. Kılıç, B., C. Özturan and A. Sen, “A Cluster Based System for Analyzing Ethereum Blockchain Transaction Data”, *2020 Second International Conference on Blockchain Computing and Applications (BCCA)*, pp. 59–65, IEEE, 2020.

24. Kılıç, B., C. Özturan and A. Sen, “Parallel Analysis of Ethereum Blockchain Transaction Data Using Cluster Computing”, *Cluster Computing*, Jan. 2022.

25. Kılıç, B., C. Özturan and A. Sen, “Analyzing Large Scale Blockchain Transaction Graphs for Fraudulent Activities”, J. Soldatos and D. Kyriazis (Editors), *Bigdata and Artificial Intelligence in Digital Finance*, chap. 14, Springer, 2022.

26. Vogelsteller, F. and V. Buterin, *EIP 20: Token Standard*, 2015, `https://eips.ethereum.org/EIPS/eip-20`, accessed in October 2021.

27. Message Passing Interface Forum, *MPI Documents*, 2021, `https://www.mpi-forum.org/docs/`, accessed in October 2021.

28. Entriken, W., D. Shirley, J. Evans and N. Sachs, *EIP-721: Non-Fungible Token Standard*, 2018, `https://eips.ethereum.org/EIPS/eip-721`, accessed in

November 2021.

29. Radomski, W., A. Cooke, P. Castonguay, J. Therien, E. Binet and R. Sandford, *EIP-1155: Multi Token Standard*, 2018, `https://eips.ethereum.org/EIPS/eip-1155`, accessed in November 2021.

30. Reid, F. and M. Harrigan, "An Analysis of Anonymity in the Bitcoin System", Y. Altshuler, Y. Elovici, A. B. Cremers, N. Aharony and A. Pentland (Editors), *Security and Privacy in Social Networks*, pp. 197–223, Springer New York, New York, 2013.

31. Ron, D. and A. Shamir, "Quantitative Analysis of the Full Bitcoin Transaction Graph", A.-R. Sadeghi (Editor), *International Conference on Financial Cryptography and Data Security*, pp. 6–24, Springer, Berlin Heidelberg, 2013.

32. Cormen, T. H., C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to algorithms*, MIT press, London, 2nd edn., 2001.

33. Kalodner, H., M. Möser, K. Lee, S. Goldfeder, M. Plattner, A. Chator and A. Narayanan, "Blocksci: Design and Applications of a Blockchain Analysis Platform", *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2721–2738, 2020.

34. Kalodner, H., M. Möser, K. Lee, S. Goldfeder, M. Plattner, A. Chator and A. Narayanan, *citp/BlockSci: A High-performance tool for Blockchain Science and Exploration*, 2020, `https://github.com/citp/BlockSci`, accessed in February 2021.

35. Guo, D., J. Dong and K. Wang, "Graph Structure and Statistical Properties of Ethereum Transaction Relationships", *Information Sciences*, Vol. 492, pp. 58–71, 2019.

36. Chan, W. and A. Olmsted, "Ethereum Transaction Graph Analysis", *2017 12th*

International Conference for Internet Technology and Secured Transactions (IC-ITST), pp. 498–500, IEEE, 2017.

37. Anoaica, A. and H. Levard, “Quantitative Description of Internal Activity on the Ethereum Public Blockchain”, *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, IEEE, 2018.

38. Chen, T., Z. Li, Y. Zhu, J. Chen, X. Luo, J. C.-S. Lui, X. Lin and X. Zhang, “Understanding Ethereum via Graph Analysis”, *ACM Transactions on Internet Technology (TOIT)*, Vol. 20, No. 2, pp. 1–32, 2020.

39. Victor, F. and B. K. Lüders, “Measuring Ethereum-based Erc20 Token Networks”, I. Goldberg and T. Moore (Editors), *International Conference on Financial Cryptography and Data Security*, pp. 113–129, Springer, 2019.

40. Somin, S., G. Gordon and Y. Altshuler, “Network Analysis of Erc20 Tokens Trading on Ethereum Blockchain”, A. J. Morales, C. Gershenson, D. Braha, A. A. Minai and Y. Bar-Yam (Editors), *International Conference on Complex Systems*, pp. 439–450, Springer International Publishing, 2018.

41. Somin, S., G. Gordon, A. Pentland, E. Shmueli and Y. Altshuler, “ERC20 Transactions over Ethereum Blockchain: Network Analysis and Predictions”, *arXiv preprint arXiv:2004.08201*, 2020.

42. Zheng, P., Z. Zheng, J. Wu and H.-N. Dai, “XBlock-ETH: Extracting and Exploring Blockchain Data from Ethereum”, *IEEE Open Journal of the Computer Society*, Vol. 1, pp. 95–106, 2020.

43. Zheng, P., *XBlock-ETH Download*, 2021, `http://xblock.pro/xblock-eth.html`, accessed in February 2021.

44. CoinMarketCap, *Cryptocurrency Prices, Charts And Market Capitalizations*, 2021, `https://coinmarketcap.com/`, accessed in October 2021.

45. Etherscan, *Ethereum Full Node Sync (Archive) Chart*, 2021, `https://etherscan.io/chartsync/chainarchive`, accessed in February 2021.

46. Infura, *Ethereum API — IPFS API & Gateway — ETH Nodes as a Service*, 2021, `https://infura.io`, accessed in February 2021.

47. Cloudflare, *Ethereum Gateway — Cloudflare Distributed Web Gateway docs*, 2021, `https://developers.cloudflare.com/distributed-web/ethereum-gateway`, accessed in February 2021.

48. GetBlock.io, *GetBlock - Blockchain Nodes Provider. Get Access to Full Nodes with Us*, 2021, `https://getblock.io`, accessed in February 2021.

49. Day, A., E. Medvedev, N. AK and W. Price, *Introducing Six New Cryptocurrencies in BigQuery Public Datasets—and How to Analyze Them — Google Cloud Blog*, Feb. 2019, `https://cloud.google.com/blog/products/data-analytics/introducing-six-new-cryptocurrencies-in-bigquery-public-datasets-and-how-to-analyze-them`, accessed in February 2021.

50. Kaggle, *Elliptic Data Set*, 2019, `https://www.kaggle.com/ellipticco/elliptic-data-set`, accessed in February 2021.

51. Blockchain ETL, *Blockchain ETL*, 2021, `https://github.com/blockchain-etl`, accessed in February 2021.

52. Özturan, C., A. Şen and B. Kılıç, *Transaction Graph Dataset for the Ethereum Blockchain*, 2021, `https://doi.org/10.5281/zenodo.4718440`, accessed in October 2021.

53. Blockchain Explorer, *Transaction: 26f7dc6a448ffdea187740c93afd420e325c1cc0f01d17929081c374c19dd0a3*, 2020, `https://www.blockchain.com/btc/tx/26f7dc6a448ffdea187740c93afd420e325c1cc0f01d17929081c374c19dd0a3`, accessed in November 2021.

54. Bitcoin Project, *RPC API Reference - Bitcoin*, 2021, `https://developer.bitcoin.org/reference/rpc/`, accessed in March 2021.

55. Ethereum Wiki, *json-rpc*, 2021, `https://eth.wiki/json-rpc/API`, accessed in March 2021.

56. Beregszaszi, A., *EIP-609: Hardfork Meta: Byzantium*, 2017, `https://eips.ethereum.org/EIPS/eip-609`, accessed in March 2021.

57. Go Ethereum Authors, *debug Namespace*, 2021, `https://geth.ethereum.org/docs/rpc/ns-debug#debug_tracetransaction`, accessed in March 2021.

58. Go Ethereum, *go-ethereum/consensus.go at master*, 2021, `https://github.com/ethereum/go-ethereum/blob/master/consensus/ethash/consensus.go`, accessed in March 2021.

59. Aleth Authors, *aleth/mainNetwork.cpp at master — ethereum/aleth*, 2021, `https://github.com/ethereum/aleth/blob/master/libethashseal/genesis/mainNetwork.cpp`, accessed in March 2021.

60. Go Ethereum Authors, *go-ethereum/genesis_alloc.go at master — ethereum/go-ethereum*, 2021, `https://github.com/ethereum/go-ethereum/blob/master/core/genesis\_alloc.go`, accessed in March 2021.

61. Bitcoin Who's Who Blog, *The Most Frequently Reported Bitcoin Scams Of 2020*, 2021, `https://www.bitcoinwhoswho.com/blog/2021/01/01/the-most-frequently-reported-scam-bitcoin-addresses-of-2020/`, accessed in February 2021.

62. Spiro, J., *Chainalysis Blog — New OFAC Sanctions and DOJ Complaint for North Korea-Linked Cryptocurrency Laundering Scheme: What You Need to Know*, 2020, `https://blog.chainalysis.com/reports/north-korea-cryptocurrency-addresses-ofac-doj-march-2020`, accessed in February 2021.

63. Team, C., *Chainalysis Blog — Alt-Right Groups and Personalities Involved In the January 2021 Capitol Riot Received Over $500K In Bitcoin From French Donor One Month Prior*, Jan. 2021, `https://blog.chainalysis.com/reports/capitol-riot-bitcoin-donation-alt-right-domestic-extremism`, accessed in February 2021.

64. Team, C., *Chainalysis Blog — U.S. Government Targets Russian Influence Operations with Cryptocurrency Nexus*, 2021, `https://blog.chainalysis.com/reports/ofac-fbi-russian-influence-synthetic-identity-document-vendor-sanctions`, accessed in April 2021.

65. Team, C., *Chainalysis Blog — The Twitter Hack: What We Know One Week Later*, 2020, `https://blog.chainalysis.com/reports/twitter-hack-july-2020-update`, accessed in May 2021.

66. Masarah, P.-C., R. Matteo, H. Bernhard and C. Tomas, *Spams Meet Cryptocurrencies: Sextortion in the Bitcoin Ecosystem*, 2019, `https://doi.org/10.5281/zenodo.3515199`.

67. Xia, P., H. Wang, X. Luo, L. Wu, Y. Zhou, G. Bai, G. Xu, G. Huang and X. Liu, *COVID-19 Themed Cryptocurrency Scams*, 2020, `https://covid19scam.github.io/`, accessed in February 2021.

68. Clegg, P., *Tracing Ransomware: CipherTrace Helps McAfee Follow NetWalker Funds - CipherTrace*, 2020, `https://ciphertrace.com/tracing-ransomware-ciphertrace-helps-mcafee-follow-netwalker-funds/`, accessed in February 2021.

69. Paquet-Clouston, M., B. Haslhofer and B. Dupont, *ransomware-dataset/seed_addresses.csv at master · behas/ransomware-dataset*, 2018, `https://github.com/behas/ransomware-dataset/blob/master/data/seed_addresses.csv`, accessed in February 2021.

70. Paquet-Clouston, M., B. Haslhofer and B. Dupont, “Ransomware Payments in the Bitcoin Ecosystem”, *Journal of Cybersecurity*, Vol. 5, No. 1, 05 2019.

71. CryptoScamDB, *Home*, 2021, `https://cryptoscamdb.org`, accessed in February 2021.

72. MyEtherWallet, *ethereum-lists/addresses-darklist.json at e3e535079baf00533b9f9bd76d0a89b968fe0930 · MyEtherWallet/ethereum-lists*, 2020, `https://github.com/MyEtherWallet/ethereum-lists/blob/e3e535079baf00533b9f9bd76d0a89b968fe0930/src/addresses/addresses-darklist.json`, accessed in October 2020.

73. Etherscan, *Label Word Cloud*, 2021, `https://etherscan.io/labelcloud`, accessed in February 2021.

74. Castonguay, P., *USDC Banned Addresses*, 2020, `https://dune.xyz/phabc/usdc-banned-addresses`, accessed in October 2020.

75. Karypis, G. and K. Schloegel, *ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library Version 4.0*, University of Minnesota, Department of Computer Science and Engineering, Minneapolis, MN 55455, Mar. 2013.

76. Shi, H. and J. Schaeffer, “Parallel Sorting by Regular Sampling”, *Journal of Parallel and Distributed Computing*, Vol. 14, No. 4, pp. 361–372, 1992.

77. Chevalier, C. and F. Pellegrini, “PT-Scotch: A Tool for Efficient Parallel Graph Ordering”, *Parallel Computing*, Vol. 34, No. 6-8, pp. 318–331, 2008.

78. Zhang, Y., A. Azad and Z. Hu, “FastSV: A Distributed-memory Connected Component Algorithm with Fast Convergence”, *Proceedings of the 2020 Society for Industrial and Applied Mathematics Conference on Parallel Processing for Scientific Computing*, pp. 46–57, Society for Industrial and Applied Mathematics, 2020.

79. Massachusetts Institute of Technology, *StarCluster*, 2013, `http://star.mit.edu/cluster/`, accessed in November 2021.

80. Harper, C., *How the PlusToken Scam Absconded With Over 1 Percent of the Bitcoin Supply*, 2019, `https://bitcoinmagazine.com/business/how-the-plustoken-scam-absconded-with-over-1-percent-of-the-bitcoin-supply`, accessed in November 2021.

81. Lee, S., C. Yoon, H. Kang, Y. Kim, Y. Kim, D. Han, S. Son and S. Shin, "Cybercriminal Minds: An Investigative Study of Cryptocurrency Abuses in the Dark Web", *Network & Distributed System Security Symposium*, pp. 1–15, Internet Society, 2019.

APPENDIX A: CODE SNIPPETS

```
SELECT ‘hash‘,
       block_number,
       ARRAY(SELECT STRUCT(index,addresses,value) FROM t.inputs) as inputs,
       ARRAY(SELECT STRUCT(index,addresses,value) FROM t.outputs) as outputs
FROM ‘bigquery-public-data.crypto_bitcoin.transactions‘ as t
```

Figure A.1. SQL code to query Bitcoin transactions from Google BigQuery.

```
SELECT DISTINCT block_number, transaction_index
FROM ‘bigquery-public-data.crypto_ethereum.traces‘
WHERE block_timestamp < "2017-10-18" AND block_number < 4370000 AND status = 0
```

Figure A.2. SQL code to query the block number and transaction index of failed Ethereum transactions before Byzantium fork from Google BigQuery.

```
SELECT  block_number, transaction_index
FROM ‘bigquery-public-data.crypto_ethereum.transactions‘
WHERE block_timestamp > "2017-10-16" AND
      block_number >= 4370000 AND
      receipt_status = 0
```

Figure A.3. SQL code to query the block number and transaction index of failed Ethereum transactions after Byzantium fork from Google BigQuery.

APPENDIX B: COPYRIGHT NOTICES

The articles [24,25] are open access articles distributed under the terms of the Creative Commons CC BY license (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The images that emerged within the scope of this thesis study and whose copyrights were transferred to the publishing house were used in the thesis book in accordance with the “editorial policy on the reuse of the author’s own writing and graphics” on the website of the publisher.