Cluster Computing (2022) 25:1885–1898
https://doi.org/10.1007/s10586-021-03511-0

Parallel analysis of Ethereum blockchain transaction data using cluster computing

Baran Kılıç[1] · Can Özturan[1] · Alper Sen[1]

Received: 28 April 2021 / Revised: 2 November 2021 / Accepted: 6 December 2021 / Published online: 4 January 2022

Abstract
Ability to perform fast analysis on massive public blockchain transaction data is needed in various applications such as tracing fraudulent financial transactions. The blockchain data is continuously growing and is organized as a sequence of blocks containing transactions. This organization, however, cannot be used for parallel graph algorithms which need efficient distributed graph data structures. Using message passing libraries (MPI), we develop a scalable cluster-based system that constructs a distributed transaction graph in parallel and implement various transaction analysis algorithms. We report performance results from our system operating on roughly 5 years of 10.2 million block Ethereum Mainnet blockchain data. We report timings obtained from tests involving distributed transaction graph construction, partitioning, page ranking of addresses, degree distribution, token transaction counting, connected components finding and our new parallel blacklisted address trace forest computation algorithm on a 16 node economical cluster set up on the Amazon cloud. Our system is able to construct a distributed graph of 766 million transactions in 218 s and compute the forest of blacklisted address traces in 32 s.

Keywords Parallel · Cluster · Blockchain · Graph · Transaction

1 Introduction

Recently, public blockchain platforms that operate autonomously under the control of no one have become popular globally. Blockchains can be used to generate and keep securely ownership records of cryptocurrencies and tokenized assets. Tokens can be used to represent valuable assets such as company shares, tickets, governance privileges, stable coins that represent national currencies (such as the USD and the EURO) and various resources. Recently blockchain-based fraudulent incidents such as ransomware and theft of crypto assets have increased [1, 2]. Also, since the public blockchain networks span the globe, they can also be used to transfer assets among different jurisdictions making it possible to evade the regulations. As a result, a system that performs fast tracing fraudulent activities on massive public blockchain transaction data is needed in the field of finance. This need has also led to the emergence of firms such as the Chainalysis [3] that is highly valued or the CipherTrace that has recently been acquired [4]. Financial Action Task Force (FATF) is an inter-governmental body that publishes recommendations for combatting global money laundering and terrorist financing activities. FATF has recently prepared guidance [5] that calls for monitoring of virtual assets. Since blockchain transaction throughputs are improving, massive transaction data will be accumulating. All these developments provide evidence that scalable and parallel systems will be needed that can analyze big blockchain graph transaction data in the near future. This is the problem that is addressed in this paper.

In this work, we focus on the Ethereum Mainnet blockchain, which also supports the execution of smart

This work was carried out as part of the Infinitech Project which is supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 856632.

✉ Can Özturan
ozturaca@boun.edu.tr

Baran Kılıç
baran.kilic@boun.edu.tr

Alper Sen
alper.sen@boun.edu.tr

[1] Bogazici University, Istanbul, Turkey

contracts. Ethereum Mainnet is the most popular blockchain supporting smart contracts and hosting billions of dollars worth of token contracts. Standardized ERC20 smart contracts are now used to deploy tokens (stable coins) representing fiat currencies in a 1-to-1 manner (for example, 1 token equaling 1 EUR or 1 USD) by finance companies. Gemini USD (GUSD), Tether USD (USDT), Tether Gold (XAUT), Statis Euro (EURS), and Turkish BiLira (TRYB) are examples of tokens currently available on the Ethereum blockchain. Note that these 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 stable coins and stable coins can, in turn, be redeemed as fiat money in the traditional finance ecosystem. Therefore, assets that are acquired fraudulently can go through various transfers and exchanges on the blockchain and end up as fiat money in different countries. It is possible that a business accepts crypto assets that can be traced to addresses involved in fraudulent activities. Holding crypto assets that can be traced to fraudulent or sanctioned addresses can be risky for businesses. As a result, a transaction graph processing system can help us to detect such cases and take appropriate actions. Our work aims to provide a scalable distributed transaction graph processing system that will enable fast graph operations to be performed on the massive blockchain transaction data.

The blockchain data that is synced as a node is accessible as a sequence of blocks containing transactions. This way of accessing transaction data, however, is too slow for applications that require a transaction graph to be constructed. Since blockchain data is continuously growing and since future blockchain versions will support hundreds to thousands of transactions per second (tps), a scalable transaction graph system is needed. We contribute:

- A cluster-based system that constructs a distributed transaction graph in parallel out of raw transaction data that comes from the blockchain.
- A parallel algorithm that computes all shortest path based transaction traces to fraud-related blacklisted blockchain addresses.
- A parallel system that offers the advantage of being able to scale by simply increasing the number of nodes in the cluster.
- Performance tests of our system using the whole Ethereum blockchain data.

Our system has been developed using the C++ language and the MPI message passing interface. MPI was chosen since it is the de facto standard communication library that is used on high performance clusters.

In the rest of the paper, we first cover the previous work done on the topic of blockchain transaction graph analysis in Sect. 2. Section 3 explains the architecture of the system we are developing. Section 4 presents our distributed graph construction algorithm. Section 5 presents our newly contributed shortest path based parallel calculation of the blacklisted address trace forest algorithm. Section 6 presents various tests that use the distributed graph and report timings obtained on an economical cluster set up on Amazon cloud. Finally, the paper is ended with a discussion and conclusions in Sect. 7.

2 Previous work

One of the earliest works carried out to study blockchain transaction graphs is that of [6] who analyze bitcoin transactions mainly for privacy issues. They investigate the structure of two networks, the transaction and the user networks, constructed from the public Bitcoin blockchain data and their implications for user anonymity. They conclude that it is possible to relate many addresses with each other. By making use of external identifying information and appropriate tools, they also conclude that the activity of known users can be observed.

The work by [7] analyzes bitcoin transactions for statistical properties. They download 180,000 HTML files from a blockexplorer site containing bitcoin transactions for the period from January 2009 to May 2012 and parse them to obtain transactions. They use a union-find graph algorithm [8] so that they can locate sets of addresses which are expected to belong to the same user. They combine all the nodes and the transactions which can be related to an identity and thus, form a graph called entity graph out of these. They report statistics and subgraphs of related transactions using the entity graph as well as the original bitcoin transaction graph with Bitcoin addresses as the nodes.

There have also been some recent works on the analysis of Ethereum transaction graph. The work in [9] studies Ethereum transactions from the perspective of network science and statistical laws of the data. The two datasets they use are small and contain 610K and 680K transactions which are extracted from blocks numbered 200K–300K and 3M–3.2M, respectively. The work by [10] focuses on the topic of whether attackers can de-anonymize addresses on the blockchain. They make use of Neo4j graph database to store and analyze the graph. The results they present are quite limited to simple cases.

Recently, ERC20 tokens have been quite popular. There are some works on the analysis of token transactions. Reference [11] provide an overview of 64K ERC20 token networks and analyze the top 1K tokens. Their results show that individual token networks are frequently dominated by a single hub and spoke pattern. The work by [12] presents a

tool called TokenScope to inspect all token transactions and check to see if the behaviors of the deployed tokens are conforming to the ERC20 standard.

Table 1 presents a comparison of our work with related works. There are also parallel distributed graph processing works such as [13] that operate on massive graphs in the area of scientific computing. The work by [14] discusses the importance of big graph processing, the challenges and what needs to be done in the future for big graph processing to be successful. Scalability, efficiency, diversified querying and analytical capabilities are listed as requirements. The work by [15] discusses parallel graph analytics and identifies three main challenges (i) large graph size, (ii) diversity in graph structure and (iii) complex patterns of parallelism in graph problems. They mention that most current systems may not be providing solutions to all these challenges. We are not aware of any work that performs parallel transaction graph analysis on massive blockchain data.

Our proposed work in this paper differs from the previous works in various aspects. The earlier works used data generated over a few years representing the early blockchain technology emergence phase. The data was relatively small and could fit on one node of a computer system. This, however, will no longer be the case because blockchain data is constantly accumulating and new technologies such as sharding and proof of stake in Ethereum2 [17], and Snowflake to Avalanche family of metastable consensus protocols [18] are expected to increase transaction throughputs greatly. In particular, experiments in [19] demonstrate that 3400 tps can be achieved in Avalanche. There has also been work carried out by [20] in order to accelerate transaction speeds and scalability of proof-of-work based blockchain systems. The work by [21] is another approach that attempts to improve blockchain performance by using graph data structure and parallel mining. All these efforts provide evidence that there will be blockchain systems that provide much higher throughput, which in turn, will lead to massive growth of transaction data. As a result, in order to do analysis on the whole blockchain transaction graph, we need a cluster computer based parallel graph system especially designed for the blockchain data so that as the numbers of transactions grow drastically, the system can simply be scaled by using a larger number of cluster nodes. This is the main objective that we try to achieve in this paper.

Our proposed work also differs from the previous works in that we design our system so that it can handle both cryptocurrency and the popular ERC20 token transactions. In particular, we parse calldata of Ethereum transactions in order to extract 40 popular ERC20 token transfers. Earlier works concentrated mainly on bitcoin or ether cryptocurrency transactions.

We note that blockchains also have important uses in the Internet of Things (IoTs) area and have been the target of recent IoT research works [22, 23]. IoT devices can generate massive amounts of data and transactions. Hence, our work, which aims to analyze transactions in a parallel and scalable way, can also be relevant to the analysis of massive IoT blockchain transactions. For example, the work [24] proposes a scalable decentralized service composition solution that uses a combination of blockchain, software defined networks and fog computing technologies.

This paper is an extended version of our conference paper [25]. In particular, we extend our earlier work by contributing a new parallel algorithm that computes all shortest path based transaction traces to fraud-related blacklisted blockchain addresses. This computation results in a compact forest of trace trees which can later be used by web servers to provide fast transaction trace query service to end-users. The extended version also contributes an implementation of parallel connected components algorithm of [26] and its performance tests on the massive Ethereum blockchain graph. Overall, this extended paper provides a total of nine application tests as opposed to five tests that appear in [25]. The blockchain data that has been

Table 1 Comparison with previous works

Paper	Application	Parallel computer	Dataset
[6]	Blockchain	No	1M bitcoin transaction data
[7]	Blockchain	No	3M bitcoin transaction data
[9]	Blockchain	No	680K and 610K ether transaction data
[10]	Blockchain	No	Subset of ether transaction data
[16]	Scientific computing/social networks	Yes, on a 128 node cluster	Mesh, geometric, Delaunay, web graphs
[12]	Scientific computing	Yes, on 2048 node Blue Waters supercomputer	Mesh, sparse matrix, web graphs
This paper	Blockchain	Yes, on a 16 node cloud cluster	Massive 766M ether and token transaction data

used in the tests have also been updated to cover a larger roughly 5-year interval.

3 Blockchain graph system architecture

Figure 1 depicts the architecture of our blockchain graph analysis system architecture. Block data can be retrieved either from a blockchain node or an Ethereum gateway such as the Infura or Cloudflare Ethereum gateways. Syncing a node can take days and therefore, in our case, we have used the Cloudflare gateway to retrieve blocks. The blocks are then parsed for ether cryptocurrency and ERC20 token transfer transactions. These transactions are then saved in files. These files are then input into our blockchain graph system. The dataset and its detailed format are available at the Zenodo site [27]. In addition to ether, the transfers of 40 ERC20 token contracts including stable coins such as USDT, PAX, EURS, BUSD, GUSD, TRYB and XAUT are extracted from the blocks.

The first blue bottom layer of our system stack in Fig. 1 illustrates the infrastructure layer that provides the computational and data resources. Currently, we use the Amazon cloud to build our cluster providing computational and storage services. For cluster configuration and management, we use the StarCluster tool [28]. The second blue layer in the figure shows the parallel programming libraries that can be used to do programming. In particular, since we are aiming to build a scalable system, we want to maintain a dynamically growing transaction graph which is partitioned into several parts. Hence, each cluster node will be responsible for a part of the graph and MPI message passing libraries will be used to facilitate communication among the nodes.

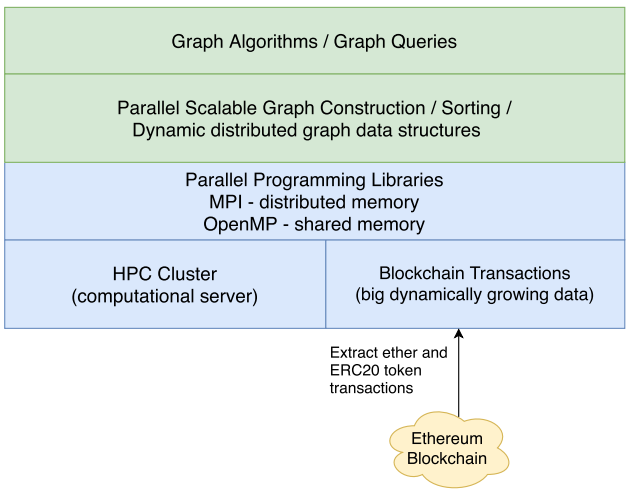


Fig. 1 Blockchain transaction graph system architecture

The third 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 fourth layer which is the graph analysis layer that includes graph queries and algorithms. Given the big dataset files that contain ether and token transfers in the form of *from* and *to* addresses and the *amount* of transfer, this part constructs the global distributed transaction graph by first computing distinct blockchain addresses and then associating the transactions with each address. Note that both addresses and transactions are kept as partitioned data on each node, hence paving the way for a scalable system that can be scaled easily by simply increasing the number of nodes in the cluster.

4 Distributed transaction graph construction

The distributed transaction graph construction algorithm is presented in Algorithm 1 and the symbols used in it are described in Table 2.

In our blockchain transaction data, we have both ether transactions and the ERC20 token transfer transactions of the tokens listed in Table 3. The union of the set of account addresses V_c and the set of smart contract addresses V_s makes up the nodes V of the distributed transaction graph. Each ether transaction E_{ETH} and token transfer transaction E_t is an edge E of the transaction graph.

The first phase (lines 2–9) of distributed transaction graph construction involves the building of the whole node set V of the graph by finding unique Ethereum 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 amount of the transaction. The detailed transaction format is given in [27]. Each processor takes a subset of these files as input, reads the files and creates the set of sender (s) and receiver (t) addresses (lines 2–5). This is the local address set.

The address set is copied to an array because we cannot sort a set. The address set is sorted using parallel sample sort (line 6) [29]. This parallel sample sort is not local. It sorts all addresses in the address array in each processor globally. For better understanding, the global sorting can be thought of as taking the address array of each processor, concatenating these arrays, sorting the concatenated array, splitting the sorted array and putting the addresses back into the address array in each processor. The sample sort is not centralized but distributed and works in parallel.

Table 2 Symbols and their meanings

Symbol	Meaning
V_c	Set of account addresses
V_s	Set of smart contract addresses
V	All blockchain addresses $V = V_c \cup V_s$
E_{ETH}	All blockchain transactions with zero or more ether payments
T	Set of major ERC20 tokens tracked, $T = \{USDT, PAX, TRYB, \ldots\}$, full list given in Table 3
E_t	ERC20 token t transfer transactions, $t \in T$
E	All transfer transactions $E = E_{ETH} \cup E_t$
$G(V, E)$	Transaction graph
ID(v)	Global ID of address v, $ID(v) \in [0, \lvert V \rvert - 1]$
P	Number of processors
p	ID of current processor (0-indexed)
V^p	Blockchain addresses on processor p
E^p	Transfer transactions on processor p
$G^p(V^p, E^p)$	Transaction subgraph on processor p
B	Set of blacklisted addresses
A^p	Set of addresses to be traversed on processor p
F^p	Trace forest on processor p
D^p	Distance of addresses on processor p from blacklisted address
S^p	Set of parent node and depth pairs on processor p to be sent to remote processors
R^p	Set of parent node and depth pairs received from other processors
C	Total number of parent node and depth pairs to be sent

Table 3 Ethereum blockchain data statistics used in tests

(a)	Blocks	0–10,199,999
(b)	Time coverage of blocks	30.07.2015–04.06.2020
(c)	No. of transactions	766,899,042
(d)	No. of addresses	78,945,214
(e)	No. of 40 major ERC20 token transfer transactions	43,371,941
(f)	List of symbols of 40 major ERC20 tokens	USDT TRYb XAUt BNB LEO LINK HT HEDG MKR CRO VEN INO PAX INB SNX REP MOF ZRX SXP OKB XIN OMG SAI HOT DAI EURS HPT BUSD USDC SUSD HDG QCAD PLUS BTCB WBTC cWBTC renBTC sBTC imBTC pBTC

Before sorting the addresses, the addresses in each processor are locally unique, but they are not necessarily unique after sorting them. Suppose that the same address is on the first and second processors. These two addresses will be on the same processor 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 7). Comparing the boundary values, i.e., the last address in one processor and the first one in the next processor, is not needed since sample sort places the same elements on the same processors.

Algorithm 1 Distributed Transaction Graph Construction

```
1:  procedure GENERATEGRAPH(E^p, p)
2:      V^p ← ∅
3:      for each (s,t) ∈ E^p do
4:          V^p ← {s,t} ∪ V^p
5:      end for
6:      SA^p ← parallelSampleSort(V^p)                    ▷ sorted array
7:      V^p ← {SA_i^p | i = 1 ∨ (i ∈ [2..|SA^p|] ∧ SA_i^p ≠ SA_{i−1}^p)}
8:      IdStart ← Σ_{j=0}^{p−1} |V^j|                     ▷ by parallel scan
9:      ID(v_j) = IdStart + j     ∀j ∈ [0..V^p − 1]
10:     SR_0^p ← {(v, ID(v)) | v ∈ V^p}                   ▷ send/recv buffer
11:     SR_1^p ← ∅                                        ▷ send/recv buffer
12:     for i ∈ [0..P − 1] do
13:         j ← i mod 2                                   ▷ index of current SR
14:         k ← i + 1 mod 2                               ▷ index of SR to recv.
15:         for each (s,t) ∈ E^p do
16:             sID ← binarySearch(SR_j^p, s)
17:             if sID ≠ null then
18:                 ID(s) ← sID
19:             end if
20:             tID ← binarySearch(SR_j^p, t)
21:             if tID ≠ null then
22:                 ID(t) ← tID
23:             end if
24:         end for
25:         n ← (i + 1) mod P                             ▷ ID of next proc.
26:         m ← (i − 1 + P) mod P                         ▷ ID of prev. proc.
27:         SR_k^n ← SR_j^p                               ▷ send SR to next proc.
28:         SR_k^p ← SR_j^m                               ▷ recv. SR from prev. proc.
29:     end for
30:     IDE^p ← {(ID(s), ID(t)) | (s,t) ∈ E^p}
31:     SIDE^p ← parallelSampleSort(IDE^p)
32:     formAdjacencyListofGraph(SIDE^p)
33:     return G^p(V^p, E^p)
34: end procedure
```

At this point, we have the unique ethereum addresses and need to give each address a global ID. The addresses in all processors as a whole stand for our addresses. The ID of the first address in the current processor is found by summing the number of addresses in the processors, whose ID ranges from 0 to the current processor ID (excluding) (line 8). The ID of an address is the sum of the ID of the first address and the index of the address in the array (line 9).

In the second phase (lines 10–33), the adjacency list of the graph is constructed from the Ethereum transactions and global IDs of the transaction addresses. The sender and receiver addresses need to be mapped to their corresponding global IDs. Since the address set is distributed, each processor has only a part of the addresses. But we need to know the global ID of all addresses to be able to map them. To solve this problem, each processor sends its addresses to the next processor and receives the addresses from the previous processor in a ring fashion. These send and receive operations are done for the number of processor times. At each iteration, the processor 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 processor and receive the addresses from the previous processor at the same time (lines 10–11). The buffer that contains the addresses will alternate with every iteration (line 13). The other buffer only receives the addresses for the next iteration (line 14). Each processor iterates for the number of processor times (line 12) and processes the address information again (line 15). If the sender or receiver address is found in the address set (located in the buffer), the address is given its global ID (lines 16–23). The current address set is sent to the next processor (line 27). The address set to be used at the next iteration is received from the previous processor (line 28).

The edges of the graph are converted from local IDs to the global IDs (line 30). The edges are sorted according to IDs of address pairs, ($ID(s)$, $ID(t)$), making up the transactions (line 31) and the adjacency list representation of the graph is formed. This adjacency list is the output of the algorithm. Figure 2 illustrates the execution of our algorithm on a small example dataset.

5 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. Such a trace forms a directed subgraph in general. 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 trees trace. 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 3a, b show a forest of trees computed to depth 5 and an example tree trace to a blacklisted address of the DragonEx hacker [30].

The distributed calculation of the blacklisted addresses trace forest is presented in Algorithm 2 and the symbols used in it are described in Table 2. 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 processor. The algorithm first visits all local nodes and stores the

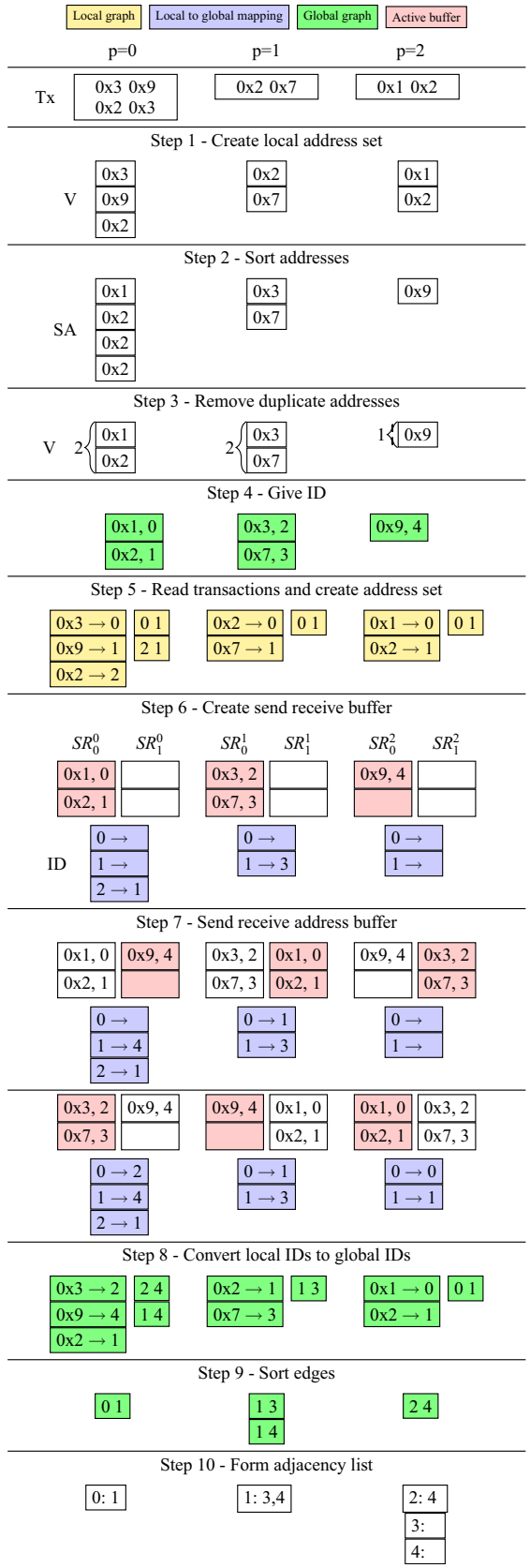
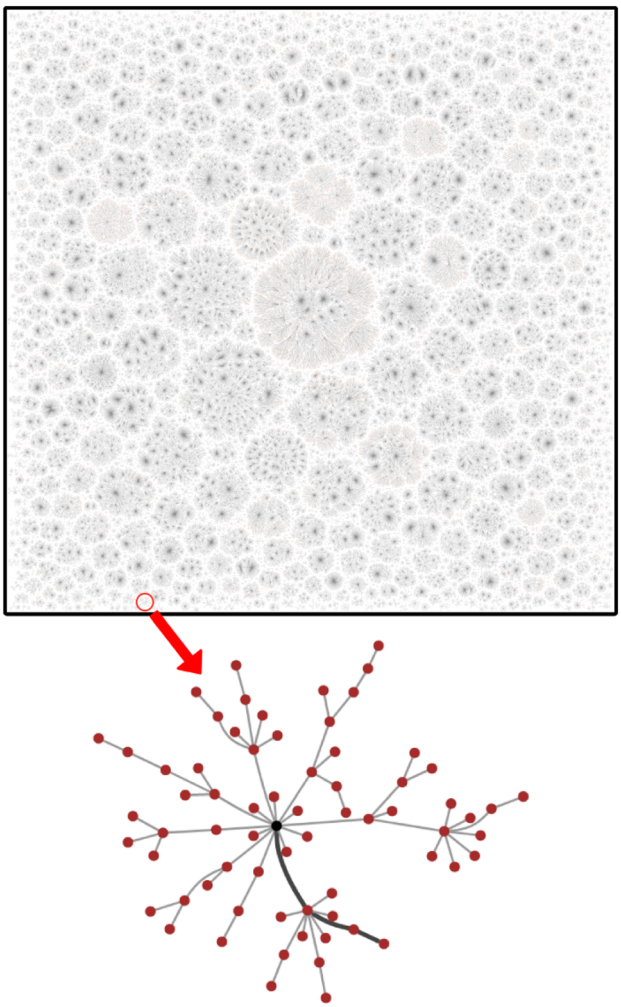
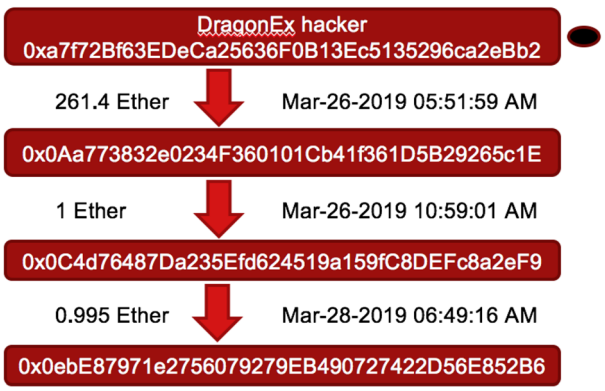


Fig. 2 Example showing steps of Algorithm 1



(a)



(b)

Fig. 3 Example tree trace to blacklisted address of the DragonEx hacker [30]

remote nodes that need to be visited. The stored node information is then exchanged between processors. These new nodes are traversed locally again. This traversal and communication cycle continues until there is no remote node left to be visited.

Algorithm 2 Distributed Calculation of the Blacklisted Node Trace Forest

```
1: function BUILDTRACETREES(p, B, G^p(V^p, E^p))
2:     A^p ← ∅
3:     for all v ∈ V^p do                                   ▷ initialize the forest
4:         F^p(v) ← ∅
5:     end for
6:     for all b ∈ B do      ▷ add blacklisted addresses as root nodes
7:         if b ∈ V^p then
8:             F^p(b) ← b                                  ▷ root node points to itself
9:             D^p(b) ← 0
10:            A^p ← A^p ∪ {b}                             ▷ add to stack to visit later
11:        end if
12:    end for
13:    |A| ← Σ_{i=0}^{P−1} A^i                             ▷ allreduce
14:    while |A| > 0 do                                    ▷ while there is nodes to visit
15:        S^p ← ∅
16:        while A^p ≠ ∅ do    ▷ until no local node remained to visit
17:            A^p ← A^p \ {s}    ∃s ∈ A^p                 ▷ pop a node from stack
18:            for all (s,t) ∈ E^p do                      ▷ visit the neighbors of s
19:                if t ∈ V^p then                         ▷ if local node
20:                    VISITNODE(F^p(t), s, t, D^p(s) + 1, D^p(t))
21:                else                                    ▷ if remote node
22:                    if (S^p(t) = ∅) ∨
23:                       ((S^p(t) ≠ ∅) ∧
24:                       (D^p(s) + 1) < d | (v,d) ∈ S^p(t))) then
25:                        S^p(t) ← (s, D^p(s) + 1)
26:                    end if
27:                end if
28:            end for
29:        end while
30:        C ← Σ_{i=0}^{P−1} |S^i|                         ▷ allreduce
31:        if C > 0 then        ▷ if there is any node info to send/recv
32:            for all t ∈ S^p do
33:                Send S^p(t) to processor x such that t ∈ V^x
34:            end for
35:            Receive sent S^p(t) to R^p(t)
36:            for all t ∈ R^p do
37:                (s,d) ← R^p(t)
38:                VISITNODE(F^p(s), s, t, d, D^p(t))
39:            end for
40:        end if
41:        |A| ← Σ_{i=0}^{P−1} A^i                         ▷ allreduce
42:    end while
43:    return F^p
44: end function

45: procedure VISITNODE(f, s, t, d1, d2)
46:    if (f = ∅) ∨ ((f ≠ ∅) ∧ (d1 < d2)) then   ▷ is null or shorter
47:        A^p ← A^p ∪ {t}                                 ▷ add target to stack
48:        F^p(t) ← s                                      ▷ target points to source in tree
49:        D^p(t) ← d1
50:    end if
51: end procedure
```

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 processor adds the blacklisted nodes that belong to the current processor to the forest as a root and to its stack (lines 6–12). The total number of nodes in the stack of all processors is calculated (line 13). While there are nodes to be visited, the traversal and communication cycle will continue (lines 14–42).

In the traversal part (lines 15–29), 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 target node of the edge is a local node (line 19), and if the node is unvisited or it is visited but a shorter path is found, the node is added to the stack, the source 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 source node (lines 46–50). If the target node of the edge is a remote node (line 21), the source node and depth information is stored in a map S to be sent to the corresponding node in the communication part (lines 22–26). This source node and depth information in the map is updated if a shorter path is found.

After the traversal part, the number of cut edges is calculated to determine whether the cut edge communication part is needed (line 30). If there are cut edges, the cut edge information is sent and received. For each received node (line 36), if the node is unvisited or it is visited but a shorter path is found, the node is added to the stack, the source 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 source node (line 38).

At the end of the loop, the total number of nodes in the stack of all processors is calculated again (line 41). If there is any node in the stack, the traversal and communication cycle will continue. If not, the algorithm will return the forest. Figure 4 illustrates the execution of our algorithm on a small example dataset.

6 Tests

Our cluster-based system for analyzing Ethereum blockchain transaction data is tested on Amazon EC2 cloud using 16 c5.4xlarge machine instances, each of which has 16 virtual CPUs (8 core with hyper-threading) and 32 GiB memory. We store the Ethereum data in a gp3 volume (Amazon EBS). Network file system (NFS) is set up on the master node to access data. Amazon provides 4750 Mbps EBS bandwidth and up to 10 Gbps network bandwidth to EC2 instances. The placement group of machines is set as "cluster" so that EC2 packs the instances close together. Various statistics about the Ethereum blockchain dataset that we used are given in Table 3.

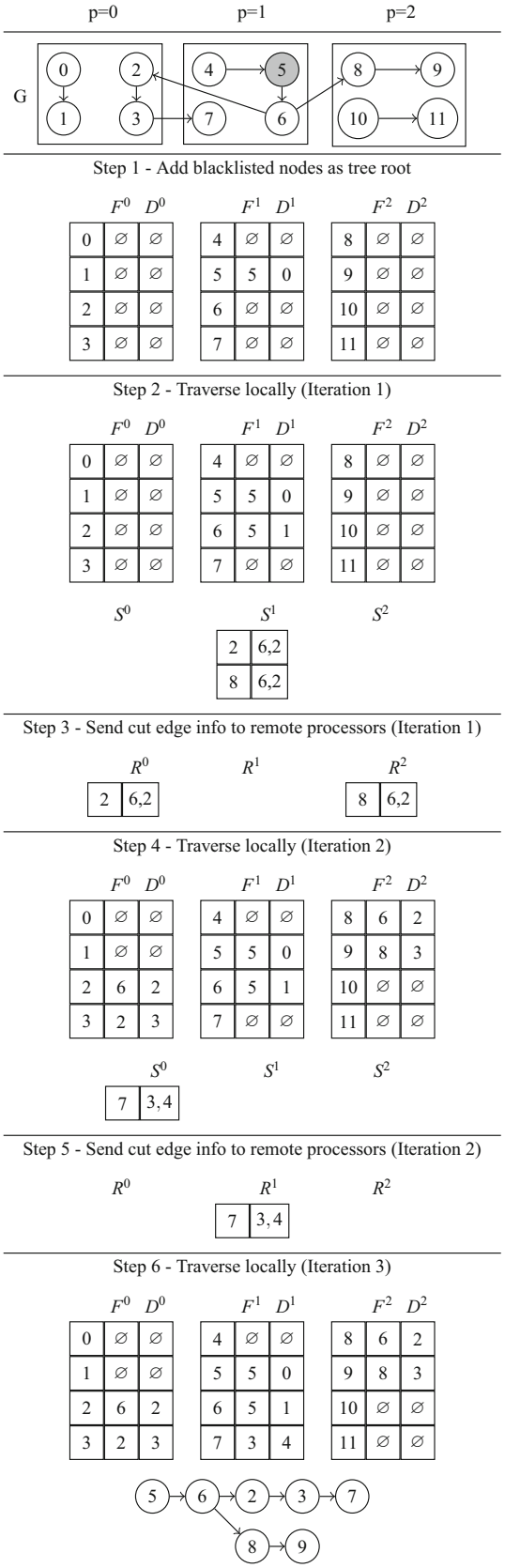


Fig. 4 Example showing steps of Algorithm 2

Table 4 Description of tests

Test	Description
T1	Transaction graph construction
T2	Graph partitioning using ParMetis [31]
T3	Page ranking on transaction graph
T4	Node degrees and degree distributions of transaction graph
T5	No. of transfer transactions of 40 major ERC20 tokens
T6	Connected component count
T7	Blacklisted node trace forest
T8	Extracting node features
T9	Example of trace subgraph of a blacklisted address

The descriptions of the tests that are carried out are given in Table 4. All these tests are programmed in C++ using the MPI libraries. In the first test, T1, the directed, transposed, and undirected transaction graph is constructed in parallel. The parallel directed transaction graph construction algorithm is presented in Sect. 4. In test T2, we use ParMetis [31] software in order to partition the undirected transaction graph in parallel. Test T3 runs the parallel page ranking algorithm on the transaction graph. Test T4 is a smaller test involving computing degree distributions (i.e., number of incoming and outgoing edges) of each address node in the 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 trace forest of blacklisted nodes is built. In test T8, the following features are calculated for each node: 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 calculated.

Since each node of the cluster has 16 virtual CPUs, we also run tests with multiple MPI processes per cluster node. Table 5 shows the timings obtained from the tests. In the table, we report P, which is the total number of MPI processes.

Each test is repeated twice and their average is given in the table. Note that ParMetis [31] partitioner test (T2) takes a long time to complete. In order to avoid high Amazon cloud costs, we only performed it on 16 nodes with 1 MPI process per node.

Table 5 Timings of tests in seconds

Test	1 MPI process per node			
	P = 4	P = 8	P = 12	P = 16
T1	1314	748	509	443
T2	–*	5768	6056	5718
T3	991	622	370	372
T4	14.5	10.8	8.3	7.5
T5	149	85.6	53.4	45.1
T6	1319	855	580	491
T7	11243	3383	1082	323
T8	116	70.5	39.9	34.1
T9	4.0	1.9	1.4	1.2
	2 MPI processes per node			
	P = 8	P = 16	P = 24	P = 32
T1	712	430	331	297
T3	566	369	276	247
T4	10.2	8.2	7.1	6.8
T5	81	45.3	26.8	20.8
T6	841	487	355	329
T7	2178	355	108	58.2
T8	66.4	32.3	26.3	22.3
T9	1.9	1.1	0.9	0.8
	4 MPI processes per node			
	P = 16	P = 32	P = 48	P = 64
T1	480	327	286	256
T3	401	232	206	167
T4	8.3	7.1	11.1	11.0
T5	43	20.7	33.8	21.3
T6	505	309	254	219
T7	260	58.2	46.1	33.5
T8	34.2	21.1	18.7	14.3
T9	1.2	0.7	0.6	0.5
	8 MPI processes per node			
	P = 32	P = 64	P = 96	P = 128
T1	401	287	227	218
T3	252	163	150	156
T4	7.1	10.5	14.5	18.3
T5	20.8	20.2	19.8	19.3
T6	324	224	197	180
T7	57.5	37.2	37.6	32.4
T8	22.3	14.2	12.7	12.7
T9	0.8	0.5	0.5	0.5
	12 MPI processes per node			
	P = 48	P = 96	P = 144	P = 192
T1	506	303	251	234
T3	213	212	211	219
T4	14.2	15.9	25.6	35.5
T5	44.4	30.9	25.7	25.3
T6	305	229	177	186
T7	53.7	51.9	33.0	37.6
T8	24.8	15.9	16.2	14.9
T9	0.8	0.6	0.7	1.5
	16 MPI processes per node			
	P = 64	P = 128	P = 192	P = 256
T1	426	352	331	331
T3	247	219	209	224
T4	14.2	24.6	38.3	55.0
T5	29.5	28.2	27.7	31.6
T6	284	216	188	184
T7	39.0	34.9	42.3	42.7
T8	20.0	17.1	14.8	16.1
T9	0.9	0.6	0.5	1.5

*Out of memory

We plot the distributed transaction graph test T1 in Fig. 5. The plot shows that as we increase the number of processors, the timings decrease, leveling off at around 128 processors. At 128 processors, the whole distributed graph can be constructed in 218 s.

ParMetis graph partitioner (test T2) reports the number of edges cut among partitions. These results are shown in Table 6.

Parallel MPI implementation of Pagerank algorithm is also run as test T3 on the whole distributed transaction graph that was constructed. In the ranking computed, the topmost important five are reported to be addresses of Binance, Bitfinex, Shapeshift, Plus Token and MSD Token. With the exception of Plus Token and MSD Token, the rest are cryptocurrency exchange companies. Using 96 processes with 8 per node, Pagerank algorithm is able to rank the addresses in 150 s.

Tests T4 and T5 involve simpler queries over the addresses. Test T4 computes node incoming and outcoming degrees and degree distributions of transaction graph and its fastest run took 7.1 s with 32 processes. We compute individual incoming and outcoming node degrees for each node because this information can be made use of as features in machine learning algorithms in the future. Outcoming nodes degrees are readily available in the data structures, but incoming degrees require communication

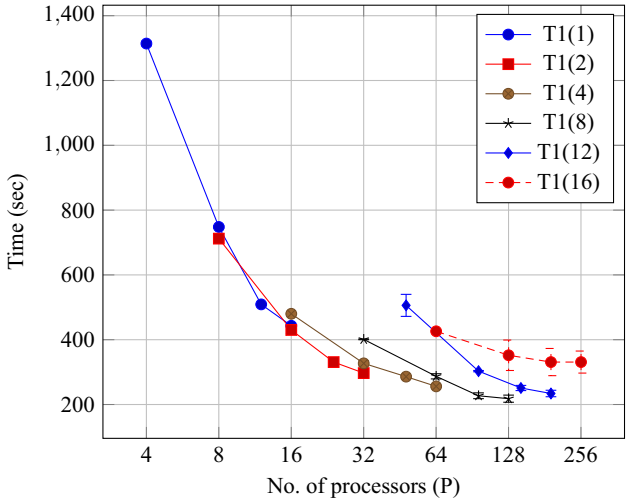


Fig. 5 Test T1 distributed transaction graph construction timings. The integers i in parentheses T1(i) indicate number of MPI processes per cluster node

Table 6 Edges cut across partitions reported by ParMetis

P	No. of edges cut
4	–*
8	35,527,579
12	43,722,740
16	47,462,684

*Out of memory

among processors. Note that in T4 test the fraction of computation is very little compared to communication. That is why the running time increases as the number of processors is increased. Test T5 computes number of transfer transactions which involve a global summation operation and its fastest run took 19.3 s with 128 processes. Figure 6 shows the indegree and outdegree distributions of the blockchain addresses. Figure 7 shows the numbers of transfer transactions of ten major ERC20 tokens reported by our program. Figures 8 and 9 show plots of timings obtained on the tests T4 and T5, respectively.

For test T6, we implement the parallel connected components algorithm of [26] named FastSV on our blockchain transaction graph system. The fastest run of this algorithm takes 177 s on 144 processes. Considering a costly supercomputer is used to carry out FastSV tests in [26], our experiments with the FastSV algorithm confirm that this algorithm runs fast on economical cloud based clusters. The timing plot for test T6 is given in Fig. 10.

Test T7 runs the distributed trace forest algorithm contributed in Sect. 5. In this test, full forest of trees computation is performed with no limitation on tree depth (i.e., no pruning). The fastest run takes 32.4 s on 128 processors. Figure 11 plots the timings obtained when running test T7.

Finally, we note that test T8 involves iterating over all addresses but does a simple calculation and hence taking a small amount of time. On the other hand, T9 involves traversals around the blacklisted node, hence taking a very small amount of time.

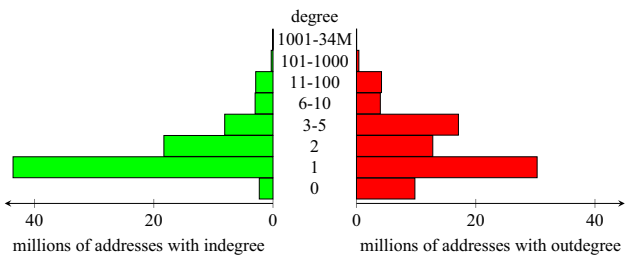


Fig. 6 Degree distributions of transaction graph nodes

7 Discussion and conclusion

Currently, Ethereum blockchain is able to deliver around 14 tps. As of April 23, 2021, the total number of transactions since July 30, 2015, was reported to be 1.1 billion [32]. If tps reaches 100, it will mean around 3.15 billion transactions per year. If tps reaches 1000, then 31.5 billion yearly transactions may be possible. If one needs to do graph analysis on such a huge and dynamically growing graph, a parallel cluster-based system is needed so that the system can be scaled by increasing the number of nodes. Our objective in this work is to develop the cluster and software stack so that when the blockchain data deluge hits because of the arrival of new consensus technologies with large tps [17, 19], we will be able to carry out our blockchain transaction graph analysis processes without interruption.

The timings obtained for graph construction as well as other queries were good. Our blockchain transaction data was 81 GB of edge data. Our system was able to load and construct adjacency distributed graph representation in

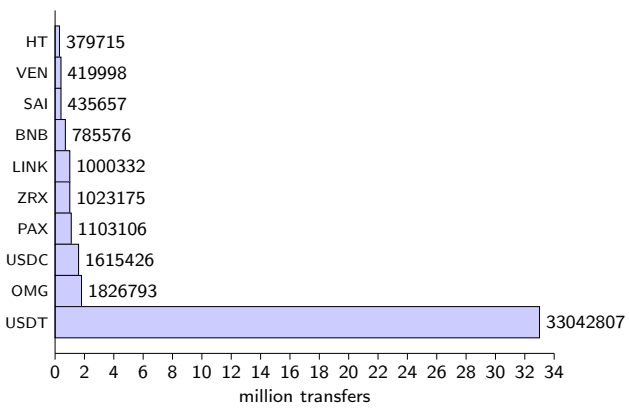


Fig. 7 Numbers of transfer transactions of ten major ERC20 tokens

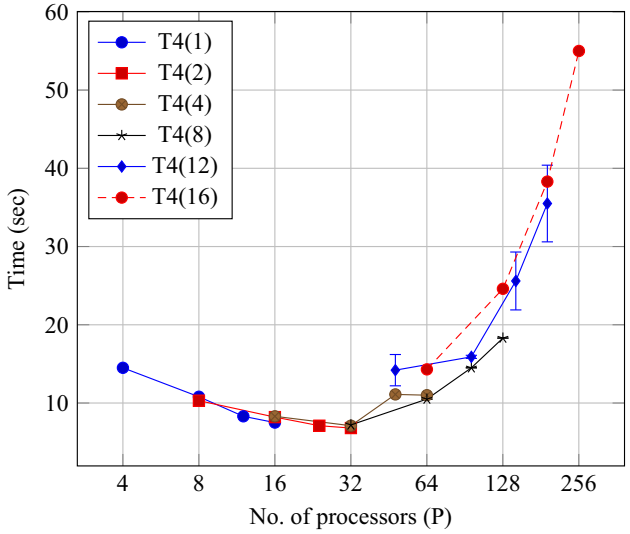


Fig. 8 Test T4 node indegree and outdegree degree distributions of transaction graph nodes computation timings. The integers i in parentheses T4(i) indicate number of MPI processes per cluster node

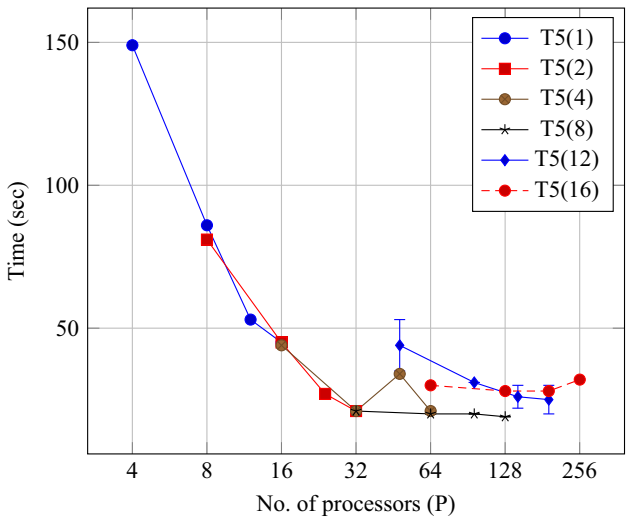


Fig. 9 Test T5 no. of transfer transactions of 40 major ERC20 tokens computation timings. The integers i in parentheses T5(i) indicate number of MPI processes per cluster node

about 3.63 min on a 16 node Amazon cluster. With 68 cents per node cost, our 16 node cluster system would enable 1 h of analysis to be carried out for 10.88 dollars. A large company like a bank can keep this cluster running continuously and offer analysis services to many small businesses by charging a subscription fee.

Graph partitioning packages like ParMetis divide the graph into multiple parts in such a way that (i) each part is load-balanced, i.e. having roughly the same number of graph nodes and (ii) the number of edges cut across partition boundaries are minimized. The latter enables communication volume to be reduced. We perform graph

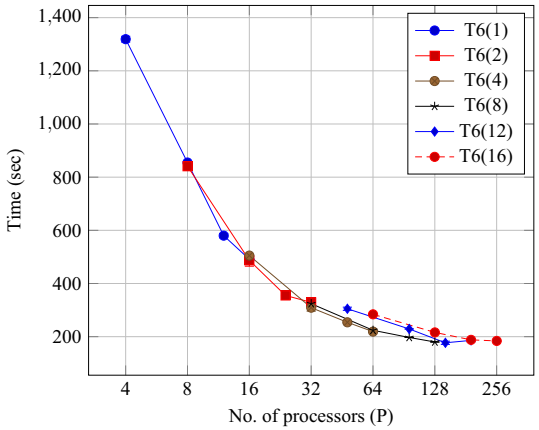


Fig. 10 Test T6 connected component count computation timings. The integers i in parentheses T6(i) indicate number of MPI processes per cluster node

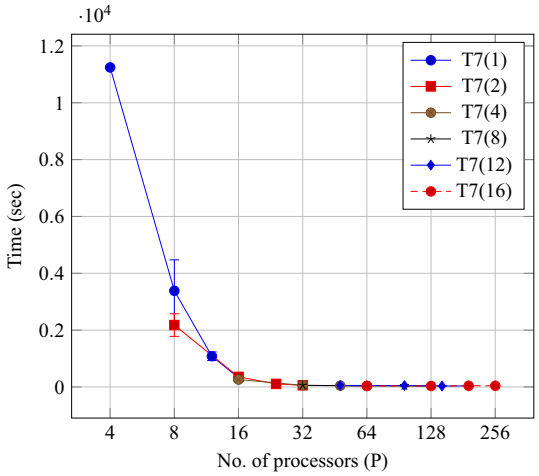


Fig. 11 Test T7 distributed forest of trees blacklisted address trace computation timings. The integers i in parentheses T7(i) indicate number of MPI processes per cluster node

traversals for tracing addresses to see if they are involved in transactions originating from fraudulent addresses. Therefore, if the number of edges cut among partitions is minimized, it can reduce jumping from one partition (i.e., one processor) to another while doing traversals. ParMetis is currently taking more than an hour to partition the whole graph into 16 parts. In the future, we plan to work on developing our own speedier graph partitioning heuristics designed especially for the blockchain transaction graph. Now that we have our distributed graph infrastructure, we also plan to use it to perform graph analysis based on machine learning algorithms.

Author contributions All authors contributed to design and development of the system as well as the manuscript. All authors have read and approved the final manuscript.

Funding This work has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 856632.

Data availability The datasets generated during and/or analysed during the current study are available in the Zenodo Repository: https://zenodo.org/record/4718440 (version 2).

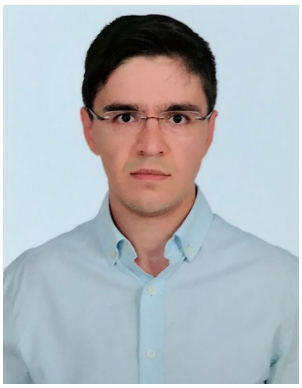
Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Bing, C.: Exclusive: U.S. to give ransomware hacks similar priority as terrorism (June 2021). https://www.reuters.com/technology/exclusive-us-give-ransomware-hacks-similar-priority-terrorism-official-says-2021-06-03/. Accessed 2 Nov 2021
2. Office of Public Affairs: Department of Justice Seizes 2.3 Million in Cryptocurrency Paid to the Ransomware Extortionists Darkside. Office of Public Affairs (June 2021). https://www.justice.gov/opa/pr/department-justice-seizes-23-million-cryptocurrency-paid-ransomware-extortionists-darkside. Accessed 2 Nov 2021
3. del Castillo, M.: Bitcoin investigation giant to raise 100 million at 1 billion valuation (Nov 2020). https://www.forbes.com/sites/michaeldelcastillo/2020/11/20/bitcoin-investigation-giant-to-raise-100-million-at-1-billion-valuation. Accessed 2 Nov 2021
4. Mastercard acquires CipherTrace to enhance crypto capabilities (September 2021). https://www.mastercard.com/news/press/2021/september/mastercard-acquires-ciphertrace-to-enhance-crypto-capabilities. Accessed 2 Nov 2021
5. FATF: Guidance for a Risk-Based Approach, Virtual Assets and Virtual Asset Service Providers. FATF, Paris (2019). https://www.fatf-gafi.org/publications/fatfrecommendations/documents/guidance-rba-virtual-assets.html. Accessed 2 Nov 2021
6. Reid, F., Harrigan, M.: An analysis of anonymity in the bitcoin system. In: 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing, pp. 1318–1326 (2011)
7. Ron, D., Shamir, A.: Quantitative analysis of the full bitcoin transaction graph. In: International Conference on Financial Cryptography and Data Security, pp. 6–24. Springer (2013)
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2009)
9. Guo, D., Dong, J., Wang, K.: Graph structure and statistical properties of ethereum transaction relationships. Inf. Sci. **492**, 58–71 (2019)
10. Chan, W., Olmsted, A.: Ethereum transaction graph analysis. In: 12th International Conference for Internet Technology and Secured Transactions (ICITST), 2017, pp. 498–500. IEEE (2017)
11. Victor, F., Lüders, B.K.: Measuring ethereum-based ERC20 token networks. In: International Conference on Financial Cryptography and Data Security, pp. 113–129. Springer (2019)
12. Chen, T., Zhang, Y., Li, Z., Luo, X., Wang, T., Cao, R., Xiao, X., Zhang, X.: TokenScope: automatically detecting inconsistent behaviors of cryptocurrency tokens in ethereum. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (2019)
13. Slota, G.M., Rajamanickam, S., Devine, K., Madduri, K.: Partitioning trillion-edge graphs in minutes. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2017, pp. 646–655 (2017)
14. Sakr, S., Bonifati, A., Voigt, H., Iosup, A., Ammar, K., Angles, R., Aref, W., Arenas, M., Besta, M., Boncz, P.A., et al.: The future is big graphs: a community view on graph processing systems. Commun. ACM **64**(9), 62–71 (2021)
15. Lenharth, A., Nguyen, D., Pingali, K.: Parallel graph analytics. Commun. ACM **59**(5), 78–87 (2016)
16. Meyerhenke, H., Sanders, P., Schulz, C.: Parallel graph partitioning for complex networks. IEEE Trans. Parallel Distrib. Syst. **28**(9), 2625–2638 (2017)
17. Ethereum 2.0 phases (2019). https://docs.ethhub.io/ethereum-roadmap/ethereum-2.0/eth-2.0-phases/. Accessed 2 Nov 2021
18. Rocket Team: Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies (2018)
19. Rocket Team, Yin, M., Sekniqi, K., van Renesse, R., Sirer, E.G.: Scalable and probabilistic leaderless BFT consensus through metastability (2019). arXiv preprint arXiv:1906.08936
20. Hazari, S.S., Mahmoud, Q.H.: A parallel proof of work to improve transaction speed and scalability in blockchain systems. In: 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), pp. 0916–0921 (2019)
21. Kan, J., Chen, S., Huang, X.: Improve blockchain performance using graph data structure and parallel mining. In: 2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN), pp. 173–178. IEEE (2018)
22. Alfandi, O., Otoum, S., Jararweh, Y.: Blockchain solution for IoT-based critical infrastructures: Byzantine fault tolerance. In: NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, pp. 1–4. IEEE (2020)
23. Tseng, L., Yao, X., Otoum, S., Aloqaily, M., Jararweh, Y.: Blockchain-based database in an IoT environment: challenges, opportunities, and analysis. Clust. Comput. **23**(1), 1–15 (2020)
24. Al Ridhawi, I., Aloqaily, M., Boukerche, A., Jaraweh, Y.: A blockchain-based decentralized composition solution for IoT services. In: ICC 2020-2020 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2020)
25. Kılıç, B., Özturan, C., Sen, A.: A cluster based system for analyzing ethereum blockchain transaction data. In: Second International Conference on Blockchain Computing and Applications (BCCA), 2020, pp. 59–65 (2020)
26. Zhang, Y., Azad, A., Hu, Z.: FastSV: a distributed-memory connected component algorithm with fast convergence. In: Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing, pp. 46–57. SIAM (2020)
27. Özturan, C., Şen, A., Kılıç, B.: Transaction Graph Dataset for the Ethereum Blockchain (April 2021). https://doi.org/10.5281/zenodo.4718440. Accessed 2 Nov 2021

28. StarCluster (2013). http://star.mit.edu/cluster/. Accessed 2 Nov 2021
29. Shi, H., Schaeffer, J.: Parallel sorting by regular sampling. J. Parallel Distrib. Comput. **14**(4), 361–372 (1992)
30. Khatri, Y.: Singapore-based crypto exchange DragonEx has been hacked (March 2019). https://www.coindesk.com/singapore-based-crypto-exchange-dragonex-has-been-hacked. Accessed 2 Nov 2021
31. Karypis, G., Kumar, V.: METIS: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0. University of Minnesota, Minneapolis (2009). http://glaros.dtc.umn.edu/gkhome/views/metis. Accessed 2 Nov 2021
32. Ethereum daily transactions (2020). https://etherscan.io/chart/tx. Accessed 2 Nov 2021



Baran Kılıç received his B.Sc. Degree in Computer Engineering from Bogaziçi University in 2019. He is currently pursuing his M.Sc. Degree in Computer Engineering from Boğaziçi University. His current research interests include blockchain, parallel programming and machine learning.

Can Özturan received his Ph.D. Degree in Computer Science from Rensselaer Polytechnic Institute, Troy, NY, USA, in 1995. After working as a Postdoctoral Staff Scientist at the Institute for Computer Applications in Science, NASA Langley Research Center, he joined the Department of Computer Engineering, Bogazici University in Istanbul, Turkey, as a Faculty Member in 1996. His research interests are blockchain technologies, parallel processing, scientific computing, resource management, graph algorithms and grid/cloud computing.

Alper Sen received the B.S. and M.S. Degrees in Electrical and Electronics Engineering from Middle East Technical University, Ankara, Turkey, in 1995 and 1997, respectively, and the Ph.D. Degree in Electrical and Computer Engineering from the University of Texas at Austin, Austin, TX, USA, in 2004. He is currently a Full Professor with the Department of Computer Engineering, Bogazici University, Istanbul, Turkey. His current research interests include software testing, verification, and machine learning.