Automating ETL and Mining of Ethereum Blockchain Network

Voon Hou Su
Nanyang Technological University
Singapore
vsu001@e.ntu.edu.sg

Sourav Sen Gupta
Nanyang Technological University
Singapore
sg.sourav@ntu.edu.sg

Arijit Khan
Nanyang Technological University
Singapore
arijit.khan@ntu.edu.sg

ABSTRACT

The popularity of blockchain has led to the development of many web platforms with different functionalities. Ethereum, a decentralized, open-source blockchain featuring smart contracts, introduces an interesting ecosystem of human users and autonomous agents (the contracts). It is the most actively used blockchain platform, hosting *ether*, the second largest cryptocurrency by market capitalization, as its native store of value. The Ethereum blockchain contains a vast amount of user-to-user, user-to-contract, contract-to-user, and contract-to-contract interactions that can be modeled as complex networks. To mine these interactions as graphs using any analytics toolbox, an end-user has to **e**xtract, **t**ransform, and **l**oad (ETL) the data into a desired network format. However, it is costly and time-consuming to manage the ETL pipeline for the massive and complex blockchain data. To support research in this domain, we develop an end-to-end, automated tool — EtherNet, which performs ETL tasks from a single source of truth (Google BigQuery), and provides graph equivalent representations for visualization and mining on the entire Ethereum blockchain network.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; **Exploratory data analysis**; • **Applied computing** → **Digital cash**.

KEYWORDS

Ethereum Blockchain Network, Extract-Transform-Load, Financial Data Mining

ACM Reference Format:
Voon Hou Su, Sourav Sen Gupta, and Arijit Khan. 2022. Automating ETL and Mining of Ethereum Blockchain Network. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM '22), February 21–25, 2022, Tempe, AZ, USA*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3488560.3502187

1 INTRODUCTION

Blockchain technology attracted extensive attention from the industry and from academia, giving rise to popular cryptocurrencies and web platforms. This is well suited for Web 3.0 – a decentralized, secure internet, where individuals engage with each other in economic interactions. Blockchain is a distributed ledger of transactions or records, stored in a sequential order, ensuring provenance, and can be critical in the *trust economy* of the future [10]. Bitcoin was the first blockchain implemented over a decade ago [11]. In 2014–2015, Ethereum[1] was introduced, allowing users to prove ownership of assets, and also to execute code as *smart contracts*. Ethereum is the most actively-used blockchain in the world, with the second most prominent cryptocurrency *ether* as a store of value. It is used in many applications, e.g., cryptocurrencies, decentralized finance, smart city, Internet-of-Things, and gaming [6].

Blockchain introduces new directions in online financial data mining and analysis. Ethereum blockchain contains a vast amount of information documenting user-to-user, user-to-contract, contract-to-user, and contract-to-contract interactions. Recent works [3, 5, 7–9, 12–14, 16, 17] modeled transactions, tokens, and interactions in Ethereum (and in other blockchains) as graphs to provide new insights through network mining. Graph-based analyses are powerful tools to characterize activities and time-varying patterns over blockchains [8, 17], to address attack forensics and anomaly detection [3], to detect abnormal transactions (voting gangs and frauds [18]), and to correlate price dynamics with graph features, identifying financial risks for cryptocurrency users [1].

1.1 Challenges in Ethereum Network Analysis

Past research [2, 3, 5, 9, 13, 16, 19], with the exception of ours [8, 17], extracted Ethereum data mainly by running an Ethereum node[2], or querying the blockchain using managed services.

One can join the Ethereum network of nodes through a "client" to verify blocks, transactional data, and also to interact with the blockchain. Geth is the most popular software client for running a node on Ethereum. Once a Geth node completes a "Fast" sync, it switches to full archive sync for which approximately 5TB of storage space is required[3]. Due to massive volume, it takes more than a week to fully synchronize entire data at a newly connected node. This also requires users to set up a stable system with robust hardware, huge storage space, and network bandwidth. Such an approach is not feasible for users who only require ad-hoc access to Ethereum data for analyses. Instead, users generally interact with Ethereum nodes via the `web3` library using managed services, e.g., Infura and Quicknode. The number of requests that one can make per day depends on the subscription package, and thus, users are faced with high costs if they want to extract large amounts of data. Moreover, blockchain data is stored at clients in heterogeneous, complex data structures, in binary or in encrypted format, which cannot be directly used for exploration, mining, or visualization.

WSDM '22, February 21–25, 2022, Tempe, AZ, USA

ACM ISBN 978-1-4503-9132-0/22/02…$15.00
https://doi.org/10.1145/3488560.3502187

[1] V. Buterin. https://ethereum.org/en/whitepaper/
[2] https://ethereum.org/en/developers/docs/nodes-and-clients/
[3] https://decrypt.co/24779/ethereum-archive-nodes-now-take-up-4-terabytes-of-space

Alternative efforts are made to provide users with an easy and economically viable access to a single source of truth of the Ethereum blockchain data via public datasets, e.g., Google BigQuery [4] and XBlock-ETH [19]. Ethereum blockchain data, available on Google BigQuery, is updated daily and is accessible through an SQL interface. The data is stored in eight tables, as listed in Table 1. In recent works [8, 17], we extracted relevant data from Google BigQuery.

Table 1: Ethereum Data on Google BigQuery (till 25 May 2021)

Table	Row Count (Million)	Approximate Size of Dataset (GB)
balances	153.56	8.58
blocks	11.72	12.07
contracts	37.39	31.77
logs	1 043.24	502.84
token transfers	595.69	171.89
tokens	0.19	0.03
traces	2775.28	1626.74
transactions	985.76	455.64

```
SELECT table_id,
TRUNC(row_count/1000000, 2) AS row_millions,
TRUNC(size_bytes/1073741824, 2) AS size_gb,
FROM `bigquery-public-data.crypto_ethereum._TABLES_`
```

While this method appears to be straightforward, there still exist many ETL issues and Google Cloud Platform (GCP) related charges.

— **First**, due to lack of automated ETL, users need to identify required range of data, write custom tools to curate data from Ethereum blockchain to obtain raw data, build one-off tools to ETL raw data into desired graph representations, e.g., Neo4J or NetworkX formats.

— **Second**, no solutions exist to catalogue the data being downloaded. Manual ETL workflow gets messy resulting in users downloading duplicate data that were downloaded in previous workflows.

— **Third,** GCP restricts how much data can be downloaded directly from the SQL query results page. If output > 1GB, one may use Google Cloud Storage (GCS) Bucket before the dataset can be downloaded. Users are charged for data storage and network usage, on top of BigQuery related costs of querying and storing.

1.2 Our Solution and Contributions

We build and open-source an automated tool, EtherNet [15], to simplify the ETL pipeline from Ethereum data into relevant graph representations. *This will benefit researchers, practitioners, developers, data scientists, and financial analysts dealing with Ethereum blockchain data.* EtherNet provides a Python SDK aiming at **(1)** a scalable, efficient storage system for Ethereum data, **(2)** a consistent access layer to data and ETL workflows, and **(3)** an efficient discovery of graphs and running queries. The Python SDK (Client) initiates ETL requests, while the Java (JDK 11) backend processes the requests. EtherNet's core is defined as protobufs — a language-neutral, open-source, cross-platform library to serialize structured data. Our employed dependencies support horizontal scaling.

We conceptualize four interaction networks [8, 17]: *TraceNet*, consisting of all recorded messages and successful transactions between all accounts (both users and smart contracts); *TransactionNet*, formed by all transactions from user accounts to other addresses; *ContractNet*, by considering transactions between smart contracts; and *TokenNet*, containing token-based transactions. Our system architecture is given in § 2. We discuss demonstration plan, with one visual query form and another analytical query form, as well as EtherNet's performance in § 3. *The entire* EtherNet *codebase is open-sourced [15] and a video demonstration of its usage is available at the YouTube direct link* — **https://youtu.be/DFzIya3sDfM**.

2 SYSTEM OVERVIEW

Solution architecture. The high-level architecture of EtherNet is illustrated in Figure 1. Python SDK (Client) allows the user to initiate an ETL flow by sending an `UpdateRequest` to a gRPC endpoint, while the Core (Service Layer) maps it to the concrete implementation of `GenericService` to handle the gRPC call. If all data required to serve the request are available in Hive, the structured (tabular) data are fetched, they are transformed into the desired graph format [8, 17], and are loaded into the Neo4j database. If a part of the required data is not available in Hive, the tool automatically fetches the incremental data from Google BigQuery to enrich the curated data source before creating the graph. The pertinent Neo4j graph is returned to the Python SDK (Client) for mining and visualization.

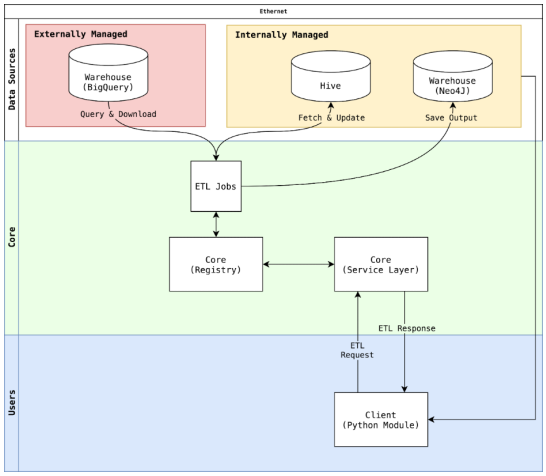


Figure 1: High level architecture diagram of EtherNet

Application topology. The high-level application topology of EtherNet and all its components is illustrated in Figure 2. EtherNet Core and EtherNet Serving components are meant to be deployed on the same host, while the Python SDK client may be deployed on separate (individual) hosts. The data store is maintained independently in a Hive warehouse hosted on a Hadoop cluster. The EtherNet Core backend is written in Java (JDK 11) using the Spring Boot framework, split into distinct layers, each with their own responsibilities.

Configuration layer ensures that EtherNet Core backend is set up correctly during runtime. *Controller layer* serves incoming client requests received via REST API endpoints. *Exception layer* allows the EtherNet Core backend to dictate how certain exceptions should be handled. *Model layer* defines the entities used throughout the EtherNet Core backend. The entities are defined as Java classes with one-to-one mapping to the Hive warehouse tables that are responsible for storing them. EtherNet Core's model layer is defined as protobufs and is generated using a protobuf compiler. *Repository layer* helps the service layer to persist and retrieve data from Hive. It abstracts persistence operations through multiple classes implemented to fetch entities like blocks, contracts, transfers, etc. *Service*

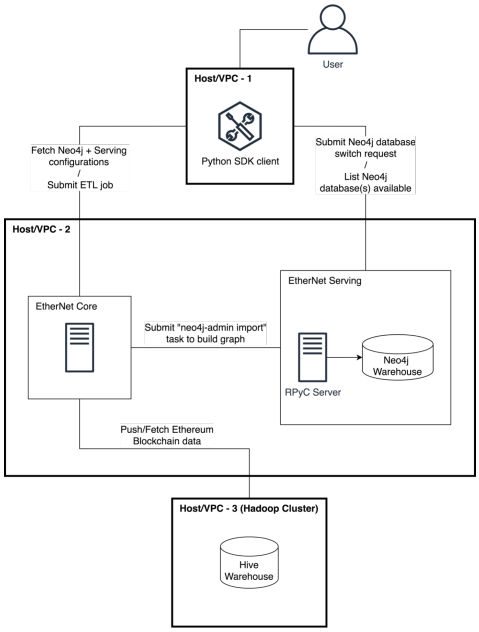


Figure 2: High level application topology of EtherNet

layer encapsulates business logic implementation and centralizes data access. It defines what functionalities are provided and how to access them. *Utility layer* provides static methods for niche tasks that do not need to be executed within the Spring context.

Software components. The EtherNet Serving component is used to help manage and serve Neo4j resources, while the Python SDK is used to interact with EtherNet Core and EtherNet Serving to submit ETL jobs and manage Neo4j graphs respectively. Table 2 shows the components within EtherNet Serving and Python SDK.

Table 2: Components of EtherNet **Serving and Python SDK**

Classes in EtherNet Serving	Description
run_neo4j_import.py	Helper script to invoke neo4j-admin tool to perform batch imports of CSV to create Neo4j graphs.
switch_db.py	Helper script to switch Neo4j databases, executed by running through a request to the RPyC server.
rpyc_classic.py	Script provided when users install the python RPyC module; used to start a classic RPyC server.
Classes in Python SDK	**Description**
get_neo4j_config	Fetches Neo4j connection configuration. Users are able to connect to the Neo4j graph directly and interact with it using Neo4j's Python driver.
get_serving_config	Gets serving connection configuration for the user. Used mainly for debugging purposes.
get_databases	Lists the Neo4j databases that have been created in previous ETL jobs, with data persisted in Hive.
switch_database	Switches to a specific Neo4j database of interest.
create_token_transfers_graph	Creates a token transfers graph from the Ethereum blockchain data in a user-specified block range.
create_traces_graph	Creates a traces graph from the Ethereum blockchain data in a user-specified block range.
create_transactions_graph	Creates a transactions graph from the Ethereum blockchain data in a user-specified block range.

3 DEMONSTRATION PLAN

Objective. EtherNet aims to support languages and environments most users are comfortable with, e.g., Python Notebooks. EtherNet uses protobufs to define the entities and gRPC services. Protobufs allows for new features to be added while ensuring backward compatibility, as well as easy language interoperability since it is implemented in a variety of languages. EtherNet uses Hive as a data warehouse. This allows EtherNet to store large amounts of data in the Hadoop Distributed File System (HDFS), making it a much more scalable solution as compared to traditional RDBMS. The use of Hive affords EtherNet the ability to scale horizontally depending on workloads if configured properly. Fluctuating workloads can be accommodated by spinning datanodes up or down.

Requirements. The requirements from EtherNet were targeted at efficiency and scalability — (i) EtherNet fetches block timestamps from block numbers to control the costs incurred in using BigQuery as a source of truth as well as to reduce eventual query execution time; (ii) EtherNet identifies incremental data required for each query and keeps a copy of the data downloaded for previous ETL jobs to reduce the monetary cost incurred for BigQuery; (iii) EtherNet is capable of persisting Ethereum blockchain data in a Hive database, as well as retrieving the data upon request; (iv) EtherNet allows users to create and store graph representations of the tabular Ethereum blockchain data retrieved from BigQuery in Neo4j format, parsing through the "token_transfers", "traces" and "transactions" tables; (v) The Python SDK allows users to list the Neo4j graph database(s) that have been created in previous ETL jobs and to switch between existing Neo4j graph databases quickly.

3.1 Demonstration using Python Notebook

Visualization. EtherNet provides APIs to support various ETL workflows, creating a preferred type of graph within a specified block range. Once the graph is created, it can be visualized via the Neo4j interface. The respective code is shown in the following snippets, while the output visualization is presented in Figure 3.

```
# Connect to EtherNet Core
from ethernet.client import Client
ec = Client(core_host="192.168.1.99",
            core_grpc_port=9090, core_http_port=8080)

# Create a token_transfers graph given a block range
response = ec.create_token_transfers_graph(2000000, 2000500)

# Switch to the Neo4j graph that has just been created
dbs = ec.get_databases()
ec.switch_database(dbs[0])
```

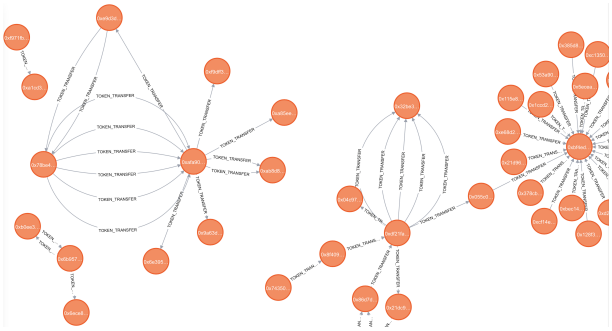
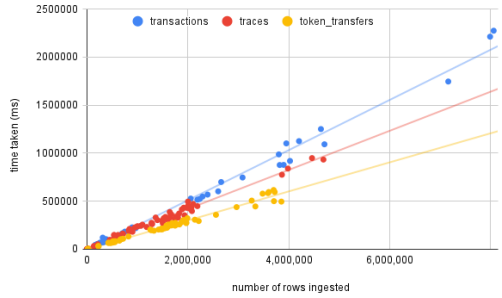
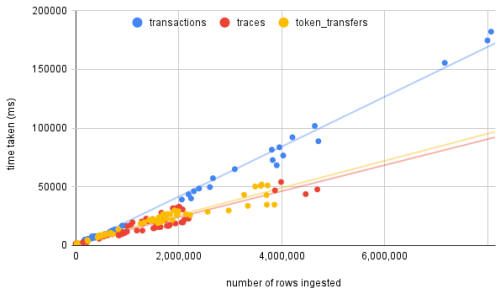


Figure 3: Output visualization for the constructed graph in Neo4J



(a) Time taken to load data into an ORC file



(b) Time taken to insert data into a Hive table

Figure 4: Data ingestion efficiency of EtherNet

Mining. Constructed graphs can be decomposed into smaller subgraphs based on edge features like token_address, block_number, block_timestamp, or node features like connected_component, degree, etc. One may also use Cypher query strings from the Python Notebook for further graph mining in Neo4j. The following Cypher query finds the number of strongly connected components (SCC) and their statistics in our constructed graph. The query computes that out of all addresses (nodes) which are involved in token transfers between blocks 2,000,000 and 2,000,500, there are 67 SCCs; the largest with 4 nodes and the smallest with 1 node.

```
query_string = """CALL gds.alpha.scc.write({nodeProjection:'Address',
    relationshipProjection:'TOKEN_TRANSFER', writeProperty:'componentId'})
YIELD setCount AS set_count,
    maxSetSize AS max_set_size, minSetSize AS min_set_size;"""
dtf_data = pd.DataFrame([dict(_) for _ in conn.query(query_string, db=db)])
```

3.2 System Performance

We test system performance characteristics of EtherNet under a controlled environment and workload, on a server having Intel(R) Xeon(R) CPU X5690 @ 3.47GHz with two cores, 8GB RAM, 100GB disk storage, supported by a Hadoop cluster with 1 namenode and 1 datanode with map-reduce execution mode for Hive.

Ingestion efficiency. The ingestion of data into Hive tables is a multistep process in EtherNet, where (i) the data of interest is downloaded from BigQuery, (ii) the BigQuery data types are mapped to ORC data types, (iii) the transformed data types are written into an ORC file in the HDFS, (iv) a temporary Hive table is created with the ORC location pointing to the ORC file in the HDFS, (v) the data from the temporary Hive table is inserted into the destination Hive table, where it is automatically chunked into smaller ORC files.

To get an idea of the ingestion performance, time taken to load data into an ORC file and time taken to insert data into a Hive table were measured over 196 ingestion cycles, with the number of rows to be ingested varying between 4,000 and 8,000,000. Figures 4(a) and 4(b) indicate that the time required to ingest data grows *linearly* with rows required to be ingested across all the tables.

Storage efficiency. The storage space required for both Hive and BigQuery is shown in Table 3 for the three main blockchain tables. Although BigQuery stores data in a compressed format (Capacitor), the amount of bytes to be read when performing a query is measured by the raw-uncompressed number of bytes required to hold the data in the BigQuery table, as presented in Table 3. Hive is configured to use ORC for storing the Ethereum blockchain data in case of EtherNet, and we find that the storage space needed is significantly lesser with Hive as compared to other options like PostgreSQL or MySQL that do not have compression enabled out of the box. As shown in Table 3, EtherNet compressed the data down to around 8.4% of its original size for traces, 18.1% its original size for token transfers, and 26.9% of its original size for transactions.

Table 3: Storage efficiency of EtherNet **(Hive) vs BigQuery**

Table	Blocks	Number of Rows	Hive	BigQuery
token transfers	0 – 6,825,750	152,587,661	8.00 GB	44.2 GB
traces	0 – 2,463,500	267,789,485	9.72 GB	115.1 GB
transactions	0 – 4,745,000	103,189,720	11.93 GB	44.3 GB

Acknowledgement. AK is supported by Singapore MOE tier-1, 2 grants RG117/19, 2019-T2-2-042.

REFERENCES

[1] N. C. Abay, C. G. Akcora, Y. R. Gel, M. Kantarcioglu, U. D. Islambekov, Y. Tian, and B. M. Thuraisingham. 2019. ChainNet: Learning on Blockchain Graphs with Topological Features. In *ICDM*.
[2] T. Chen, Y. Zhang, Z. Li, X. Luo, T. Wang, R. Cao, X. Xiao, and X. Zhang. 2019. TokenScope: Automatically Detecting Inconsistent Behaviors of Cryptocurrency Tokens in Ethereum. In *SIGSAC*.
[3] T. Chen, Y. Zhu, Z. Li, J. Chen, X. Li, X. Luo, X. Lin, and X. Zhang. 2018. Understanding Ethereum via Graph Analysis. In *INFOCOM*.
[4] A. Day and E. Medvedev. 2018. Ethereum in BigQuery: a Public Dataset for Smart Contract Analytics. https://cloud.google.com/blog/products/data-analytics/ethereum-bigquery-public-dataset-smart-contract-analytics.
[5] S. Ferretti and G. D'Angelo. 2019. On the Ethereum Blockchain Structure: A Complex Networks Theory Perspective. *Concurrency and Computation: Practice and Experience* (2019), e5493.
[6] H. Huang, W. Kong, S. Zhou, Z. Zheng, and S. Guo. 2021. A Survey of State-of-the-Art on Blockchains: Theories, Modelings, and Tools. *ACM Comput. Surv.* 54, 2 (2021), 44:1–44:42.
[7] L. Kiffer, D. Levin, and A. Mislove. 2018. Analyzing Ethereum's Contract Topology. In *Internet Measurement Conference*.
[8] X. T. Lee, A. Khan, S. S. Gupta, Y. H. Ong, and X. Liu. 2020. Measurements, Analyses, and Insights on the Entire Ethereum Blockchain Network. In *WWW*.
[9] Y. Li, U. Islambekov, C. G. Akcora, E. Smirnova, Y. R. Gel, and M. Kantarcioglu. 2020. Dissecting Ethereum Blockchain Analytics: What We Learn from Topology and Geometry of the Ethereum Graph?. In *SDM*.
[10] C. Ma, X. Kong, Q. Lan, and Z. Zhou. 2019. The Privacy Protection Mechanism of Hyperledger Fabric and its Application in Supply Chain Finance. *Cybersecur.* 2, 1 (2019), 5.
[11] S. Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System.
[12] F. E. Oggier, S. Phetsouvanh, and A. Datta. 2018. BiVA: Bitcoin Network Visualization & Analysis. In *ICDM Workshops*.
[13] S. Somin, Y. Altshuler, G. Gordon, A. Pentland, and E. Shmueli. 2020. Network Dynamics of a Financial Ecosystem. *Scientific reports* 10, 1 (2020), 4587.
[14] S. Somin, G. Gordon, and Y. Altshuler. 2018. Network Analysis of ERC20 Tokens Trading on Ethereum Blockchain. In *Complex Systems*.
[15] V. H. Su, S. Sen Gupta, and A. Khan. 2021. EtherNet Codebase. https://github.com/voonhousntu/ethernet/.
[16] F. Victor and B. K. Lüders. 2019. Measuring Ethereum-based ERC20 Token Networks. In *Financial Cryptography and Data Security*.
[17] L. Zhao, S. S. Gupta, A. Khan, and R. Luo. 2021. Temporal Analysis of the Entire Ethereum Blockchain Network. In *WWW*.
[18] Y. Zhao, J. Liu, Q. Han, W. Zheng, and J. Wu. 2020. Exploring EOSIO via Graph Characterization. In *BlockSys*.
[19] P. Zheng, Z. Zheng, J. Wu, and H.-N. Dai. 2020. XBlock-ETH: Extracting and Exploring Blockchain Data From Ethereum. *IEEE Open J. Comp. Soc.* 1 (2020).