ChainDash: An Ad-Hoc Blockchain Data Analytics System

Yushi Liu[1,2,3], Liwei Yuan[2], Zhihao Chen[1,2,3], Yekai Yu[1,2,3],
Zhao Zhang[1,3*], Cheqing Jin[1,3], Ying Yan[2]
[1]School of Data Science and Engineering, East China Normal University
[2]Blockchain Platform Division, Ant Group
[3]Engineering Research Center of Blockchain Data Management, Ministry of Education
{ysliu, chenzh, ykyu}@stu.ecnu.edu.cn, {zhzhang, cqjin}@dase.ecnu.edu.cn
{yuanliwei.ylw, fuying.yy}@antgroup.com

ABSTRACT

The emergence of digital asset applications, driven by Web 3.0 and powered by blockchain technology, has led to a growing demand for blockchain-specific graph analytics to unearth the insights. However, current blockchain data analytics systems are unable to perform efficient ad-hoc graph analytics over both live and past time windows due to their inefficient data synchronization and slow graph snapshots retrieval capability. To address these issues, we propose ChainDash, a blockchain data analytics system that dedicates a highly-parallelized data synchronization component and a retrieval-optimized temporal graph store. By leveraging these techniques, ChainDash supports efficient ad-hoc graph analytics of smart contract activities over arbitrary time windows. In the demonstration, we showcase the interactive visualization interfaces of ChainDash, where attendees will execute customized queries for ad-hoc graph analytics of blockchain data.

PVLDB Reference Format:
Yushi Liu, Liwei Yuan, Zhihao Chen, Yekai Yu, Zhao Zhang, Cheqing Jin, and Ying Yan. ChainDash: An Ad-Hoc Blockchain Data Analytics System. PVLDB, 16(12): 4022 - 4025, 2023.
doi:10.14778/3611540.3611611

1 INTRODUCTION

Web 3.0 [8] is dedicated to building the Internet of Value based on blockchain technology, which significantly advances the digitization of assets and related applications. The on-chain transactional data, produced by user interactions with these applications, is transparent and can be used for KYT (Know Your Transaction) services, etc., thus having immense analytical value. Currently, it is practical to synchronize on-chain data to off-chain stores for analytics. Still, two requirements need to be fulfilled before enabling efficient ad-hoc graph analytics over arbitrary time windows of interest: 1) high data synchronization efficiency, which guarantees the freshness of real-time data; 2) optimal graph snapshots retrieval performance, which accelerates the ad-hoc graph query processing. To illustrate the former, for example, Topnode [1] (known as Jingtan in Chinese) is a digital collection platform supported by AntChain [2]. To fulfill massive blockchain data query requests, Topnode pulls on-chain data to OceanBase [2] and retrieves query results through database interfaces. The flash sale of digital collections on Topnode is a typical scenario that requires highly efficient data synchronization, without which users will suffer significant delays in obtaining real-time data, ultimately affecting their experience. Regarding the latter, we illustrate KYT services utilized in public blockchain analytics scenarios. The KYT service is a crucial tool to assist financial institutions and exchanges to detect abnormal behaviors for anti-money laundering and counter-terrorism financing, it requires ad-hoc graph analytics to identify security incidents, which may involve a significant number of highly correlated abnormal transactions within a specific period of time. In such scenarios, blockchain-specific analytics systems necessitate strong multi-versioned graph retrieval ability, as such analytics needs to access graphs over arbitrary time windows.

However, existing blockchain data analytics systems [1, 5, 6, 9] typically synchronize data in a sequential and blocked manner. Transaction receipts are pulled block-by-block quiescently and replayed one after another accordingly, thus failing to provide real-time data freshness. In addition, prior works on multi-versioned graph stores [4, 7, 10] rarely address the retrieval of graph snapshots that are larger-than-memory. Retrieving out-of-core graphs from unoptimized underlying storage causes significant overhead, precluding their use in ad-hoc graph analytics over historical windows. To summarize, the inefficient data synchronization and limited graph snapshots retrieval capability hinder the efficiency of ad-hoc graph analytics over both live and past time windows.

To address the above limitations, we design and implement ChainDash, a blockchain data analytics system that facilitates efficient ad-hoc graph analytics of blockchain data. ChainDash functions as a middleware system comprising two key building blocks: a highly-parallelized data synchronization component and a retrieval-optimized temporal graph store. The former is responsible for extracting transaction receipts from the blockchain, obtaining event logs, and replaying these logs to produce consistent state transitions of smart contracts with the blockchain. Meanwhile, we incorporate fine-grained optimizations for replaying to enable highly-parallelized data synchronization. The synchronized states and their interactions are then converted into user-defined graph representations and stored in the temporal graph store. The store balances fast ingestion with efficient indexing for temporal blockchain data based on an append-only storage layout. Within each epoch, an approximate index is built to improve the performance of multi-versioned graph retrieval, thus serving ad-hoc graph queries efficiently. To the

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611611

[1]https://antchain.antgroup.com/products/acdc

[2]https://www.oceanbase.com/

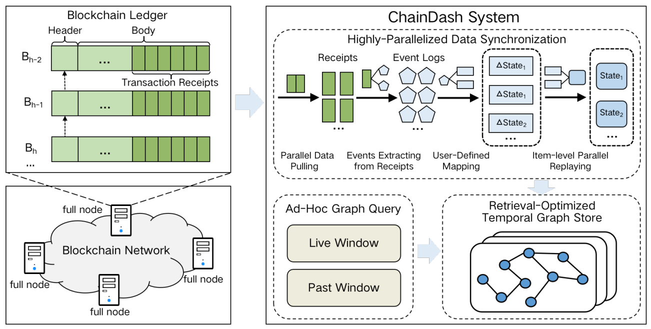


Figure 1: The Architecture of ChainDash

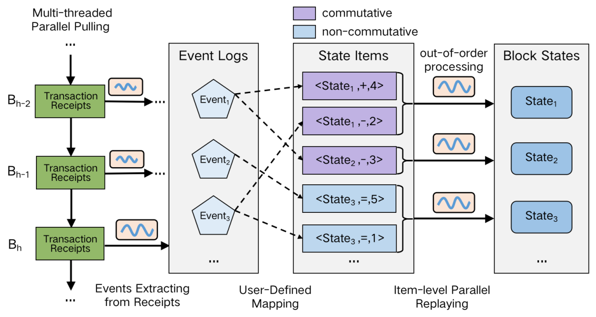


Figure 2: Highly-Parallelized Data Synchronization

best of our knowledge, ChainDash is the first systematic solution to support efficient ad-hoc graph analytics of blockchain data.

The main contributions of our work are summarized as follows:

- We propose ChainDash, an ad-hoc blockchain data analytics system that rapidly synchronizes on-chain data to our meticulously designed off-chain graph store, allowing users to perform efficient blockchain-specific graph analytics over arbitrary time windows.
- We present a highly-parallelized data synchronization component combining parallel pulling and item-level parallel replaying to ensure efficient off-chain data synchronization. In addition, we design a retrieval-optimized temporal graph store that exploits epoch-based indexing for effective graph snapshots retrieval.
- We demonstrate how ChainDash leverages efficient graph analytics capability to support the overview of on-chain activities and the detection of abnormal behaviors.

In the remainder of the paper, we introduce the system architecture and two key components of ChainDash in Section 2, demonstrate the visualization interfaces and interaction options with two scenarios in Section 3, and conclude with a summary in Section 4.

2 CHAINDASH OVERVIEW

2.1 System Architecture

The ChainDash architecture embraces a middleware design consisting of two main components, as depicted in Figure 1.

Highly-Parallelized Data Synchronization. ChainDash designs a highly-parallelized data synchronization component that continuously pulls transaction receipts from trusted blockchain nodes in a multi-threaded manner. The transaction receipts represent internal activities within smart contracts and contain event logs that can be replayed to get the same on-chain state transitions for the off-chain data store. ChainDash enables replaying at the item-level granularity, which provides high parallelism while ensuring block-level determinism and consistency. With this component, ChainDash can efficiently synchronize blockchain data and ensure real-time data freshness.

Retrieval-Optimized Temporal Graph Store. The states and interactions are reorganized into the graph format, where each vertex represents an address state and each edge represents their interaction. For persistence, vertices and edges are encoded as key-value pairs. Each key contains a vertex identifier and an embedded block height, while the value holds the block-level state value and a list of related edges in the same block. ChainDash uses an append-only storage layout to persist the temporal data. Based on this layout, ChainDash logically partitions files into epochs and builds an approximate index structure for each epoch to quickly filter and determine the existence of intra-epoch data, thereby providing effective graph retrieval for window-based ad-hoc graph analytics.

2.2 Highly-Parallelized Data Synchronization

The highly-parallelized data synchronization component, as shown in Figure 2, leverages parallelizing techniques to improve the efficiency of two phases: 1) data pulling phase, and 2) state replaying phase. ChainDash implements a multi-threaded parallel pulling mechanism to optimize data pulling and furnish ample input data for the next phase. During the state replaying phase, ChainDash extracts event logs from transaction receipts, maps them into state items, and subsequently replays the logs to produce block-level state values of smart contracts involved in the block.

The conventional replaying approach sequentially replays transaction receipts, adhering to the order and dependency constraints defined by each block. To break through this limitation, ChainDash designs a DAG-based parallel replaying method. By analyzing read-write dependencies and determining the serialization order of conflicting receipts, ChainDash constructs a conflict graph to achieve transaction-level parallel state replaying. Moreover, to guarantee robust replaying parallelism in high-contention scenarios, ChainDash implements a finer-grained item-level parallel replaying, which replays items (instead of transaction receipts) by relaxing the transaction-level ACID requirements while setting synchronization barriers to ensure the complementary block-level data consistency. In particular, ChainDash classifies the state items into two categories: commutative and non-commutative (as exemplified by *State Items* in Figure 2). For commutative items, they can be processed in parallel and out-of-orderly since their effects do not depend on the processing order and will not affect block-level consistency. For non-commutative items, we ensure that they are processed in the order specified by the conflict graph. The item-level parallel replaying ensures high replaying parallelism while guaranteeing off-chain data consistency and freshness.

For the empirical evaluation, we select 200K transfer events from the USDT smart contract on Ethereum as data to be synchronized. We measure cumulative data synchronization time (from data replaying/extracting to persistence, excluding asynchronous pulling) of ChainDash, The Graph [1] and Ethereum-ETL[3] to evaluate the performance of data synchronization. The results shown in Table 1 demonstrate that ChainDash achieves superior data synchronization efficiency, saving up to 82.9% and 48.9% process time compared to the other two synchronization solutions, respectively.

Table 1: Synchronization Efficiency

Method	ChainDash	The Graph	Ethereum-ETL
Process Time (s)	9.1	53.1	17.8

2.3 Retrieval-Optimized Temporal Graph Store

The retrieval-optimized temporal graph store aims to support efficient graph snapshots retrieval for ad-hoc graph queries. As shown in Figure 3, the store comprises two key techniques: 1) an append-only storage layout and 2) an epoch-based index structure. Regarding the former, as blockchain-specific records include the block height as a version prefix, they are naturally ordered and multi-versioned. Therefore, ChainDash uses an append-only storage layout to sequentially store the temporal data, without the need for background compaction to merge redundant records and reorder. Each file of the store holds temporal data within a specific block height range, and multiple consecutive files logically belong to an epoch. In addition, we design a built-in index for each epoch to improve the performance of intra-epoch data indexing.

The window-based graph queries necessitate quickly determining the existence of neighbor vertices within the queried version range. However, maintaining the filtering structure per file is inefficient for such queries, as the need to traverse each filter within the window results in costly overhead. Therefore, as for the second key technique, the store builds an index for each epoch, which adopts a searching data structure [3] combining Count-Min Sketch and Bloom Filter to support fast and space-efficient filtering with a low false positive rate. The intra-epoch index is composed of R tables, and each table housing B partitions, each of which contains a bloom filter and a File_Ids set. To maintain the index for an epoch, the newly-flushed file is mapped to a partition through a hash function. The existence information of all the involved vertices is then recorded in the corresponding bloom filter, while the file identifier is stored in the File_Ids set. This process is repeated R times due to the presence of R tables. Based on this index structure, we can efficiently search which files contain the queried vertex within the epoch by traversing the bloom filters in the table, and obtain the result file set by taking the union of the results returned by the hit bloom filters. Then, we perform the same operation on the remaining $R-1$ tables and take the intersection of all File_Ids sets to obtain the result more precisely. To summarize, by employing the append-only store with epoch-based indexing, ChainDash enables efficient graph snapshots retrieval, which is beneficial for ad-hoc graph queries over arbitrary time windows.

To evaluate the performance of window-based graph queries, we construct graphs comprising USDT transfer events from the 13th million block to the 17th million block on Ethereum within ChainDash and compare it with Nebula [4], a commercial graph

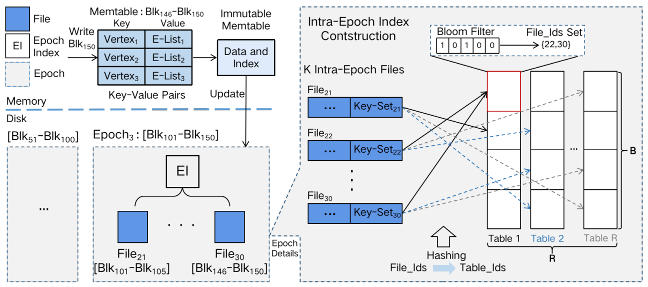


Figure 3: Retrieval-Optimized Temporal Graph Store

database utilizing RocksDB [5] as underlying storage. To ensure a fair comparison, we deploy them with the same single-machine configuration, and we measure the K-hop query performance within the window size of 1000K blocks. The empirical results are shown in Table 2, where p50 indicates the query latency that ranks at 50% of the whole dataset (similarly for p90). ChainDash outperforms Nebula, with an average latency saving of 34% for 1-hop queries, and shows a more evident advantage with average latency savings of 47% and 41% respectively for queries with 3-hop and 5-hop.

Table 2: K-hop Query Performance

K-hop Queries	p50 (ms) / Result Size (K rows)		p90 (ms) / Result Size (K rows)	
	Nebula	ChainDash	Nebula	ChainDash
1-hop	7.2 / 2.6	4.8 / 2.1	27.7 / 10.8	18.2 / 11.3
3-hop	671.4 / 215.5	369.8 / 181.6	6.6K / 1000.3	3.3K / 1000.3
5-hop	77.0K / 8999.5	47.7K / 10227.4	176.1K / 13326.4	100.3K / 13033.4

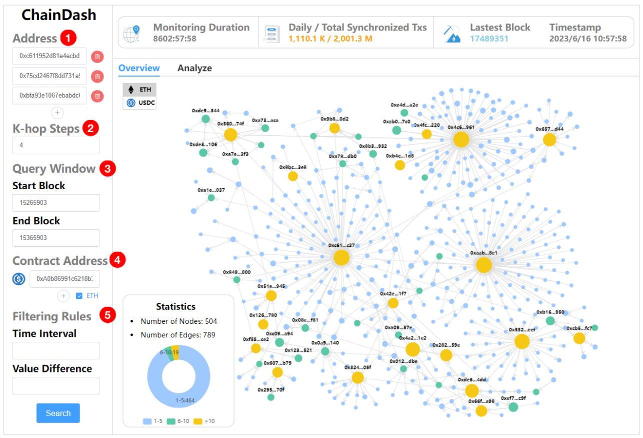
3 DEMONSTRATION

In this demonstration, we showcase two scenarios. Scenario 1 is an on-chain activity overview, where attendees execute K-hop graph queries and obtain an overview of the on-chain internal activities of an interested smart contract. Scenario 2 is the abnormal behavior analytics, where attendees screen out abnormal behaviors among on-chain activities and track the flow of funds step by step.

3.1 Scenario 1: On-Chain Activity Overview

Figure 4 shows the ChainDash visualization interfaces, which enable attendees to view statistical information from the dashboard and execute customized graph queries by filling out the form. The form contains several input fields, including ❶ "Address" for specifying the address(es) to be queried, ❷ "K-hop Steps" for screening out the data within k hops of the center address, ❸ "Query Window" for constraining the time range, ❹ "Contract Address" for specifying the monitored smart contract address(es), and ❺ "Filtering Rules" for defining filtering conditions for anomaly detection. For instance, given a starting vertex $addr_1$, to query its k-hop transaction network graph related to $contract_1$, between $start\_block$ and end_block, the following steps will be included: **(i)** Attendees first input the following parameters: $(addr_1, k, start\_block, end\_block, contract_1)$, and click the **"Search"** button; **(ii)** ChainDash executes the k-hop query and generates an overview of internal smart contract activities in

[3]https://github.com/blockchain-etl/ethereum-etl/
[4]https://www.nebula-graph.io/
[5]https://github.com/facebook/rocksdb



ChainDash

Monitoring Duration 8602:57:58 | Daily / Total Synchronized Txs 1,110.1 K / 2,001.3 M | Lastest Block 17489351 | Timestamp 2023/6/16 10:57:58

Address: 0xc611952d81e4ecbd, 0x75cd2467f8dd731a!, 0xbfa93e1067ebabdc(

K-hop Steps: 4

Query Window — Start Block: 15265903; End Block: 15365903

Contract Address: 0xA0b86991c6218b; ETH

Filtering Rules — Time Interval: ; Value Difference:

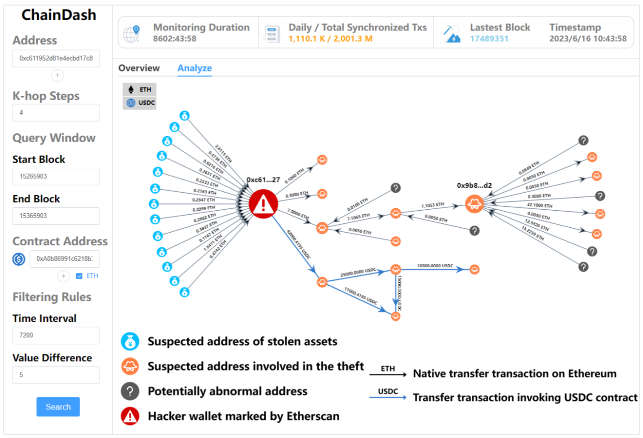
Statistics
- Number of Nodes: 504
- Number of Edges: 789

Figure 4: On-Chain Activity Overview

the "**Overview**" view; **(iii)** Attendees interact with the generated graph to obtain specific information. As shown in Figure 4, the queried result is represented as a network consisting of multiple vertices and edges, where the vertices represent addresses and the edges represent the activities between two addresses. In addition, the pie chart in the bottom left corner of the view displays the distribution of vertices with different degrees (number of associated activities) in the network.

3.2 Scenario 2: Abnormal Behavior Analytics

In the on-chain activity overview mentioned, we note that some addresses have a high volume of transactional activities in a short time, and more detailed analytics for them are required. In this case, filtering rules enable customized anomaly detection from various perspectives like time range and transfer amount. In Scenario 2, we apply the "Filtering Rules" to an address identified as "hacker wallet" (marked by ⚠) on Ethereum to screen out abnormal activities within k hops. Attendees can progressively experience it by following these steps: **(i)** Attendees first define a filter ($\Delta_{blockNumber}$, Δ_{value}), and activities meeting the criteria of receiving multiple assets and transferring them out within a block range of $\Delta_{blockNumber}$, with a difference between outgoing and incoming amounts not exceeding Δ_{value}, will be marked by the filter; **(ii)** Subsequently, by clicking the "Search" button, an abnormal behavior network graph is generated in the "**Analyze**" view. Figure 5 presents the output of anomaly detection on ETH and USDC transfer activities associated with the "hacker wallet"; **(iii)** As different address types are marked with different icons and transaction information is labeled, attendees can easily observe the flow of funds. In Figure 5, the 💰 mark indicates the wallet's theft, the 🕵 mark signifies the address flagged as the hacker by our filtering rules, and the ? mark represents addresses involved in abnormal activities whose identity is obscure. Overall, the efficient versioned graph data retrieval and graph analytics mechanisms supported by ChainDash contribute to the timely detection of abnormal behaviors and tracking of fund flows. In some illegal activities such as money laundering and fraud, this helps to effectively intervene before funds are involved in coin mixing or cross-chain transactions, thereby avoiding greater economic losses.



ChainDash

Monitoring Duration 8602:43:58 | Daily / Total Synchronized Txs 1,110.1 K / 2,001.3 M | Lastest Block 17489351 | Timestamp 2023/6/16 10:43:58

Address: 0xc611952d81e4ecbd17c8

K-hop Steps: 4

Query Window — Start Block: 15265903; End Block: 15365903

Contract Address: 0xA0b86991c6218b; ETH

Filtering Rules — Time Interval: 7200; Value Difference: 5

Figure 5: Abnormal Behavior Analytics

4 CONCLUSION

We propose ChainDash, a blockchain-specific data analytics system that supports efficient ad-hoc graph analytics for applications like KYT services. By parallelizing the data synchronization and optimizing the graph snapshots retrieval ability, ChainDash can effectively meet the demands for ad-hoc graph analytics over arbitrary time windows. In the demonstration, we showcase ChainDash with two scenarios based on real-world data from Ethereum, where attendees are able to query on-chain internal activities of smart contracts through a visualization interface and further analyze abnormal behaviors using ad-hoc graph analytics.

ACKNOWLEDGMENTS

This work was supported by National Science Foundation of China (No. 61972152) and Program of Shanghai Academic/Technology Research Leader (No. 23XD1401100).

REFERENCES

[1] The Graph. 2023. The Graph. https://thegraph.com/
[2] Ant Group. 2023. AntChain. https://antchain.antgroup.com/
[3] Gaurav Gupta, Minghao Yan, Benjamin Coleman, Bryce Kille, Ryan A. Leo Elworth, Tharun Medini, Todd J. Treangen, and Anshumali Shrivastava. 2021. Fast Processing and Querying of 170TB of Genomics Data via a Repeated And Merged BloOm Filter (RAMBO). In *SIGMOD Conference*. ACM, 2226–2234.
[4] Xiaoen Ju, Dan Williams, Hani Jamjoom, and Kang G. Shin. 2016. Version Traveler: Fast and Memory-Efficient Version Switching in Graph Processing Systems. In *USENIX Annual Technical Conference*. USENIX Association, 523–536.
[5] Harry A. Kalodner, Malte Möser, Kevin Lee, Steven Goldfeder, Martin Plattner, Alishah Chator, and Arvind Narayanan. 2020. BlockSci: Design and applications of a blockchain analysis platform. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020.* USENIX Association, 2721–2738.
[6] Yang Li, Kai Zheng, Ying Yan, Qi Liu, and Xiaofang Zhou. 2017. EtherQL: A Query Layer for Blockchain System. In *DASFAA (2) (Lecture Notes in Computer Science)*, Vol. 10178. Springer, 556–567.
[7] Peter Macko, Virendra J. Marathe, Daniel W. Margo, and Margo I. Seltzer. 2015. LLAMA: Efficient graph analytics using Large Multiversioned Arrays. In *ICDE*. IEEE Computer Society, 363–374.
[8] Gavin Wood. 2017. ĐApps: What Web 3.0 Looks Like. https://gavwood.com/dappsweb3.html
[9] Haotian Wu, Zhe Peng, Songtao Guo, Yuanyuan Yang, and Bin Xiao. 2021. VQL: efficient and verifiable cloud query services for blockchain systems. *IEEE Transactions on Parallel and Distributed Systems* 33, 6 (2021), 1393–1406.
[10] Xiaowei Zhu, Marco Serafini, Xiaosong Ma, Ashraf Aboulnaga, Wenguang Chen, and Guanyu Feng. 2020. LiveGraph: A Transactional Graph Storage System with Purely Sequential Adjacency List Scans. *Proc. VLDB Endow.* 13, 7, 1020–1034.