AlterEgo: A Dedicated Blockchain Node For Analytics

Qi Guo[1,*], Mahdi Alizadeh[2,†], Ali Falahati[2,†] and Laurent Bindschaedler[2]

[1]MPI-INF, Campus E 1 4, Saarbrücken, Germany
[2]MPI-SWS, Campus E 1 5, Saarbrücken, Germany

Abstract

Blockchains today amass terabytes of transaction data that demand efficient and insightful real-time analytics for applications such as smart contract hack detection, arbitrage on decentralized exchanges, or trending token analysis. Conventional blockchain nodes, constrained by RPC APIs, and specialized ETL-based blockchain analytics systems grapple with a trade-off between materializing pre-calculated query results and analytical expressiveness. In response, we introduce AlterEgo, a blockchain node architected specifically for analytics that maintains parity with traditional nodes in ingesting consensus-produced blocks while integrating a robust analytics API. Our prototype supports efficient transactional and analytical processing while circumventing the rigidity of ETL workflows, offering a better trust model, and achieving significant performance improvements over the state-of-the-art.

Keywords

Blockchain Analytics, Real-Time Analytics, Blockchain Node Architecture, HTAP, ETL, Decentralized Analytics

1. Introduction

Decentralized Ledger Technology (DLT) is rapidly gaining traction across various industries. As of December 2023, Ethereum, one of the leading blockchains, contains over one terabyte of transaction data, growing at a rate of 1 million transactions per day [1]. As a result, the need for efficient and low-latency blockchain analytics systems has become increasingly paramount. While adept at handling transactional data, traditional blockchain nodes fail to provide comprehensive analytics capabilities. These nodes typically offer Remote Procedure Call (RPC) APIs, allowing clients to access detailed information about transactions, states, and events. However, these APIs cannot handle complex queries essential for in-depth analytics, such as filters, joins, and aggregation.

The prevailing approach in blockchain analytics involves executing Extract, Transform, and Load (ETL) processes from a blockchain node. This paradigm first entails extracting fine-grained information from the blockchain node and processing the data in external analytics systems, which presents several challenges. First, it demands extensive resources to extract and store the data. Secondly, it introduces considerable latency in data updates. It also creates a dependency on the blockchain node serving the raw data as a single source of truth. Finally, it requires knowing the specific data to be queried in advance, making it impossible to respond to unexpected queries without initiating another ETL process to collect the missing data. Overall, these issues contribute to system inefficiency and suboptimal user experience.

This paper proposes a novel blockchain analytics solution, *AlterEgo*, representing a radical shift from the current ETL paradigm. AlterEgo is a blockchain node that mirrors the functionality of a standard node but is *inherently designed to support analytics workloads.* It maintains a copy of the entire blockchain, functioning as an archive node[1], and directly ingests new blocks produced by the consensus mechanism instead of extracting them from a single blockchain node via RPC. Aside from adding an analytics API, AlterEgo nodes are indistinguishable from traditional nodes from the perspective of the blockchain system: they can participate in consensus, relay transactions, and validate blocks. As a result, AlterEgo nodes provide additional redundancy and increase the decentralization of the underlying blockchain. What sets AlterEgo apart is its internal architecture, which revolves around a columnar-vectorized store that supports analytical query workloads and fast data ingestion.

To validate our concept, we have developed a prototype of AlterEgo and conducted a comprehensive evaluation of its performance. We compare our approach against traditional blockchain RPCs and the leading solution in blockchain analytics, The Graph Protocol [2]. Our findings demonstrate that AlterEgo significantly outperforms current solutions while supporting low-latency

DOALP '24: 26th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, March 25, 2025, Paestum, Italy
*Corresponding author.
†These authors contributed equally.
qiguo@mpi-inf.mpg.de (Q. Guo); malizade@mpi-sws.org (M. Alizadeh); afalahat@mpi-sws.org (A. Falahati); bindsch@mpi-sws.org (L. Bindschaedler)
https://jaden-qi-guo.github.io (Q. Guo); https://alizademhdi.github.io/ (M. Alizadeh); https://ali-falahati.github.io/ (A. Falahati); https://binds.ch (L. Bindschaedler)
0009-0008-5338-8121 (Q. Guo); 0009-0003-7830-5541 (M. Alizadeh); 0009-0001-3064-4187 (A. Falahati); 0000-0003-0559-631X (L. Bindschaedler)
© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

[1]An archive node stores a copy of the blockchain since inception and allows querying the state at any block height.

data updates, more expressive querying capabilities, and a superior trust model for data integrity and provenance.

This paper makes the following contributions:

- We introduce AlterEgo, a new kind of blockchain node for analytics that integrates seamlessly into existing blockchain environments.
- We present the design of AlterEgo and highlight its advantages over the state-of-the-art.
- We implement a prototype of AlterEgo and demonstrate its significant performance improvements and increased flexibility.

2. Background & Motivation

2.1. Blockchains & Smart Contracts

Blockchains have evolved from their original role as platforms for simple transactional exchanges to becoming complex ecosystems supporting smart contracts with intricate states and interactions. Smart contracts are written directly into code, residing on the blockchain, and can interact with other contracts and user accounts.

In this paper, we focus on blockchains compatible with the Ethereum Virtual Machine (EVM), e.g., Ethereum [3], Polygon [4], and Arbitrum [5], but our ideas broadly extend to other systems. Transactions in EVM blockchains execute smart contracts. When a smart contract runs, it can emit events logged within the blockchain. These events are crucial for tracking the activity of smart contracts and state changes. Furthermore, each transaction generates a receipt, which provides essential details about the transaction, including its status, the gas used for its execution, and the logs of emitted events.

The complexity and richness of blockchain interactions require advanced analytics to understand, monitor, and optimize the performance and security of blockchain applications and facilitate decision-making.

2.2. Blockchain Analytics

Blockchains are fundamentally designed to support online transaction processing (OLTP) and, therefore, require integrating online analytics processing (OLAP) for effective analytics. Unsurprisingly, current blockchain analytics systems revolve around ETL processes that transfer blockchain data to an off-chain data management system for further processing. EtherQL [6] is a query infrastructure for Ethereum, offering a RESTful API that supports range and limit queries. Blockchain ETL [7] provides a collection of public datasets in relational format stored in Google BigQuery [8]. BlockSci [9] is an in-memory blockchain analytics database capable of importing data from multiple blockchains that offers a domain-specific language (DSL) for specifying graph queries. The Graph Protocol [2] is a commercial solution that lets programmers select a subset of the blockchain data to extract and query using a GraphQL interface.

However, while these systems provide efficient offline blockchain analytics, they face inherent limitations in real-time data analytics. Specifically, the ETL process introduces synchronization delays and requires trusting the blockchain nodes that provide the original data, raising concerns about the data's integrity and provenance.

2.3. An Integrated Blockchain Node

This paper advocates for a different approach to blockchain analytics that addresses the inherent limitations of current systems. The primary issue with existing solutions lies in using an explicit ETL process between two separate systems, creating a disconnect between them. Instead, we propose a unified system paralleling the Hybrid Transaction Analytics Processing (HTAP) architecture, which combines blockchain and analytics functionalities into a single system. This integrated approach removes synchronization challenges and enables low update latency while reducing trust assumptions by sourcing transactions directly from the consensus layer.

3. Design & Implementation

In this section, we describe the system design and implementation details of AlterEgo. AlterEgo focuses on achieving two primary goals: low-latency synchronization of blockchain data to enable real-time analytics and efficient and expressive data analytics supporting arbitrary queries on any data stored in the node.

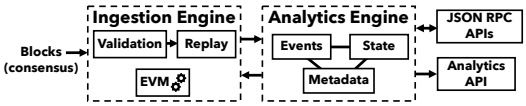

Figure 1: High-level architecture of an AlterEgo node.

The architecture of AlterEgo, shown in Figure 1, follows from these design goals. The system ingests blocks, validates and replays transactions, and inserts the corresponding data into an analytics store that exposes a dedicated analytics API in addition to the standard JSON RPC APIs. In the rest of the section, we present the two main components of AlterEgo in more detail before discussing its implementation.

3.1. Ingestion Engine

AlterEgo receives blocks directly from the blockchain's consensus layer as it runs a complete blockchain node. Each block contains a set of transactions representing state changes within the blockchain, from value transfers to smart contract invocations. The contents of each block are verified and agreed upon by the network participants, providing data integrity.

Upon receipt of a block, the system proceeds, much like a conventional blockchain node, with the validation and state replaying of each transaction. Validation ensures that each transaction adheres to the network rules and does not violate any invariants, such as preventing

double-spending. State replaying entails executing transactions to generate event logs and compute the new state of the blockchain. AlterEgo performs this replay in parallel for efficiency by determining the serialization order of conflicting transactions whenever possible.

Finally, AlterEgo stores the block and transaction metadata, the event logs, and the state changes in its embedded analytics database. Blocks and transactions can be batched together for efficient insertion. AlterEgo dynamically adjusts the batch size to balance performance and data freshness.

3.2. Analytical Engine

The analytical engine utilizes a columnar-vectorized store focused on efficient data retrieval to handle the vast and growing datasets in blockchain environments. Table 1 shows the schema for the central table. The engine indexes and stores all the contents of the blockchain, akin to the functioning of an archive node, enabling arbitrary analytics involving any smart contract or state within the blockchain.

Column	Data Type	Description
`log_id`	BIGINT	Log identifier
`block_num`	BIGINT	Block number (sequential)
`contract`	CHAR(40)	Contract address
`signature`	CHAR(64)	Event siganture
`block_ts`	TIMESTAMP	Block timestamp
`from_addr`	CHAR(40)	Sender address
`to_addr`	CHAR(40)	Receiver address
`data`	BLOB	Data field

Table 1
Central table's schema. `log_id` is the primary key. We add indexes on all fields except `data`.

AlterEgo's analytics API is based on SQL, enabling users to execute complex queries with familiar SQL syntax. In addition to standard SQL, we support window functions and common table expressions to enhance data analysis capabilities and facilitate working with time, an important requirement for blockchain analytics.

3.3. Implementation

We chose the official Ethereum node implementation in Golang, known as Geth [10], as the foundation for AlterEgo for prototyping efficiency, modifying ~270 lines of code. This decision enabled us to focus on specific enhancements, namely integrating our analytics storage engine and adding support for the analytics API, while relying on Geth's efficient transaction validation and state updates. While our prototype utilizes Geth, our approach can support other EVM-compatible blockchains with minimal adjustments. The analytical component of AlterEgo is based on DuckDB [11], an embedded OLAP database that aligns well with the needs of our system while providing satisfactory performance.

4. Evaluation

In this section, we conduct an evaluation of AlterEgo using several representative query benchmarks executed over the Ethereum blockchain. We focus on providing answers to the following research questions.

RQ1: How does the performance of AlterEgo compare with traditional RPC-based analytics and state-of-the-art solutions such as The Graph Protocol?
RQ2: How fast can AlterEgo apply updates?
RQ3: How expressive is AlterEgo when compared with existing solutions?

4.1. Experimental Setup

Benchmarks We consider four representative queries:

Q1: Count the number of transfers of an ERC20[2] token (USDT) over a given period.
Q2: List the k most active addresses of an ERC20 token (USDT) over a given period.
Q3: List the k addresses with the largest balance of an ERC20 token (USDT) at any given time.
Q4: Calculate the total trading volume of a trading pair (USDT-ETH) on a decentralized exchange (Uniswap v2).

Baselines We compare with the following baselines:

- **Golang+RPC**: Collect the data necessary from the query from a local Ethereum blockchain node (Geth version 1.12.1) using JSON-RPC and process it in golang (version 1.21.3). For each query, we execute `eth_getLogs` iteratively on successive block ranges with the appropriate event filter on the relevant smart contract(s) to extract the necessary data before processing it in a custom golang program.
- **The Graph**: Execute the query in The Graph Protocol's GraphQL query format. We run a dedicated subgraph for each query inside a local The Graph deployment (version 0.33.0) synchronized with a local Ethereum node (Geth version 1.12.1).

Configuration We perform all experiments on a server equipped with four Intel(R) Xeon(R) E7-8857 v2 CPUs with a total of 48 cores and 1.5 TB DRAM, running Debian with Linux kernel version 5.15.130.1.amd64-smp.

4.2. Overall Performance Comparison

We begin with a performance comparison of AlterEgo with Golang+RPC and The Graph on each of the four queries (Q1-4) for different time ranges from 1k blocks to 100k blocks, as shown in Figure 2. To complete the comparison, we also include a version of AlterEgo where the query results are materialized (AlterEgoMat) similarly to The Graph. Each block range starts from the 10,000,000th Ethereum block.

The results show that using Golang+RPC for complex queries across extensive block ranges is impractical due

[2]ERC20 is the standard for fungible tokens on Ethereum.

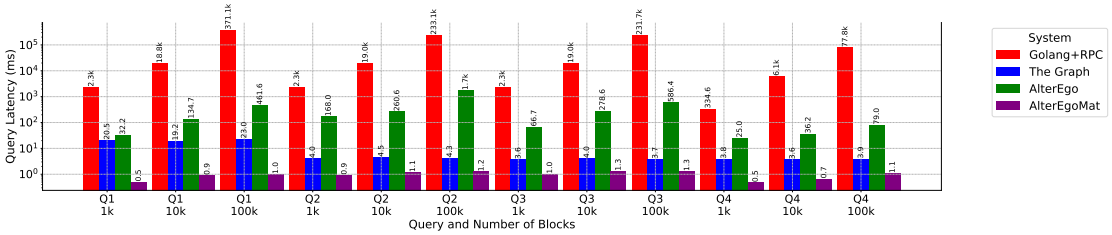

Figure 2: Comparison of query latency for the different systems. Note that the y-axis is in logscale for plot readability.

to the round-trip overheads and excessive intermediate data. In contrast, AlterEgo consistently delivers fast responses of under 2 seconds to all queries and shows better scalability as the query range expands. The Graph excels in handling these predefined queries, achieving impressive performance by pre-computing all results. Finally, AlterEgoMat, which adopts a similar strategy of materializing query results, achieves response times of ~1 ms, surpassing The Graph. Overall, the results underscore AlterEgo's efficiency and indicate a potential to optimize performance for known queries.

4.3. Update Latency

Next, we consider the data update latency (or analytical latency), i.e., the time between the receipt of a transaction and the point at which the new data is available for queries of the different systems. We use different batch sizes by grouping blocks and applying them together to update the state of the analytics system. Table 2 compares the average per-block update latency for AlterEgo and The Graph across various batch sizes. Notably, The Graph's functionality is confined to indexing a single smart contract. To establish a comparison under equivalent conditions, we present data for a similar workload in AlterEgo, referred to as 'AlterEgo Single,' which contrasts with 'AlterEgo Full,' which indexes the entire blockchain. We perform this experiment starting from the 10,000,000th Ethereum block.

Batch Size	AlterEgo Full	AlterEgo Single	The Graph
1	91.70 ms	14.49 ms	31.16 ms
10	71.30 ms	12.64 ms	28.18 ms
100	66.85 ms	11.01 ms	30.87 ms
1,000	65.30 ms	10.73 ms	31.17 ms
10,000	64.00 ms	10.61 ms	31.20 ms

Table 2
Average Update Latency (in milliseconds) for AlterEgo and The Graph with Different Batch Sizes (number of blocks).

AlterEgo Single ingests new blocks up to 3× faster than The Graph. Despite the increased workload undertaken by processing entire blocks, AlterEgo Full applies updates in a timeframe of only 2-3× longer than The Graph. These findings underscore the practicality of ingesting whole blocks within a 100-ms window, given that this time frame is shorter than the network propagation delay for blocks on public blockchains. We conclude that AlterEgo can maintain low update latency, even when processing data on a block-by-block basis.

4.4. Expressiveness

AlterEgo, RPC nodes, and The Graph present distinct paradigms influencing their expressiveness. AlterEgo employs a schema-based query model akin to traditional blockchain nodes, offering flexibility to formulate queries across various parameters and supporting ad hoc querying for evolving analytical needs. The queries in §4.1 are readily expressible in AlterEgo and benefit from its efficient execution. In contrast, while sharing some of this expressive capacity, traditional RPC nodes incur a significant efficiency trade-off due to their inability to perform advanced filtering and aggregation. They also do not operate on timestamps but on block numbers, requiring cumbersome back-and-forth translation. Finally, The Graph utilizes a fixed computational model that materializes the results of predefined queries, which confines its query potential to the bounds of anticipated queries and limits exploratory and real-time analysis. For instance, accommodating the queries from §4.1 requires extensive pre-calculation for all block heights (i.e., timestamps), resulting in increased storage demands, performance overhead, and restrictions on data granularity.

5. Conclusions and Future Work

We introduced AlterEgo, a specialized blockchain node with advanced analytics capabilities that maintains the core functionalities of traditional blockchain nodes, offering substantial performance improvements, better user experience, and higher trustworthiness than the state-of-the-art. We plan to open-source AlterEgo upon publication of this article.

The encouraging outcomes from this initial prototype motivate the exploration of further research avenues. One key direction is reducing the load on a single node for long-running queries by distributing the execution across multiple AlterEgo nodes. Another research opportunity involves directly integrating pattern tracking and anomaly detection within the node to enhance its real-time analytics capabilities. A third area for future work is connecting analytics nodes for different blockchains or non-blockchain systems to analyze their interactions. All in all, our ultimate objective is the design of a comprehensive framework for decentralized blockchain analytics.

References

[1] https://etherscan.io, 2023.

[2] https://thegraph.com, 2023.

[3] G. Wood, et al., Ethereum: A secure decentralised generalised transaction ledger, Ethereum project yellow paper 151 (2014) 1–32.

[4] J. Kanani, S. Nailwal, A. Arjun, Matic whitepaper, Polygon, Bengaluru, India, Tech. Rep., Sep (2021).

[5] L. Bousfield, R. Bousfield, C. Buckland, B. Burgess, J. Colvin, E. W. Felten, S. Goldfeder, D. Goldman, B. Huddleston, H. Kalodner, et al., Arbitrum nitro: A second-generation optimistic rollup (2018).

[6] Y. Li, K. Zheng, Y. Yan, Q. Liu, X. Zhou, Etherql: a query layer for blockchain system, in: Database Systems for Advanced Applications: 22nd International Conference, DASFAA 2017, Suzhou, China, March 27-30, 2017, Proceedings, Part II 22, Springer, 2017, pp. 556–567.

[7] Evgeny Medvedev, Blockchain etl, http://blockchainetl.io, ???? Accessed: 2023-11-23.

[8] J. Tigani, S. Naidu, Google bigquery analytics, John Wiley & Sons, 2014.

[9] H. Kalodner, M. Möser, K. Lee, S. Goldfeder, M. Plattner, A. Chator, A. Narayanan, {BlockSci}: Design and applications of a blockchain analysis platform, in: 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 2721–2738.

[10] Ethereum Foundation, Geth (go-ethereum), https://geth.ethereum.org/, ???? Accessed: 2023-11-23.

[11] DuckDB Authors, Duckdb, https://duckdb.org/, ???? Accessed: 2023-11-23.