Simple Blockchain System

Ruthvika Reddy Loka, Mahitha Pillodi, Mia Weaver

1. Abstract

A blockchain is a distributed ledger technology that enables secure and transparent transactions without the need for intermediaries or centralized entities. Blockchain technology has gained significant attention and recognition due to its decentralization and potential to revolutionize various industries, ranging from finance to supply chain management. Blockchain offers secure and immutable transactional systems.

In this project, our aim is to implement a simple blockchain using Python, to understand and explore the fundamental principles underlying blockchain technology. By developing a simple blockchain implementation, we aimed to gain insights into the core components and mechanisms that enable secure and decentralized transaction validation. The primary focus is on implementing the Nakamoto consensus/longest chain wins protocol and integrating a proof-of-work mechanism. The blockchain implementation is deployed across a small cluster of cloud nodes leveraging GRPC for coordinating consensus and propagating transactions among the nodes.

The project's objectives include evaluating the performance of the developed blockchain by simulating various client use cases and incorporating timing code to measure the time taken for a transaction to be safely considered "committed" under different system success scenarios. Furthermore, the recovery time from system failures is assessed. Through these evaluations, a comprehensive understanding of the blockchain's efficiency and reliability is obtained.

2. System Architecture

The system architecture of the implemented blockchain can be divided into several key components, each serving a specific purpose in the overall functionality of the blockchain network. The architecture is designed to facilitate consensus, transaction validation, block creation, and communication among network nodes. The integration of Nakamoto consensus, Proof-of-Work, and GRPC communication facilitates consensus, enhances network security, and enables efficient coordination among network nodes.

2.1. Blockchain Network

The blockchain network consists of multiple nodes connected in a peer-to-peer fashion. Each node maintains a copy of the blockchain, allowing for decentralized validation and consensus. Nodes communicate with each other to propagate transactions, exchange blocks, and reach consensus.

2.2. Consensus Layer

The consensus layer is responsible for achieving agreement among the network nodes on the validity and order of transactions. The implemented blockchain utilizes the Nakamoto consensus algorithm, where the longest chain with the most accumulated proof-of-work is considered the valid chain. Nodes compete to solve a computationally intensive proof-of-work puzzle to create new blocks and append them to the blockchain. Consensus is reached when the majority of nodes agree on the longest chain, ensuring a secure and tamper-resistant network.

2.3. Transaction Management

Transactions are the fundamental units of data in the blockchain. Transactions contain information such as sender and receiver addresses, transaction amounts, and additional data. When a transaction is submitted to the network, it undergoes validation to ensure the integrity and authenticity of the transaction data. The validation process includes verifying digital signatures, checking available balances, and ensuring the transaction adheres to predefined rules.

2.4. Block Creation and Validation

Nodes in the network collect validated transactions into a block. Once a block is created, it undergoes additional validation to ensure it meets specific criteria, such as the proof-of-work requirement. The proof-of-work algorithm, typically based on hashing functions, requires nodes to find a nonce value that, when combined with the block's data, produces a hash with specific properties. The mining process involves iterating through nonce values until a suitable hash is found, which satisfies the proof-of-work criteria. Once a valid block is created, it is broadcasted to other nodes in the network for verification and inclusion in their local copies of the blockchain.

2.5. GRPC Communication

GRPC (Google Remote Procedure Call) is utilized for communication between the nodes in the blockchain network. GRPC provides a high-performance, language-agnostic framework for defining services and message types using Protocol Buffers. Nodes use GRPC to exchange information, propagate transactions and blocks, and coordinate consensus. The use of GRPC ensures efficient and reliable communication among the distributed network nodes.

2.6. Proposed Block Chain Architecture

- Clients issue TXs to any node.
- Nodes batch TXs locally and propagate blocks.
- Only blocks and block status messages sent between nodes.
- Block order established via Nakamoto consensus.
- 4th block back committed; 3 most recent nodes pending/subject to change.

2.7. Nakamoto Consensus

Definition: A consensus mechanism used in blockchain networks to ensure agreement on the state of the blockchain, named after Satoshi Nakamoto, the pseudonymous creator of Bitcoin.
How it works: Nodes in the network compete to solve a mathematical puzzle (proof-of-work) to validate a new block and add it to the chain. The first node to solve the puzzle broadcasts the new block to the network, which then verifies and accepts it.
Advantages: Decentralized, secure, resistant to tampering.

2.8. Proof of Work (PoW) Mechanism

Definition: A computational puzzle that miners (nodes) must solve to validate new blocks in a blockchain network.
How it works: Miners use computing power to perform calculations and find a solution to the puzzle. Typically, this involves incrementing a nonce value until its value hashed with the block hash is within some small, pre-determined range of values. The solution is then included in the new block as proof of work.
Purpose: The proof-of-work algorithm ensures that the process of adding new blocks to the blockchain is resource-intensive and time-consuming, making it difficult for malicious actors to manipulate the blockchain.
Bitcoin's Use: Bitcoin, the first blockchain-based cryptocurrency, utilizes the proof-of-work algorithm (SHA-256) to validate transactions and secure the network.

3. Implementation Details

3.1. Key Functionalities

3.1.1. Block Creation A node can generate a block if it has any local pending transactions. A miner batches some or all of its transactions, and (pair-wise) hashes them together into a "Merkle root." The Merkle root is hashed with the miner node's identifier and with the previous hash. The previous hash allows for it to fit on the chain. The hashing of the miner identifier prevents other nodes from claiming the rewards of the block.

3.1.2. Block Appending A node accepts a received block and appends it to its local blockchain if:
The received block's previous hash is equivalent to the block hash of the last block in the local chain.
The node can recreate the received block's block hash by hashing the transactions stored in the received block with the miner ID and previous hash values.
• - Valid transactions and miner

The last six characters of the resulting string when the received block nonce is hashed with the received

block hash is >= “fffffd”
• - Valid proof-of-work

3.1.3. Transaction Handling A client can issue transactions to any node in the network. However, transactions are NOT shared among nodes. All transactions sent to a node are batched locally and sent into the network within a block. If a node crashes, the pending transactions on that node are lost. If the replica receives an update, transactions from discarded blocks that originated locally will be added back to the pending transaction pool so that they may be batched into a block again later.

3.1.4. Consensus Mechanism Whoever mines and propagates the block the quickest determines the transactions and order of transactions for that “step”/portion of the ledger. If a node receives a block while it is in the mining process, the mining process will continue but the block it is mining will be discarded, as the length of the chain has changed during mining. Nodes alert each other of their chain length and of the block hashes of their current chain, which can be used to reach a consensus when there is a discrepancy among nodes.

3.1.5. Hashing Mechanism We use the simple built-in hash() command provided by Python to hash data in our blockchain.

A block hash is comprised of:

Merkle root: Items in the list of transactions are recursively hashed together until one item called the Merkle root, remains.

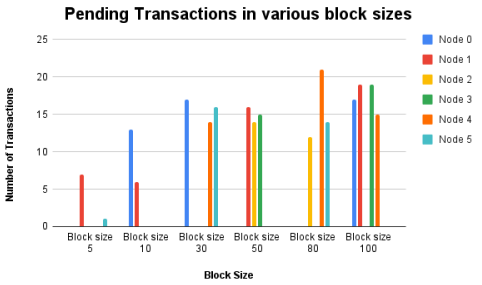
Miner Node ID: The Merkle root is hashed with the ID of the block miner, so that no later node may try to claim credit for the block.

Previous Hash: The previous hash value is hashed with the Merkle root and miner node ID so that it fits onto the chain. Nonce is hashed with the block hash to show proof-of-work.

3.1.6. GRPC Implementation We use Python gRPC to coordinate transaction and block ordering. Client can issues transactions to any node. Nodes communicate only to propagate blocks. On the server side, each RPC invocation is either sending a block or responding to a block that was sent. When a miner sends a block, a replica responds with the status of the block (whether it was appended to the chain or not), the length of its current chain, and a list of block hashes in its current chain. If a block append fails and the replica is shorter than the miner’s chain, then the miner finds the last common block hash and sends a series of blocks until the replica is updated.

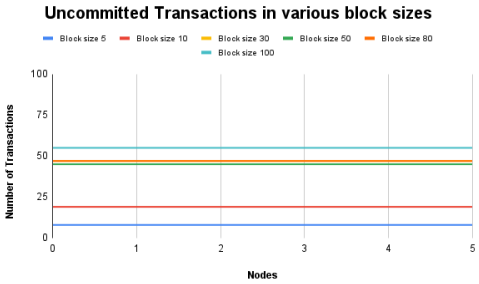
4. Performance Evaluation

To evaluate the performance of the system, we consider several metrics. During our initial performance test, we investigated the number of pending transactions across different block sizes. This analysis was based on a dataset of 100 transactions, and we adjusted the y-axis values to a maximum of 25 in order to enhance the visibility of the bars on the graph. In this context, "pending transactions" refers to those that have not yet been included in a block. The block sizes tested were 5, 10, 30, 50, 80, and 100. Upon examining the results, we noticed that nodes 2 and 3 exhibited no pending transactions for block sizes of 5, 10, and 30. This suggests that these nodes were able to mine blocks at a faster rate compared to the other nodes for those specific block sizes. Additionally, as the block size increased, the number of pending transactions also increased. Notably, Node 1 had a higher number of pending transactions compared to any other node, indicating slower mining performance. Overall, Out of the total 100 transactions analyzed, most nodes had less than 15% of the transactions pending on average.

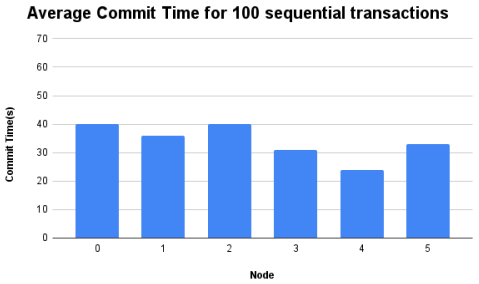


During our second test, we focused on measuring the quantity of uncommitted transactions across different block sizes. Similar to the initial experiment, we analyzed a set of 100 transactions, which were represented on the left axis of the graph. In this context, "uncommitted" refers to transactions that have been added to the blockchain but have not yet been

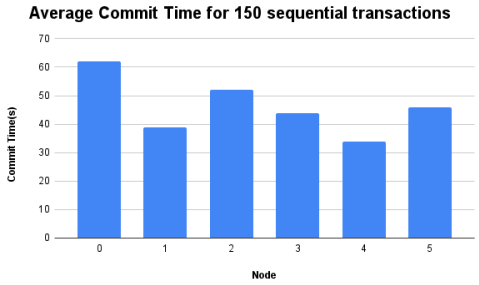
executed in user wallets. We repeated the block size variations of 5, 10, 30, 50, 80, and 100. As anticipated, we discovered that as the block size increased, so did the number of uncommitted transactions. This finding highlights the influence of larger block sizes on the volume of transactions awaiting execution in user wallets. However, it is noteworthy that the count of uncommitted transactions remained below 50, and even below 25 for lower block sizes, up until a block size of 80. Only when the block size reached 100 did the count slightly exceed 50.



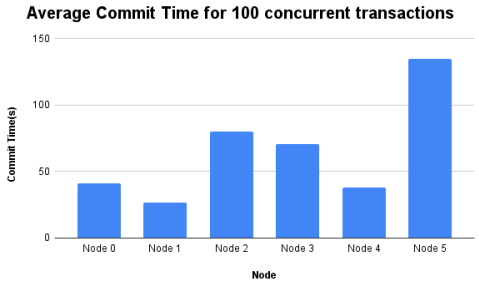
Moving forward, we shifted our focus to measuring the commit times of transactions across different nodes. We conducted a sequential transmission of 100 transactions from the client, deliberately introducing a 2-second delay between each transaction. Throughout this experiment, we found that the maximum commit time recorded was 40 milliseconds. Notably, node 4 demonstrated the fastest commit time, completing transactions in as little as 24 milliseconds. On average, the commit latency was 34 milliseconds.



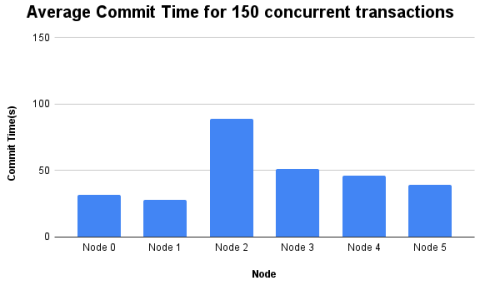
Continuing the experiment, we repeated the process with 150 transactions, maintaining the same delay between each transaction. Interestingly, we observed that the commit times remained relatively consistent, with the highest recorded commit time reaching 60 milliseconds. On average, the commit latency was 46 milliseconds.



We proceeded to measure the commit times of 100 concurrent transactions across different nodes. On average, the commit latency was 65 milliseconds.



Additionally, we measured the commit latency for 150 concurrent transactions. Surprisingly, the average commit latency of 47 milliseconds was similar to the average commit latency observed for 150 sequential transactions.



5. Conclusion

Blockchain consensus is relatively simple to implement compared to something like RAFT. Fewer RPC must be passed through the network. However, PoW and Nakamoto consensus require a significant delay between when a TX is proposed and when it is

committed. This may also be an artifact of our implementation choice to have TXs batched locally before being sent into the network. Nodes don't have to agree on the selection of TXs to batch from. This implementation decision may mean TXs get dropped if the node crashes, or could be "stuck" pending for a while.

6. Discussion

Blockchain architecture with Nakamoto consensus and proof-of-work algorithms provides a secure and decentralized approach to transaction validation. These mechanisms have been successfully implemented in various blockchain networks, enabling trust, transparency, and immutability. As blockchain technology continues to evolve, alternative consensus mechanisms and algorithms are being explored to address scalability and energy efficiency concerns.

7. Future Work

We could improve our blockchain system by having nodes communicate issued TXs rather than batching locally. We chose to focus on TX ordering as our main implementation goal, with less priority given to anonymity and security, the traditional goals of a cryptocurrency scheme. Our design leaves open the opportunity for a malicious node to create and batch illegitimate transactions into the network. This is unverifiable, since there is no shared, agreed upon pool of TXs to select from. Transactions are sent transparently through the network. This was so that all nodes can update the wallet values they're tracking and can be verified as equivalent at the end of the simulation. These design choices would certainly not work in a real blockchain environment and could stand to be fixed in future work.

8. References

- Referenced this blog for merkle tree implementation reference.
- Referenced this link for Proof of Work.