CRYPTONIX

Blockchain Data Structures & Consensus Algorithms

Design and Implementation of a Simplified Blockchain System Using Merkle Trees, Linked Lists and Proof of Work Consensus Algorithm.

Rishi Kaneria • Saher Dev • Sahil Narkhede • Sahilpreet Singh • Sake Akshay

B23CS1058 • B23CS1059 • B23CS1060 • B23CS1061 • B23CS1062

TA : Prajjwal Nijhara

Introduction to Data Structures & Algorithms

Course Code : CSL2020

TABLE OF CONTENTS

About the Project

This project implements a simplified blockchain system in C++ with core elements essential for secure, decentralized transaction management. C++ is efficient in memory and processing management, essential for blockchain systems that require high-speed, low-latency operations. The project focuses on creating a lightweight, efficient system for transaction verification, block propagation, and consensus establishment, exploring the following key components:

- **Data Structures:** Utilizes *Merkle Trees* to organize transactions within blocks, ensuring data integrity and efficient verification. Bloom Filters are integrated to optimize block propagation across nodes, reducing network overhead while maintaining accuracy.

- **Consensus Mechanisms:** Implements *Proof of Work (PoW)* algorithms, allowing the network to reach a consensus on valid blocks. This ensures decentralized decision-making and strengthens the system's security.

- **Relevance**: Blockchain technology offers a *secure, transparent solution* for decentralized data management, finding applications in finance, supply chain, and data privacy. This project provides hands-on experience with blockchain fundamentals, equipping developers to understand and implement blockchain technology in real-world scenarios.

This project is a learning bridge, demonstrating blockchain concepts while harnessing C++'s performance to create a secure, optimized system.

Introduction

Blockchain technology was introduced in 2009 with Bitcoin by an anonymous creator known as Satoshi Nakamoto, offering a decentralized and secure way to record transactions. Originally intended for cryptocurrency, blockchain now spans industries like finance, healthcare, supply chain, and more, due to its ability to create tamper-resistant and transparent records. A blockchain operates as a distributed ledger, where each block contains a list of transactions and is linked cryptographically to the previous one, forming an immutable chain verified by a network of nodes.

Blockchain systems rely on several algorithms and approaches to maintain security and efficiency. **Consensus mechanisms** like Proof of Work (PoW) and Proof of Stake (PoS) ensure that new blocks added to the chain are valid, while cryptographic **hashing algorithms** (e.g., SHA-256) secure the data within each block. Real-world uses of blockchain include digital payments, decentralized applications, and data security across industries. This project explores blockchain implementation in C++, where data structures and algorithms (DSA) like linked lists, Merkle Trees, and hashing algorithms are crucial.

Implementing blockchain in C++ provides performance advantages due to efficient memory usage and control over data structures, making it suitable for handling cryptographic and data-heavy operations essential to blockchain. C++ allows precise management of resources, which enhances the speed and security of blockchain networks.

Key Learnings:

- Gaining a deep understanding of data integrity and security through **cryptography** and **consensus mechanisms** like Bitcoin's Proof of Work.
- **Implementing DSA principles** to optimize data management, verification, and processing within the blockchain like Merkle Tree Implementation.
- Applying **C++ strengths** in handling low-level data management, improving the performance of blockchain applications.

Data Structures & Algorithms Used

In this blockchain project,we have integrated secure data structures and algorithms to ensure efficient transaction management, verification, consensus, and user access through networking.

- **Linked List for Block Storage**: Each block in the blockchain is connected in a linked list format, where each block points to the hash of the previous block, securing the sequence. This structure makes it easy to traverse the chain and ensures immutability, as changing any block would alter all subsequent blocks' hashes, invalidating the chain.

- **Merkle Tree for Transaction Verification:** Transactions within each block are organized using a Merkle Tree structure, which enables efficient, secure transaction verification. By hashing transaction pairs and combining these hashes up to a single root, the Merkle Tree provides quick verification without needing to access every transaction. This structure is especially effective in wallet systems, enabling swift confirmation of transaction authenticity.

- **Proof of Work for Consensus:** Proof of Work (PoW) is implemented as the consensus mechanism to ensure that blocks added to the blockchain are valid and accepted by the network. Miners adjust a block's *nonce* to solve a cryptographic puzzle, searching for a hash that meets the required difficulty level (e.g., with leading zeros). This process is computationally intensive, making it costly to alter past blocks and reinforcing network security.

- **Nonces in Proof of Work:** Nonces are crucial in the mining process, as miners incrementally adjust this variable to generate a unique hash. The nonce is adjusted until a valid hash that satisfies the difficulty target is found, adding computational complexity and preventing malicious alterations to the blockchain.

- **SHA-256 for Hashing:** The SHA-256 algorithm is used for creating fixed-length, secure hashes for each block and transaction. This algorithm ensures unique and unpredictable hashes, making tampering practically impossible and providing high security for the blockchain.

- **Socket Programming for User Access:** Socket programming is utilized to facilitate secure user login and interaction with the blockchain network. Through socket connections, users can remotely access the blockchain system, submit transactions, and view block data in real-time. This setup provides a network interface for users to interact with the blockchain, allowing seamless transaction logging and monitoring. Secure protocols are applied within the socket connections to protect login data and transaction integrity.

The design choices, including linked lists for block storage, Merkle Trees for transaction verification, Proof of Work for consensus, and SHA-256 for hashing, create a secure and efficient blockchain system. Additionally, the use of socket programming enables user access, making the blockchain easily interactable in a networked environment.

Methodology

The B*lockchain Implementation* in C++ follows a structured methodology that incorporates several key concepts and technologies to create a secure and efficient system.

Block Initialization as a Linked List

In the implementation, blocks are initialized and stored in a **vector**. Each block in the blockchain contains a reference to the previous block's hash, making the blockchain a chain of blocks. The ***previousBlockHash*** in each block ensures that the blocks are connected in a linear sequence, and this linkage is crucial to the security of the blockchain. By storing blocks in a vector, we can dynamically manage the growing blockchain while maintaining the necessary references for block linkage.

Transaction Handling and Merkle Tree Verification

Transactions are stored in a **vector** within each block. To ensure that transactions are valid and secure, the ***Merkle Tree*** structure is employed. A *Merkle Tree* allows for efficient and secure verification of transactions by creating a hash of each transaction and storing these transaction hashes in the tree. The Merkle Root, a single hash at the top of the tree, represents the entire set of transactions in a block. This structure ensures that tampering with any transaction would require recalculating all hashes up to the root, making it extremely difficult to alter the blockchain without detection.

SHA-256 Algorithm for Secure Hashing

The core of the blockchain's security is based on the **SHA-256 hashing algorithm**. *SHA-256* is a cryptographic hash function that converts an input (in this case, transaction data, previous block hashes, and nonce) into a fixed-length hash. The algorithm is **collision-resistant**, meaning it is computationally infeasible to find two different inputs that produce the same hash. This ensures that every block's hash is unique and secure, preventing tampering or malicious activities.

Use of Basic Data Structures and C++ Concepts

Throughout the implementation, fundamental **C++ data structures** such as **vectors** and **strings** are used to manage and store blockchain data. Vectors provide a dynamic and efficient way to manage an ever-growing blockchain, and strings are used to handle data like hashes and transaction details.

Peer-to-Peer Network Integration

The blockchain implementation includes a **Peer-to-Peer (P2P) network** to facilitate decentralized communication between nodes in the system. P2P networks allow each node to share information about blocks and transactions with others in the network, ensuring that the blockchain remains synchronized across all participating nodes. This decentralized approach prevents any single point of failure, contributing to the robustness and security of the blockchain system.

Frontend Integration for Visualization

To enhance the user experience, a **frontend interface** is integrated with the blockchain implementation. The frontend allows users to interact with the blockchain by viewing the blocks, transactions, and blockchain status. It provides a graphical representation of the blockchain and enables users to initiate transactions and add blocks to the chain. The frontend communicates with the backend through API calls, ensuring seamless interaction between the user interface and the underlying blockchain logic.

By combining these elements, the blockchain system in C++ achieves high security, scalability, and ease of use, making it suitable for various applications requiring distributed and tamper-resistant data storage.

Resources

- Visual Demo- Blockchain
- https://github.com/tko22/simple-blockchain
- https://github.com/HemaZ/my_blockchain
- https://github.com/Savjee/SavjeeCoin
- Proof Of Work

Conclusion

This project provided a valuable opportunity to explore and implement various algorithms and data structures that form the foundation of blockchain technology. Working with *hashing algorithms like SHA-256* and *linked lists* to organize and secure blocks has deepened our understanding of cryptography and data integrity. These are fundamental concepts not only in blockchain but also in many areas of computer science. Additionally, implementing *Merkle Trees* and *Proof of Work* as consensus mechanisms allowed us to apply data structures in real-world scenarios, enhancing both security and efficiency within the blockchain.

The hands-on experience in C++ has sharpened our skills in low-level data management, making us more proficient in optimizing performance and ensuring the reliability of blockchain systems.By combining these technical concepts, we were able to develop a robust blockchain system that can securely record and verify transactions.

Moving forward, we aim to improve this project to scale for a larger audience by:

- *Enhancing scalability:* Optimizing the blockchain to handle larger volumes of transactions efficiently.
- *Integrating additional consensus mechanisms:* Such as Proof of Stake or hybrid models for better energy efficiency and faster transactions.
- *Implementing a user-friendly interface:* To make blockchain interactions more accessible.

In addition to the backend development, this project also gave us the chance to work on the *front end*, helping to create a user-friendly interface for interacting with the blockchain system. This allowed us to bridge the gap between technical implementation and user experience. Furthermore, getting familiar with GitHub was an essential part of the project, where we learned the importance of version control, collaboration, and got familiar with open-source contributions.

Team

Rishi Kaneria (B23CS1058)
Saher Dev (B23CS1059)
Sahil Narkhede (B23CS1060)
Sahilpreet Singh (B23CS1061)
Sake Akshay (B23CS1062)

GitHub Link

https://github.com/Rk1805/DSA_Project_BlockChain