Highly Dependable Systems:HDS Serenity Ledger

Group 41

Guilherme Pascoal 99079 Miguel Vale 99113 Tiago Caldas 99125

1. Introduction

For this stage of the project, we will focus on addressing the negative aspects identified in the first stage, implementing transaction and balance checking methods, and conducting further testing.

2. Fixes from first delivery

In the latest update from our first delivery, we've made several key improvements to enhance security and efficiency. Firstly, we introduced signatures and nonces to prevent byzantine nodes from tampering with data and to avoid replay attacks, thereby ensuring non-repudiation. Secondly, for communication within the link, we've opted to use MACs instead of digital signatures. Additionally, we've implemented a feature to piggyback the quorum of Prepare messages whenever a round change occurs. Finally, to improve the reliability of client interactions, we've made adjustments so that clients now wait for a quorum of messages before proceeding. These fixes are part of our ongoing commitment to refining the system and ensuring its robustness and reliability.

3. Solution Design

3.1. Accounts

To store the clients accounts we have a map where the key is the hash of the client's public key and the value is the account. This implementation makes it impossible for a client to have multiple accounts, but could be easily extended so it handles it. Instead of the value being the client account it could be a list of the clients account, each account has two balances, the book and the authorized.

3.2. Check Balance

As the check operation doesn’t modify the state of the blockchain - it's a read - the client will wait for a quorum of messages, the value that the quorum will use to do the frequency is the book balance and the authorized balance

combined in a string. We introduced two types of balances, the authorized and the book. Like in a real world bank, transactions are not instantaneous and have a cost, so with this implementation, the client it's better informed with his balance at the moment and future balance when all transactions are processed.

3.3. Transfer Funds

For simplification, when requesting a transfer, the client only needs to introduce the destiny client's id and the amount and a Transfer Message will be sent to the nodes. In the backend logic, everything works with the hashes of both clients' public keys. A transaction it's signed by the sender and has a nonce, guaranteeing non repudiation and preventing replay attacks. Transactions modify the state of the blockchain, and because consensus is very costly, a consensus of transactions it's only started when 3 transactions occur. This number was chosen by us as just a measure of concern about the costs and resources utilized in consensus, as it would not make sense to have a block per transaction, and can be easily modified in the future for a more realistic value. To process transactions, we store them on a list of current transactions and when we start the consensus, we serialize this list to a string and apply the algorithm as done in the first delivery. When commiting, all transactions are processed and a block it's created on the list of transactions. The gas fee it's just a deterministic value. In the future we would like to introduce different types of transactions. Client's that want their transactions to be processed faster would have to pay a higher fee.

3.4. Commit Values

When sending the confirmed message back to the client, the Confirmed Message now includes the ledger position and the nonce of each transaction. This is implemented so that when the client receives a quorum of confirmed messages, they can distinguish them by using the nonce, because the ledger position will be the same for all transactions in that block, while the nonce is unique for each transaction.

3.5. BlockChain

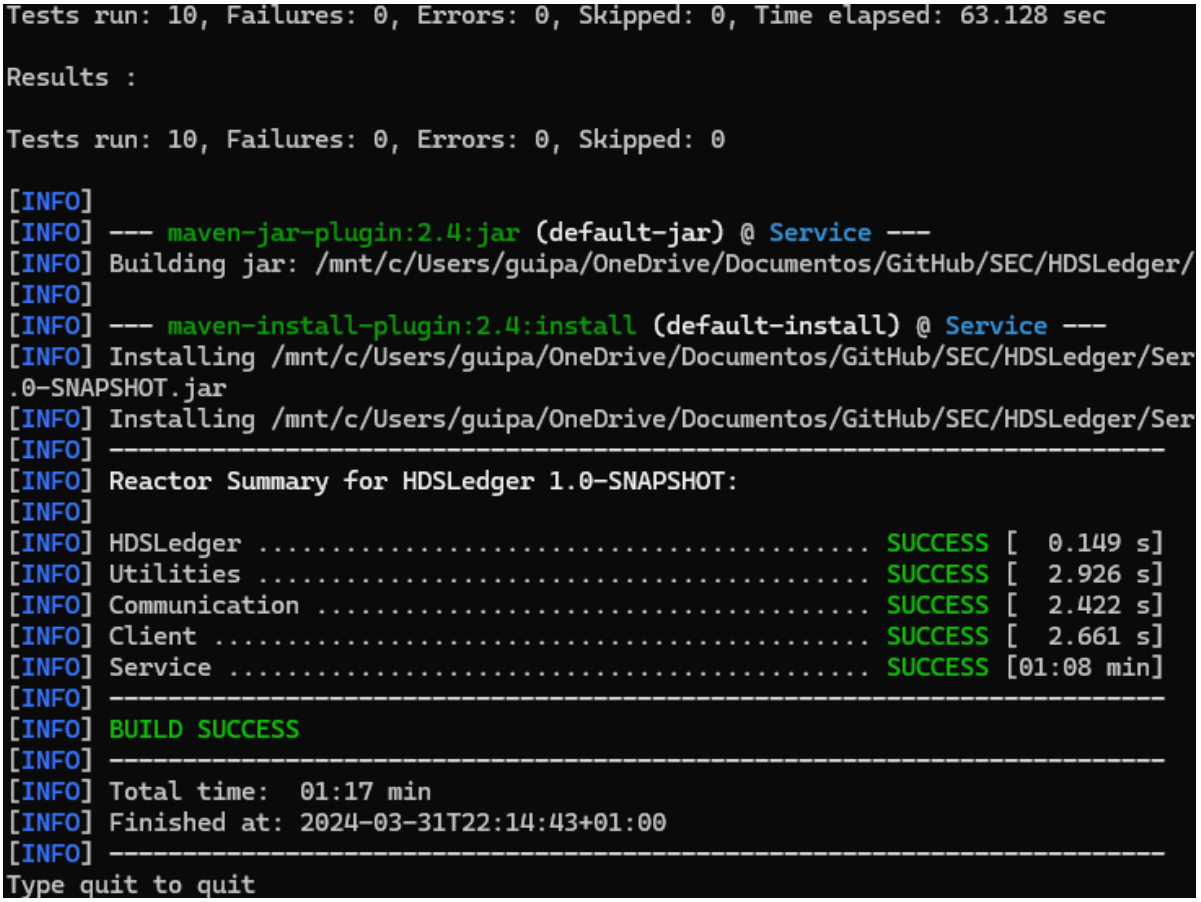
In our blockchain, we maintain a list of current transactions, where each list includes 3 transfers. Once we've added 3 transfers to the list and committed the values, we proceed to create a block. To do so, we require the hash of the previous block. We then create the block using the previous hash and the list of transactions. Afterward, we clear the transaction list, making it ready to accept new transactions for the upcoming block.

4. Tests

4.1. Our Tests

- Test when our leader is byzantine
- Test when we have a node that isn't the leader that is byzantine
- Test when we have two clients that send a append messages and the first message received isn’t the same for every node
- Test when we have a transaction in a normal behavior when we don't have any byzantine activity
- Test when someone is trying to do a replay attack of a transaction
- Test when a byzantine client tries to sign the data with a random signature
- Test when a byzantine client tries to sign the data with his signature
- Test when we have insufficient balance
- Test if forcing a negative balance is allowed
- Test when we try to transfer a negative amount to another account

4.2. Results



5. Conclusion

Overall, the development of the HDS Serenity Ledger represents a significant step towards creating a highly secure and efficient blockchain system. Through meticulous fixes, thoughtful solution design, and rigorous testing, our team has enhanced system integrity and reliability. Looking ahead, we remain committed to further refinement and innovation, aiming to set the standard for decentralized systems' security and usability.