Blockchained Global Wallet Service

Bruno Carvalho (52910), Luís Chula (52340)
NOVA School of Science and Technology
{bmd.carvalho, l.chula}@campus.fct.unl.pt

Abstract—This report is carried out in the scope of the course Reliability of Distributed Systems. In the course project, it was proposed that we create our virtual currency. The solution we created implements a decentralized service for a distributed and replicated ledger of registration and control of operations of clients, based on a consistent replication architecture, with multiple servers and with byzantine fault tolerance guarantee. The ledger is inspired by a Blockchain storage architecture solution in a chain based, where transactions are replicated in a chain of blocks with irreversibility and verifiability, across all servers. In our solution, it is also possible to install and use smart-contract to validate transactions. We also implement a private transaction mechanism using the SJ-Homo library, which allows the user to spend their coins privately.

Index Terms—BFT-SMaRt, Blockchain Technologies, Proof of Work, Blockchain Technologies, Smart Contracts, Homomorphic Cryptography, Bitcoin.

1 Introduction

Blockchain has attracted attention as the basis of cryptocurrencies such as Bitcoin, but its capabilities extend far beyond that, enabling existing technology applications to be vastly improved and new applications never previously practical to be deployed. Also known as distributed ledger technology, blockchain is expected to revolutionize industry and commerce and drive economic change on a global scale because it is immutable, transparent, and redefines trust, enabling secure, fast, trustworthy, and transparent solutions that can be public or private.

Blockchain, like the Internet, is an open, global infrastructure that allows companies and individuals making transactions to cut out the middleman, reducing the cost of transactions and the time-lapse of working through third parties. The technology is based on a distributed ledger structure and consensus process. The structure allows a digital ledger of transactions to be created and shared between distributed computers on a network. The ledger is not owned or controlled by one central authority or company and can be viewed by all users on the network.

Blockchains can be public and unpermissioned, allowing anybody to use them (bitcoin is a case in point) or private and permissioned, creating a closed group of known participants working, perhaps, in a particular industry or supply chain.

The main goal of our project is to implement a dependable decentralized ledger for a blockchain global wallet service supporting distributed client wallets and peer-to-peer crypto transactions. We will be using the BFT-SMaRt library to ensure the total ordination of all transactions in the ledger. BFT-SMaRt implements a modular state machine replication protocol on top of a Byzantine consensus algorithm and for this reason, we can say that our solution is Byzantine fault-tolerant and has a resilience guarantee for intrusion tolerance. Transactions stored in the ledger maintain authenticity and integrity proofs about the provenance and destination of transfers, allowing the use of these proofs for any necessary dispute or auditing verification, by third parties. The integrity and irreversibility of our ledger are guaranteed by the Proof-of-Work (discuss in section 2.3).

To improve our study on this topic, we have also implemented a mechanism for installing and using Smart Contracts similar to the one used by the Hyperledger Fabric. For this, we created more BFT-SMaRt replicas (we can call these replicas endorsers) with the function of verifying if the Smart Contract is good to be used or not. These new replicas have a more restrictive policy not allowing the code written in Smart Contracts to break the system. If the smart contract is well implemented and does not violate any policy, it will be installed and made available to blockchain users.

To increase anonymity and allow the use of private transfers, we use SJ-HomoLibrary to be able to perform operations on encrypted data. This approach allows the data present in the ledger to be processed without the data having to be decrypted. This ensures that third parties cannot calculate the account amount of users with private transactions.

The remainder of this paper describes the architecture of our project and our experience with it. Section 2 summarizes all the background needed to understand the rationale behind our solution. Section 3 introduces the system model and architecture of BGWS in detail, illustrating the transaction execution flow. In Section 4, the key components of BGWS are defined, in particular, the WalletClient, Wallet, BFTServer Replicas, and BFTEndorser. Section 5 will be used to describe the issues we face and how we solve them. Our results and analysis will be presented in Section 6, in this section we will discuss the throughput of our solution and the reasons for that. Section 7 is reserved to present the methods we implemented. To finalize our paper, we will take the conclusions in Section 8.

• *NOVA School of Science and Technology — FCT NOVA*

2 BACKGROUND

In this section are presented the concepts on which BGWS is built.

2.1 BFT-SMaRt

The support for state machine replication under the byzantine faults assumption was assured by the BFT-SMaRt library.

BFT-SMaRt is a robust Java-based BFT SMR library that implements a modular state machine replication protocol on top of a Byzantine consensus algorithm.

Clients send their requests to all replicas, triggering the execution of the consensus protocol. Each consensus instance begins with the leader replica proposing requests to be decided within that consensus, which pattern is shown in figure 1.

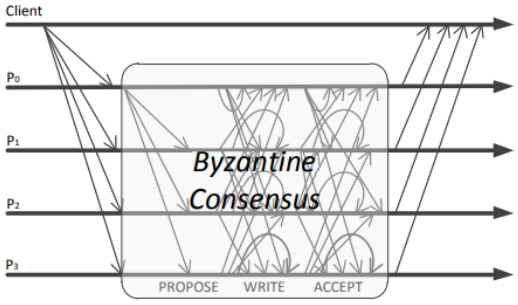

Fig. 1: BFT-SMaRt normal phase message pattern.

Each consensus instance begins with the leader sending a **PROPOSE** message to all the replicas, containing the batch of requests to be decided. All the replicas that receive a **PROPOSE** message verify if its sender is the leader and if the proposed batch is valid. If these conditions are met, they register the request being proposed and send a **WRITE** message to all replicas containing a cryptographic hash of the proposed batch. If a replica receives a majority of **WRITE** messages with the same hash, it sends an **ACCEPT** message to all other replicas containing this hash. If some replica receives a majority of **ACCEPT** messages for the same hash, it delivers its correspondent batch as the decision for its respective consensus instance.

BFT-SMaRt also implements a state transfer protocol that allows the replicas to be repaired and reintegrated into the system in case of failure. This technique log batches operations in a single disk while they are executing takes snapshots during execution in different replicas to avoid stopping the system, and performs state transfer in a collaborative way, with each replica sending different parts of the state to the recovering replica.

2.2 Blockchain Technologies

A blockchain is an open database that maintains a distributed ledger typically deployed within a peer-to-peer network. It is comprised of a continuously growing list of records called ”blocks” that contain transactions. Blocks are protected from tampering by cryptographic hashes and a consensus mechanism.

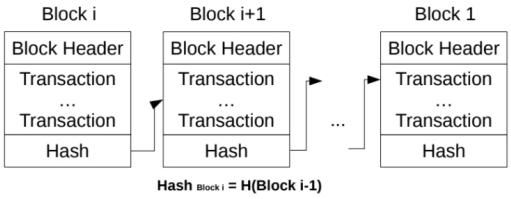

Fig. 2: Blockchain structure

The structure of a blockchain – illustrated in Figure 2 – consists of a sequence of blocks in which each one contains the cryptographic hash of the previous block in the chain. This introduces the property that block j cannot be forged without also forging all subsequent blocks j + 1...i. Furthermore, the consensus mechanism is used to prevent the whole chain from being modified; and to decide which block is to be appended to the ledger.

There are two different types of blockchain, permissionless and permissioned.

Permissionless blockchains have the benefit of enabling the ledger to be managed in a completely open way, i.e., any peer willing to hold a copy of the ledger can try to create new blocks for it. On the other hand, the computational effort associated to PoW consensus is both energy- and time-consuming; even if specialized hardware is used to find a Proof-of-Work, this mechanism still imposes a limit on transaction latency.

Permissioned blockchains, on the other hand, are completely the opposite of permissionless blockchain as they offer private networks. They are not open to the public and require permission from the owner or the admin to join the network. They can use a traditional Byzantine consensus protocol to decide the order by which the blocks are inserted into the ledger or blockchain.

2.3 Proof of Work

Proof of Work or PoW is defined as a process for achieving consensus and building on a digital record known as a blockchain. With PoW, users compete with each other via their computers to solve a puzzle. The key idea behind PoW consensus is to limit the rate of new blocks by solving a cryptographic puzzle, i.e., execute a CPU-intensive computation that takes time to solve but can be verified quickly. The first peer that presents such a solution gets its block appended to the blockchain. Roughly speaking, as long as the adversary controls less than half of the total computing power present in the network, PoW consensus prevents the adversary from creating new blocks faster than honest participants.

2.4 Smart Contracts

A smart contract is a self-executing contract with the terms of the agreement between buyer and seller being directly written into lines of code. The code and the agreements contained therein exist across a distributed, decentralized blockchain network. The code controls the execution, and transactions are trackable and irreversible.

Smart contracts permit trusted transactions and agreements to be carried out among disparate, anonymous parties without the need for a central authority, legal system, or external enforcement mechanism.

2.5 Homomorphic Cryptography

Homomorphic encryption is a method of encryption that allows any data to remain encrypted while it is being processed and manipulated. It enables a third party (such as a cloud provider) to apply functions on encrypted data without needing to reveal the values of the data. A homomorphic cryptosystem is like other forms of public encryption in that it uses a public key to encrypt data and allows only the individual with the matching private key to access its unencrypted data. However, what sets it apart from other forms of encryption is that it uses an algebraic system to allow a variety of operations on encrypted data.

In practice, most homomorphic encryption schemes work best with data represented as integers and while using addition and multiplication as the operational functions. This means that the encrypted data can be manipulated and analyzed as though it's in plaintext format without actually being decrypted.

3 BGWS System Model and Architecture

The Blockchained Global Wallet Service consists of a Byzantine fault-tolerant decentralized consistent ledger with persistent local-ledger replicas of operation in the system stored in irreversible blocks, with total order guarantees.

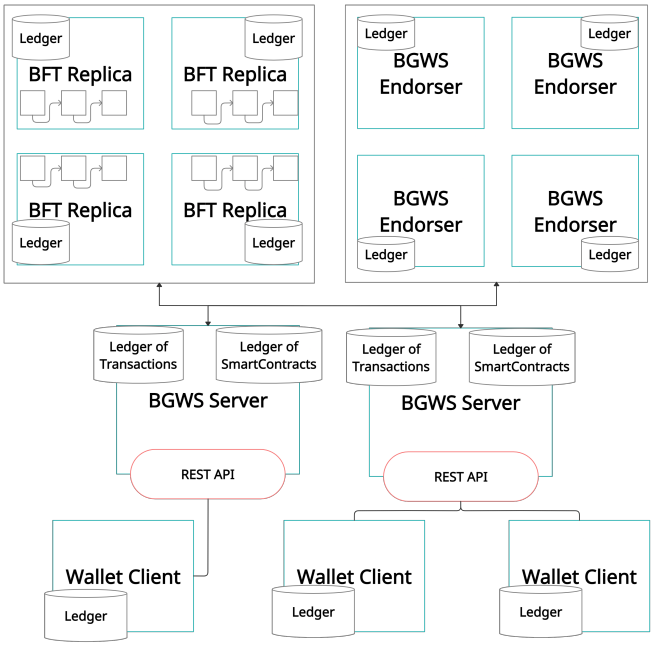

Fig. 3: Architecture of the BGWS System

The Wallet Client issues operations to the REST endpoints provided by the BGWS Server.

The BGWS Server makes asynchronous requests to the byzantine fault-tolerant Replicas that execute a consensus protocol and replicate the issued operations in the ledger.

On operations that require transaction validation using Smart Contracts, such as transferring money, the BGWS Server executes an asynchronous request to the BGWS Endorsers that execute the code in the referenced smart contract to ensure that the transactions being made are indeed valid and secure.

The BGWS Endorsers, just like the Byzantine fault-tolerant Server Replicas, execute consensus and replicate a ledger of the available Smart Contracts used to validate the transactions. Endorsers have strict runtime policies to make sure that the transaction being made does not compromise the system.

The safety in communication between the different components is assured by using the Transport Layer Security protocol and the HyperText Transfer Protocol.

4 Mechanisms and Service Planes in BGWS Software Architecture

This section details the decisions made to implement the abstractions for the transactions and blocks and the different components of the architecture.

4.1 Transaction

A transaction identifies the Wallet Client that issued the operation that generated the transaction. The operation executing specifies the issuer, the receiver if there is one, and the value if there is one.

The transaction is signed by the issuer and the public key of the issuer is attached to the transaction so that the signature can be verified.

To support private transactions, a transaction type was added, being type 0 a regular transaction, type 1 a transaction with the amount ciphered with a symmetric key, and type 2 a transaction with the amount ciphered with a homomorphic key. In case the transaction has type 1, an envelope is available, containing the symmetric key encrypted with the receiver's public key so that the receiver can obtain the encrypted value transferred to him.

The transactions of type 2 reference a transaction of type 1, justifying the homomorphic transaction made. For instance, Alice transfers to Bob an encrypted amount, a transaction T1 is added to the ledger. Then, as Bob is the receiver of a type 1 transaction, to use the money sent to him, he needs to make a type 2 transaction T2 that references T1. This way, Bob can verify and use all the private money he has.

4.2 Block

A block contains a list of finalized transactions, the hash of the previous block, except if it is the genesis block, as there are no other blocks in the chain, and a nonce that makes the hash of the block respect the number of required zeros to the left, proving the work done by the peer that mined the block.

Each block also stores the id of the Wallet Client that achieved the proof of work, along with its public key and a signature of the list of transactions.

4.3 System Reply

The BGWS Server uses an Asynchronous Service Proxy to communicate with the BFT Replicas and the BGWS Endorsers. To gather the responses from these services, there was the need to implement a Reply Listener.

The Reply Listener collects the replies from the services until it has a valid quorum of replies, containing the id and the signature of the replica that generated the reply, and even the hash of the reply for posterior validation.

The replies are stored in a System Reply that is afterward sent to the Wallet Client, allowing the client to verify the result of the issued operation, along with which replica sent the reply and the integrity of the reply.

4.4 BGWS Client

The client plane is the service responsible for communicating with the server and the application user. On this level, the blockchain user has the possibility to use a set of operations provided by the Server API.

Each client has its own key pair, which are used to sign requests made to the server and consequently to the BFT replicas. The signature ensures authenticity and allows the server and replicas to verify if the request is trustworthy or not.

All SystemReplies are stored in a local ledger allowing the client to verify their integrity and authenticity if anything goes wrong.

The client also has a Keystore where it stores the public key of all clients present on the network. This information allows the client to be able to execute private transactions. To support private transactions the client has a PaillierKey, which permits the client to perform operations in the ciphertext.

4.5 BGWS Server

BGWS Server provides a set of operations through its REST API. Each request made by the Wallet Client is verified before being replicated. The signature sent by the clients is validated, ensuring the authenticity of the client and guaranteeing that the requested transaction is legitimate.

The BGWS Server plane is used as a proxy, allowing communication between the client and the BFT replicas. For each request made to the BFT-SMaRt, BGWS Server has to wait for $\frac{n+f+1}{2}$ valid replies. This approach ensures that the consensus was reached and BFT-SMaRt can tolerate faulty nodes.

As we mentioned before, the presented solution supports smart contract installation and we had to ensure that all installed smart contracts followed a strict policy, not allowing the possibility of smart contracts to break the system. For each smart contract that the user wants to install on the system, BGWS Server sends it to the endorsers' quorum to be validated. If the server receives $\frac{n+f+1}{2}$ valid responses, the smart contract will be stored with the remaining valid smart contracts.

4.6 BGWS Endorsers

The BGWS Endorser extends a DefaultSingleRecoverable, a class that provides a basic state transfer protocol using the interface SingleExecutable, offered by the BFT-SMaRt library.

The Endorser provides an interface to handle messages sent by the BGWS Server, requesting for Smart Contracts installation. This interface verifies the signature of the Wallet Client that issued the request and verifies the Smart Contract. If the Smart Contract is valid and a majority of Endorsers reach a consensus on the contract installation, then the contract can be added to the ledger.

In the reply, the Endorser sends the Reference to the Smart Contract, the hash of the operation, and a signature, so that the Client can verify the integrity and authenticity of the reply.

4.7 BFT Replicas

As mentioned above similar to Endorsers, BFT Replicas extends the DefaultSingleRecoverable class of BFT-SMaRt. This gives us the ability to support ordered and unordered invokes. Ordered invokes were used to ensure total ordering of messages, and therefore were used only for write operations. Unordered invokes were used for reading operations, where the order is not that important.

The BFT replica has two different ledgers, one to save the mined blocks and the other to save unfinished transactions. Let's consider the finalized transactions as the transactions present in the mined blocks of the blockchain.

Before executing any requested transaction, the replica verifies the client's signature. If the signature is valid, the requested transaction is performed, otherwise, it is ignored. The client can also ask to be used a Smart Contract to validate the transaction, in this case, BFT Replica searches for the smart contract in their database and uses it to validate the transaction.

In the case, that the user tries to add a new block to the blockchain more verification has to be done. BFT replicas verify that the PoW is correct and that there are no situations of double-spending.

BFT Replica only replies to the client if the consensus is reached. The reply is composed of the operation result, the hash, and the signature of the operation so that the Client can verify the integrity and authenticity of the reply.

5 Coverage of the Requirements

All the ten mandatory operations were implemented with the expected persistency, consistency, and security guarantees, and producing the desired results.

The gold-operation `SearchPrivateTransactions` was also implemented, returning all the private transactions where the given Wallet Client was involved.

6 Implementation Issues

An issue that we observed in our project is the fact the sender doesn't lose money in private transactions. It is impossible for other users of our blockchain to know if a user has the money needed to make a transaction if he/she had made a private transaction in the past for other users. For instance, Alice has 150 public coins and Alice wants to transfer to Bob 100 coins using a private transaction. Bob knows that the amount of Alice is 150 so he accepts the

transaction and finalizes the transaction with his homomorphic key. If Alice tries to transfer 50 coins with a private transaction to Carlos, Carlos does know that Alice has 150 public coins, but she made a private transaction with Bob in the past. So Carlos doesn't have the guarantee that Alice has 50 coins.

7 Experimental Observations and Analysis

The tests performed on the system generated the following experimental results.

7.1 Latency

On average, when a Wallet Client issues an operation, he has to wait 0.225 seconds to get the result from the BGWS Server.

However, the first operation made by a Wallet Client takes, on average, 1.167 seconds.

The first time a Wallet Client and a BGWS communicate they execute the TLS Handshake Protocol, making performance slightly worse.

7.2 Time to Mine a Block

To determine the time needed to mine a block, different values were used for the Proof-of-Work:

8 bits with value 0: The average time to find a nonce that generates a hash with the first 8 bits at 0 is 1.012 seconds.

16 bits with value 0: On average, it takes 11.27 seconds to find a nonce that verifies this Proof-of-Work.

24 bits with value 0: It took 456.67 seconds to find a nonce that would verify the Proof-of-Work on the Genesis Block. The second block took 29.953 seconds to find the nonce.

However, the third block, with four miners working in parallel, spent more than 30 minutes trying to prove the work and none of them were able to find a nonce that would verify the Proof-of-Work.

The result obtained for the second block is clearly an outlier. The Wallet Client that mined that block got really lucky finding the nonce in such a short time.

The results obtained for the 24 bits demonstrate that the Proof-of-Work is infeasible in useful time. On the other hand, with only 8 bits the work is too easy, posing no challenge to the miners.

So we decided to request a 16 bit Proof-of-Work, as this is a value that is usable in a demonstration.

7.3 Throughput

Performing a considerable amount of transactions leads to the conclusion that the system is capable of performing about 4.38 non finalized transactions per second, meaning that every second, 4.38 transactions are added to the pool of transactions waiting to be placed in a block.

So, assuming the 16 bit Proof-of-Work and that the blocks had 50 transactions during testing, we get a total of 4.4366 finalized transactions per second.

8 Conclusion

In this paper, we describe the design, implementation, and evaluation of the Blockchained Global Wallet Service. We start by presenting the background needed to understand the choices of our solution, then we presented the architecture and planes of the system. At the end of the paper, we discuss some of the most relevant topics to be analyzed and some of the results obtained.

Our experimental evaluation shows that the latency is on average 0.225 seconds after 5 or 6 operations. We have to take into account that the first operation to be executed will take longer than the following ones due to the TLS Handshake Protocol. Performing a considerable amount of transactions leads to the conclusion that the system is capable of performing about 4.38 non finalized transactions per second.

We can also conclude that the time it takes to mine a block of transactions increases exponentially with the increase of proof of work, this is due to the decrease of hash collisions.

9 Future Work

Regarding the communication protocol used to ensure the security of the transmitted, the system uses TLSv1.2. We would like to upgrade this protocol to version 1.3 to achieve a faster handshake and to use stronger cipher suites.

Currently, the persistency of the ledgers is achieved by using data stored in local files. In the future, we plan to achieve the persistency of the ledgers by using a real database. We had in mind using either MongoDB because the data that we store in the local files is already in the JSON format, so the changes needed to achieve this would be quite simple, or Redis, due to the minimal complexity that it provides and the good performance.

Finally, we would like to develop a mobile application that runs the Wallet Client with a good UI to facilitate the interactions with the BGWS Server.

10 Acknowledgement

We would like to thank Professor Henrique Domingos for providing a challenging project that motivated us throughout the semester. We would also like to thank him for all the support he provided us throughout the semester.

References

[1] Materials provided for the Reliability of Distributed Systems course.
[2] Alysson Bessani, João Sousa, Eduardo E. P. Alchieri, "State Machine Replication for the Masses with BFT-SMART".
[3] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System".
[4] Kyle Croman et. al., "On Scaling Decentralized Blockchains".
[5] João Sousa, Alysson Bessani, Marko Vuklic, "A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform".
[6] Elli Androulaki, et. al., "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains".