TÉCNICO LISBOA

Highly Dependable Systems

2022/2023

Grupo 12:

Guilherme Santos, nº 96740

José Santos, nº 96750

Tomás Oficial, nº101259

1. Used Technologies

In our project, we used Java 17 as our programming language and as the source for our cryptographic primitives. We used maven for build automation.

As we were supposed to simulate the abstraction of fair-loss links, we used UDP sockets for the communication between the components of our project.

2. System Components and Abstractions

Our system is composed of two main parts, the servers who run and maintain the blockchain and accounts state, and the users who send requests to the servers in order to create their account, transfer money or check their balance. The set of servers is previously known by all the participants in the system and we assume that the leader never changes.

We implemented perfect links in the communication between servers, and in the communication between users and servers, in order to assure that requests from users and messages exchanged in the consensus algorithm, are always delivered and delivered just once.

As soon as the user initiates, it will broadcast a request for creating an account. The client waits for a quorum of replies. Each account is initiated with 100 tokens and has associated a write timestamp.

We developed another abstraction which is a block, a block is a set of requests of transfers from users. We defined that a block is a set of 3 requests of transfers to ease the testing process, but the block can have any number of requests. These blocks are later added to the blockchain which is a list of blocks. Each transaction that is in the block, consists in the message sent by the user that performed that transaction. It contains the message concatenated with its digital signature.

The server that is the leader at some given moment, is responsible for receiving the transactions, creating these blocks and then when the block is full of transactions, sending it to the other servers when initiating the consensus algorithm.

When the leader receives a transaction it verifies its validity, and only if it is valid, it is appended to the block.

We implemented a simplified version of the Istanbul BFT Consensus algorithm to be used by the set of servers to choose the next block of operations to append to the blockchain. Since in this stage we assume that the set of servers is

previously known by all the participants in the system and we assume that the leader never changes, we didn't implement the rounds mechanism described in the algorithm, but it was our concern to develop a simplified version of the algorithm that was easy to change and to add the rounds mechanism to it in a later phase of this project. Our algorithm supports concurrent consensus instances; whenever the leader receives a transfer from a user and the block gets full, it initiates a consensus instance with some consensus id and starts sending “PREPREPARE” messages to the other servers. Something important in this process, it that the other replicas don’t trust blindly in the leader, when they receive a “PREPREPARE” with some given block, they also verify the validity of the transactions inserted in that block. If they detect some invalid transaction they don’t proceed and don’t send the “PREPARE” messages. This would mean that the leader was byzantine and it would originate a round change, but as we said earlier we did not implement the rounds mechanism described in the algorithm. To prevent that some servers could decide to append some blocks in a different order than other servers, we added a mechanism to the algorithm that forces a given server to not append a block to the blockchain unless all the blocks of consensus instances with lower ids were already decided and appended to the blockchain, or aborted.

When a block is decided, the transactions in it are processed, and the client that made that request is informed. The client waits for a quorum of replies. We developed a reward system, each transaction pays a fee to the creator of the block the transaction is inserted in. For reasons of simplicity, we decided that the user who requested the transaction must pay a token to the owner of the block.

In the communication between servers and in the communications between servers and users, we send all the messages with a specific format we defined, concatenated with a digital signature, in order to assure the integrity, authentication, and non-repudiation of each message. The message goes in plain text.

As most ledgers are public, we did not care about confidentiality.

3. Check Balance

For the users to be able to check their account balance, they have two ways to do so, by using a strong read or a weak read mechanism, which we describe below.

3.1. Strong Reads

In strong reads, the user broadcasts a check balance request to all servers and waits for a quorum of responses.

After receiving this quorum the user will start the following algorithm:

1. Go through all the responses and check if there are 2f+1 responses with the same timestamp.

2. After determining the correct timestamp, the user checks again in the messages that have the correct timestamp their values and chooses the most frequent value. This prevents the user from taking a byzantine value but with the correct timestamp.

With this double checking system the user will return the correct value.

3.2. Weaks Reads

For weak reads the blockchain has a mechanism that implements a special block containing the actual accounts balance for a time instance that is signed by a quorum of servers. It adds this special block starting from the fourth consensus instance and repeats this every 3 consensus instances.

When the user performs a weak read request, the server that gets this request, responds with the special block, then when the user gets this it will check for the signatures inside this block and only if there is a quorum of valid signatures it will return the value inside the block for his account.

4. Guarantees

- Dropping Messages Prevention:
  - The messages exchanged between servers are always delivered due to the perfect links.
- Replay Attacks Resistant:
  - The message ID is always appended to every message. The Signature uses this ID so if the message is changed the new Signature will not be verified.
- Authenticity:
  - Signatures (RSA-4096) for server and user-server communication are generated and appended in every message over its payload.
- Client Concurrency:
  - The Server is implemented using synchronized methods that access the same data.
- Byzantine Clients:
  - if a client is byzantina and tries to perform a transfer from another account to his own account, the servers will not permit that because the source account key must match the key that provided the message signature.
- Byzantine Servers:
  - A server that is byzantine can't perform actions in the name of others because of the signature system implied in the message exchange system.

    - If a byzantine server tries to initiate a consensus instance, the other servers will detect it because of the id system.
- Man in the middle attacks:
    - All the messages are protected against attackers changing the content of the messages, because of the digital signatures.