BFTB Banking

SEC Project Report

Group 20

89627, Gustavo Pinto

102118, Rui Costa

102146, Sebastião Sotto Mayor

April 23, 2022

Changes to the Design

Firstly, we have corrected some lack of correctness in our code from the previous delivery. More specifically:

- Improved logic verification to, for example, stop users when inputting negative values for transfers.
- The bank is now parallelized, as in, once the bank receives a request from a client, instead of being sequential which would halt it from executing any concurrent requests at the same time.
- The client library (the API) now checks for freshness from the bank using the original requestID sent by it, granting a simpler and functional freshness check.
- When executing a transaction, the bank not only stores the information of the transaction, but also the signature of the executor. This will guarantee non-repudiation off the clients.
- The requestID sent by the clients is now persistent and stored in a file kept by the bank size. When starting the application, the API will request this value and return it to the client.

Another change occurred on the client side of the application, where we have implemented a login function, where the user will provide its key information, which will not be asked again when executing the transaction up until the logout from the user.

Afterwards, our goal was for it to be possible to include multiple replicas of the bank, for this, we have first altered our file structure (of CSV files), now keeping a folder for each replica, with each folder containing:

- One to store information on the clients;
- One for each client to store information on pending transfers;
- One for each client to store information on completed transfers;
- One to store the current requestID of each client;
- One to store the signatures of the transactions;
- One to store the current transactionId;

Implemented system protections

Integrity and freshness checks are done in every message exchange, which ensure the authenticity of the sender, thus eliminating the possibility of man-in-the-middle and replay attacks. To achieve this, integrity guarantees have been established, which are explained in the respective subsection.

Protections in case of Byzantine faults

In order to keep guaranteeing the security and dependability of the system in the case of Byzantine faults, we have implemented, as was required, some additional protection. It is described below.

Byzantine Atomic register

Firstly, it was our goal to prevent malfunctions in the case of a malicious user in the bank side, we have implemented an Atomic register algorithm with Byzantine protections [1]. For this, clients now store the list of replicas of the bank that are in the system, as well as their address

and when sending the request the API now loops through all of this values sending the request to all the replicas.

When writing, clients also send a signature for the request which will be used in reads, and always wait for at least $(N + f)/2$ ACK's from the replicas before returning, since this value provides Byzantine majority.

In the case of a read, when a client receives a value it checks all the signatures from the transactions included in it, and ignore any value which includes an invalid signature. It then chooses the correct value as the one with the most (valid) transactions. The information about the client like current balance is not signed since it is simply a combination of the transactions together with the (fixed) initial balance.

Byzantine clients

As it was explained in the specifications of the project, the above algorithm does not prevent alterations to the system from Byzantine clients, and so, to add this layer of security we have implemented a series of protections, more specifically:

- Puzzle checks in order to prevent scripting attacks.
- Validation of user input to prevent SQL injections.
- Prevention of DoS attacks by denying requests that:
  - Come from simultaneous clients with the same IP address
  - Do not satisfy minimum ingress data checks
  - Do not satisfy maximum load limit checks

Confidentiality Guarantees

All information is assumed to be public, so no confidentiality issues arise when sending the messages. Therefore, there is no need for confidentiality guarantees.

Dependability Guarantees

Integrity

To guarantee integrity of the system we have used:

- Message digest (hash) of the information sent between entities to prevent manipulation attacks on the message sent.
- Signing of the message digest along with a timestamp by the sender to guarantee authenticity.
- Incremental token in requests, to prevent duplicate, drop and reject attacks on messages sent through the UDP channels.
- Since sent requests are always signed by the sender, non-repudiation is also guaranteed, with the bank storing these signatures in case of an attempt of deniability.

Availability and Reliability

- In our system we first assume that a maximum of f faults can happen, with this assumption, our system guarantees availability by replicating the system at least $3f + 1$ times, and also, making sure a system is always available to receive requests by executing the requests in background and not sequentially.

- To prevent the system from terminating at every error (leading to a system failure), exceptions are caught and printed, and the client is returned to the bank options menu to resume the service.

Safety

The system in hand is a safety-critical one [2], because of significant financial loss in the event of a serious failure. This issue is mitigated by handling exceptions and returning the client to a safe state whenever an unexpected event occurs.

Maintainability

All significant system actions are logged to help maintainers detect errors before they cause a system failure. Also, the way the system is structured in a modularized way, permits the easy restoration of service in the event of a system failure.

References

[1] Cachin, C., Guerraoui, R., Rodrigues, L. (2011b). Introduction to Reliable and Secure Distributed Programming (3rd ed.). Springer Publishing. https://doi.org/10.1007/978-3-642-15260-3

[2] Spencer, J. (2019, September 16). What is Safety-Critical System? definition & meaning. Technipages. https://www.technipages.com/definition/safety-critical-system