HDS Serenity Ledger - Stage 2 - Project Report

Luís Filipe Pedro Marques, *nº 99265* Ramiro Marques Moldes, *nº 99313*
Team nº 25, Alameda

1. System Overview

Our IBFT implementation normally consists of 4 server nodes and 3 client nodes. Client nodes are very simple, they have the list of server nodes and every request is broadcasted to all, they block until f+1 responses are received. Server nodes contain 3 services, the Node service, which is where the IBFT protocol is, the Client service, that receives the client requests and interfaces with the Node service, and finally, the Mempool, that stores all client requests and creates the Blocks that are decided in the Node service.

2. Changes from Stage 1

Stage 2 of the project added new requirements, namely, converting from a simple string concatenation ledger to a block-based account transfer-requests ledger.

2.1. Client

The client changes were minimal. Append was removed, replaced with Transfer and Balance.

2.2. Server - Client service

The Client service was altered to accommodate the new Transfer and Balance requests. A Transfer request simply adds the message to the Mempool and creates a timer. If the Mempool is over the block limit (which is set to the number of clients) or the timer expires then a consensus is started.

Balance requests are simpler, we have a Map acting as a cache that contains all accounts and their balances, the values are updated on every Decide so the requests are immediately answered with the value in cache.

2.3. Server - Node service

The Node service had heavy alterations, firstly to complete the Stage 1 requirements. Piggybacking was added, RoundChange messages now have a Map of messages, which can both be Prepare messages or Commit messages, the former for justifying and the latter for recovery.

Then, the Stage 2 requirements were added, first with the introduction of Blocks, which is just a List of messages, then with the introduction of Accounts and the “cache” Map that is updated on every Decide.

All transfer requests in a Block have to pay the leader a fee, it was set by default at a single unit. Both the fee and the leader are implicit, the block only stores the instance it was decided in.

2.4. Server - Mempool

The Mempool service was added after Blocks for proper creation and maintenance. A request added to a mempool is only removed after it is found on a decided block, it utilizes a Queue so that requests that are missed by previous leaders are prioritized, and it contains a Map of timer for each request to alert of any missed requests or late blocks.

A late Block is one that stays unfilled for a long time. In our implementation since there are only 3 clients and they’re all blocking, it means the Blocks can, at max, be 3 transactions long. If one of the clients doesn’t make a transaction then the block will not be filled nor decided and the other clients will wait indefinitely.

2.5. Security

With bigger complexity, more attack vectors appear, so it’s imperative to keep the entire system secure. Both the Mempool and the piggybacking have entire messages, and the reason for it is the signature to properly verify the Prepares or ClientTransfers.

3. Tests

These are the following behaviors that we implemented to run byzantine tests:

//A process thats sleeps for too much time

SLEEP("SLEEP"),

//A client process does not send messages to the leader

NO_SEND_TO_LEADER("NO_SEND_TO_LEADER"),

//A client process sends a messages pretending to be another client

TRANSFER_CLIENT_PRETENDING("TRANSFER_CLIENT_PRETENDING"),

//A client process sends a message

CHECK_BALANCE_CLIENT_PRETENDING("CHECK_BALANCE_CLIENT_PRETENDING"),

//A client process sends a message with a wrong nonce

```
CHECK_BALANCE_WRONG_NONCE("CHECK_BAL
ANCE_WRONG_NONCE"),
    //A client tries to send the same message twice

DOUBLE_SEND_MESSAGE("DOUBLE_SEND_MESS
AGE"),
  //A process sends a round change message with a different
value

FAKE_ROUND_CHANGE("FAKE_ROUND_CHANGE")
,
     //A server process ignores messages from a client
     IGNORE_CLIENT("IGNORE_CLIENT");
```