Highly Dependable Systems

HDS Serenity Ledger

Stage 1 Report

Group nr.:	04
99239	Jaime Gosai
100731	Edson da Veiga
110916	Miguel Rocha

Computer Science and Engineering

IST-ALAMEDA

March 08, 2024

1 Architecture

1.1 Client Application

The user interface through which interaction with the system occurs.

1.1.1 Initialization

Upon initialization, the application loads the client and Node configuration files.

1.1.2 Main Loop

The application remains in a continuous loop, reading user input. Initially, only the append operation is implemented, with user input serving as the value for the append operation. The application terminates upon receiving the ”exit” command.

1.1.3 Client Service

Facilitates communication between the Client application and Blockchain nodes. The Client service listens for messages from the Blockchain nodes and offers the append-value operation, which broadcasts to all nodes in the blockchain to append a string value. Asynchronous handling of responses ensures the Client application remains unblocked.

Future Plans: In subsequent stages, the Client library/service will be implemented as a separate component, enhancing system flexibility for future changes.

1.2 Blockchain Node

Handles the logic associated with the blockchain.

1.2.1 Initialization

Similar to the Client, the Node application begins by loading necessary configuration files, including nodes and clients information. Then, it creates two distinct links, one to be used as the comunication between clients and nodes, mostly used in the Blockchain service, and another to be used between nodes, to execute consensus, used in the Node service layer.

1.2.2 Blockchain Service

Offers a simple API for Clients to request services from the Blockchain system. Handles append-requests by starting new consensus instances. The requests are cached, so later it is possible to warn the client about the status of his request.

1.2.3 Node Service

Implements the IBFT protocol for consensus among nodes. Communication primarily occurs between nodes, that execute consensus, with occasional contact with clients for status updates, whenever a value is decided.

1.3 Dependability

1.3.1 Communication

Utilizes a Link layer for communication between processes, using UDP to simulate a Fair Loss link. Already implemented features include validity using ACKs and detection of duplicated messages with message IDs: Perfect Links. However, we implemented Digital signatures since they ensure message integrity, vital for the IBFT consensus protocol.

1.3.2 IBFT and Leader Selection

The IBFT protocol paper already ensures safety across rounds and also liveness, by the view change mechanism.

The leader is selected based on the node configuration file, rotating among nodes in each round to ensure fairness. The next leader follows the order specified in the config file, which is the same for all processes. Also, the leader persists across consensus instances, since it doesn't make sense to change the leader if he correct.

1.4 Byzantine tests

To prove that this system is resilient to several byzantine attacks, the system was tested with multiple configurations, each one corresponding to a different set of processes and respective behaviors:

- **correctConfig.json:** all nodes are correct, and the system don't face any atacks;
- **ignoreRequestsConfig.json:**a single node will ignore the requests coming towards him. In particular, this one will start as leader (id = 1). The tests will show that, after all nodes start consensus, the other nodes will get their timers expired, and they will change their view, so the node with id = 2 will become the new leader, and, since it is correct, it will lead the consensus to a decision/conclusion;
- **badLeaderPropose.json:** the leader (byzantine) will propose a random value for each one of the nodes. The output will show that no node will ever receive a quorum of identical PREPARE-MESSAGE's, leading to a view change, where the original client value will finally be decided;
- **uponPrepareQuorumWrongValue.json:** shows that if a byzantine process, after receiving a quorum of PREPARE requests, and decides to broadcast a COMMIT message, with a incoherent value, the consensus will still decide the right value;
- **uponRoundChangeQuorumWrongValue.json:** similar to the above test, the byzantine and leader node will set a wrong value after receiving a quorum of ROUND-CHANGE messages.

There is a byzantine behavior that we still don't know how to protect against. If the leader is byzantine, and the client wants to append the value X, the byzantine leader could receive the request but never propose that value, and instead propose another one, and the other nodes would never suspect about it, and, so, no view change would be triggered.

Note: Resilience is guaranteed if the maximum number of byzantine nodes is within certain limits defined by the system's parameters.