HDS Serenity Ledger

Daniel Pereira *99194*

Ricardo Toscanelli *99315*

Simão Gato *99328*

Abstract

Your abstract text goes here. Just a few facts. Whet our appetites. Not more than 200 words, if possible, and preferably closer to 150. ONLY IF WE HAVE SPACE

1 System Design Overview

Our highly dependable system follows a layered architecture designed for scalability and fault tolerance. The key components are:

1. **Client Application Interface:** This user-facing layer acts as the entry point for user interactions. It captures user requests and transmits them securely to the Client Service.

2. **Client Service:** This background service acts as a mediator between the Client Application and the server-side logic. It receives requests from the Client Application, performs signature validation (using cryptographic techniques), and broadcasts the message to the relevant server-side service.

3. **SerenityLedgerService:** This core server-side service receives messages broadcasted by the Client Service. It acts as the central coordinator, orchestrating the overall message processing flow. It prepares the data based on the received message and interacts with the Node Service to retrieve the consensus value.

4. **Node Service:** This specialized service encapsulates the logic for reaching consensus on a specific value. It interacts with SerenityLedgerService to receive the prepared data and leverages a defined consensus mechanism (Instambul Byzantine Fault Tolerance) to reach an agreement with other nodes. If consensus is achieved, the resulting ledger state is returned to the SerenityLedgerService.

5. **Communication Flow:** The communication primarily follows a client-server model. User interactions initiate at the Client Application, which transmits requests to the Client Service. The Client Service broadcasts the message to the designated SerenityLedgerService on the server side. SerenityLedgerService then interacts with the Node Service to reach consensus on a value. Finally, the agreed-upon ledger state is potentially relayed back to the Client Service for further processing or user notification.

This layered architecture promotes modularity and separation of concerns. Each layer has a well-defined responsibility, improving maintainability and promoting easier integration of future functionalities. The use of a dedicated Node Service for consensus allows for flexibility in exploring different consensus algorithms depending on the specific needs of the system.

2 Relevant Implementation Aspects

This section highlights some key implementation aspects of our highly dependable system:

2.1 IBFT

We leverage the Istanbul BFT (IBFT) protocol for consensus, accordingly to the official IBFT paper [1], ensuring fault tolerance even in the presence of Byzantine failures (nodes exhibiting arbitrary behavior). The implementation has been thoroughly reviewed and corrected since the initial stages of the project.

2.2 Triggered Consensus

The IBFT protocol execution is optimized to initiate only when a specific number (X) of requests are queued. This reduces unnecessary consensus rounds and improves system efficiency. During the commit phase upon reaching a "DECIDE" state, the participating nodes process the requests accumulated in the queu, by removing them from the queue and executing them.

2.3 Parallel Consensus

Our system enables concurrent execution of multiple IBFT consensus instances. However, to maintain data consistency, ongoing instance Y waits for the completion of instance X (where X precedes Y) if transactions within Y depend on the outcome of transactions in X. This parallel execution can introduce potential complexities if such dependencies exist between concurrently processed requests.

2.4 Ledger Design

The system maintains a ledger consisting of a blockchain structure where each block stores a predefined number (X) of transactions. Additionally, the ledger holds the current state of all accounts within the system.

2.5 Client-Side Verification

When a client queries their account balance, they must receive confirmation from 2f+1 nodes to ensure the retrieved value's accuracy. This requirement arises due to the presence of account states within the ledger. While using a system like UTXO could potentially offer advantages, time constraints during development prevented its implementation.

3 Behavior under attack

The experimental evaluation that shows that the system is dependable in the presence of Byzantine behavior.

4 Tests

Sucint description of the tests that were performed to validate the system.

References

[1] Moniz H. The istanbul bft consensus algorithm, 2020. https://arxiv.org/pdf/2002.03613.pdf.