LINFO2345 : Project Erlang

Academic Years 2023 - 2024

Teacher : Peter Van Roy
Course : LINFO2345
Collaborators : Crochet Cédric (67161800) Romain Mottet(34391900)

Contents

1 Introduction

In the realm of decentralised technologies, the distributed ledger system of blockchain secures records through cryptographic signatures and hash functions. Noteworthy blockchains, such as Bitcoin, Ethereum, and Cardano, employ Proof of Work (PoW), although its energy-intensive nature has led to the emergence of Proof of Stake (PoS).

PoS, as exemplified by Ethereum's adoption in The Merge update on 15/09/22, selects nodes based on cryptocurrency collateral, thus reducing energy consumption by 99%. Our mission is to develop a Proof of Stake (PoS) blockchain using Erlang technology, aligned with the industry's pursuit of creating sustainable and efficient decentralized systems.

2 Part I: Blockchain Distributed ledger

During the project's initial phase, we chose to separate the various functions into specific Erlang modules to form a blockchain. The **'my_node' module** incorporates functions pertaining to distinct nodes, enabling the **formation of new nodes**, the **transmission of messages** to other nodes, and the subsequent **receipt and processing of these messages**.

During the project's initial phase, we chose to separate the various functions into specific Erlang modules to form a blockchain. The **'my_node' module** incorporates functions pertaining to distinct nodes, enabling the **formation of new nodes**, the **transmission of messages** to other nodes, and the subsequent **receipt and processing of these messages**. In terms of creating new nodes, the module accepts the argument for the number of desired Validator and Non-validator nodes. When the nodes are created, their Process IDs (PIDs) are added to a list, which is then passed to the builder module. Additionally, the module processes messages to facilitate message broadcasting.

Following this, we introduced the **'builder' module**, which concentrates on **creating and disseminating blocks**. The initialisation process incorporates a 'start' function that generates and registers the builder. In this function, **a block record** is assigned as a value in the builder loop to preserve vital parameters like block number and general block information for further hashing.

The **Broadcast function** is designed around a basic messaging mechanism where a message is sent to all validators within the ListValidators. While reliable broadcast methods such as BEB (Best-Effort Broadcast), RB (Reliable Broadcast), or dealing with failure scenarios exist, this implementation operates in a synchronous manner. In this setup, it mimics the behavior of a Synchronous System, as it necessitates waiting for the completion of the last action before proceeding to the subsequent step.

Due to the synchronous nature of this system, any errors encountered during execution are expected to halt the program and trigger an error. Unlike asynchronous systems where processes can continue independently, synchronous systems like this one rely on each step's completion before advancing. Consequently, any issues encountered during processing would interrupt the flow and potentially cause the program to stop and report the encountered error.

The **hash function** from the 'crypto' module, which is integrated into Erlang, was utilised during this process. The decision to use this hash function was purposeful, as it gave us the liberty to select the specific hash function that was most suitable for our needs.

Upon initialization, the builder loop accesses a CSV file including transactions, storing them in a list. It then proceeds to check if the transaction list is void of entries. Following this check, it selects the ensuing 10 transactions to construct a block. After block creation, broadcasting transpires to all nodes and is consigned to a new CSV file. The builder loop then repeats the process, eliminating the processed transactions from the list until every transaction in the CSV file has been processed. This iterative process continues until all transactions have been processed.

3 Part II: Consensus algorithm to elect validator group

For the second phase of the project, we implemented the **'election_protocol' module**. During its initialization, we generate **logs** for different validator nodes and set up a **HashMap** that contains validators, proposer group heads, and builders. Subsequently, we commence the election process using another function.

The function begins by transmitting a message to the builder node to temporarily halt block creation. In accordance with this improvement, we modified the **'builder' module** to allow message reception, enabling the interruption of block creation. Furthermore, a 'wait' function was integrated into the 'builder' module, designed to halt execution until a message is received from the election protocol.

As a result, the election protocol function then elects a new proposer group in accordance with the specified election protocol. It is worth noting that the process of selecting the new group only took **2073 microseconds** for 30 nodes. Once the new proposer group has been identified, the HashMap is updated and the information is broadcasted to the validators. Subsequently, a message is broadcasted to the builder node to signal the beginning of a new epoch, during which 10 blocks are created.

The logs of a validator contain valuable information about the ongoing processes within the blockchain network. Below is an interpretation of the provided logs:

- The logs may contain the state of the different nodes, providing the PID of the builder, the proposer group head, and ultimately, all validators.

Listing 1: Logs of the Election Protocol

```
#{builder => <0.106.0>,
    proposer_group_head => [<0.100.0>,<0.99.0>],
    validators =>
      [<0.100.0>,<0.99.0>,<0.98.0>,<0.97.0>,<0.96.0>,<0.95.0>,<0.94.0>,
       <0.93.0>,<0.92.0>,<0.91.0>,<0.90.0>,<0.89.0>,<0.88.0>,<0.87.0>,
       <0.86.0>,<0.85.0>,<0.84.0>,<0.83.0>,<0.82.0>,<0.81.0>]}.
```

- The different operations as Begin Election Broadcast:

Listing 2: Logs of the Election Protocol

```
Broadcasted begin_election from proposer group head <0.100.0>.
```

- As Shuffled List Transmission:

Listing 3: Logs of the Election Protocol

```
Send shuffled list from <0.100.0>.
```

- As Proposer Group Broadcast:

Listing 4: Logs of the Election Protocol

```
Sent new proposer group to: <0.100.0>.
```

- And other operations.

4 Part III: Proof of Stake

During the project's third phase, time constraints prevented full functionality implementation. Nonetheless, we successfully integrated a key aspect, specifically validating transactions before broadcasting a block. This required modifications to the 'builder' and 'my_node' modules.

The **'builder' module** now enhances its ability to **broadcast transactions** from its block to all validation nodes before **waiting to verify their validity**. If the validity of the block is confirmed by a majority of validator nodes, it is accepted.

Additionally, the **'my_node' module** has been enhanced to enable **transaction validity checks**. This enhancement empowers each node to evaluate the validity of transactions, thereby contributing to a more robust and secure implementation of the blockchain.

Although some functionalities are still pending due to time constraints, this progress signifies a considerable step towards realizing a comprehensive and functional blockchain system. Future development can build on this foundation to achieve the complete set of project requirements.

The logs for Part 3 of the project remain unaltered and adhere to the same principles as those in Part 2.