Winter20 CS271 Project1

January 9, 2020

Abstract

Blockchain is a list of blocks which are linked using cryptography. Each block consists of multiple transactions and each transaction has money transfer from one (sender) to another (receiver). That is, when a (sender) sends money to a (receiver), that interaction is stored as a transaction. In blockchain, each block may contain one or more transactions along with the hash value of the previous block.

In this first project, you will develop a "non-replicated" blockchain whose access is controlled by mutual exclusion. In other words, a user needs to get mutual exclusion over the blockchain in order to add a transaction to the blockchain. You should develop the application logic that uses Lamport's Distributed Solution to achieve mutual exclusion.

1 Application Component

We will assume multiple clients. Each starts with a balance of $10.
A client can issue two types of transactions:

- A transfer transaction.
- A balance transaction.

When a client initiates a transfer transaction, it needs to communicate with the other clients in order to achieve access to the blockchain. If the client tries to send more money than it has, the transaction is aborted.

2 Implementation Detail Suggestions

1. The blockchain can be implemented as a centralized linked list, with each node a transfer transaction. A transfer transaction $\langle S, R, amt \rangle$ consists of a sender, S, a receiver, R, and an amount of money amt. This is a simplified blockchain which does not require any hash pointers.

2. Clients don't need to keep track of their current balance. They just need to know their initial balance and can check with the blockchain to figure out the rest.

3. Each client should maintain a Lamport logical clock. As discussed in the lecture, we should use the Totally-Ordered Lamport Clock, ie, $\langle Lamportclock, Processid \rangle$ to break ties and each client should maintain its request queue.

4. Each time a client wants to issue a transaction, they execute Lamport's distributed mutual exclusion protocol. Once they have mutex, they send their transaction to the blockchain. There are two cases:

   (a) transfer transaction: The blockchain first checks if the client has enough $$ to issue this transaction (this will require checking back in the blockchain to calculate the current balance for the client). If valid, then add a new node with this transaction to the blockchain. If not, send back to client saying INCORRECT and hence the transaction is aborted.

   (b) balance transaction: The blockchain checks the balance and returns its value. **Note**: No new node is added to the blockchain.

3 User Interface

1. When starting a client, it should connect to all the other clients. You can provide a client's IP, or other identification info that can uniquely identify each client. Or this could be done via a configuration file or other methods that are appropriate.

2. Through the user interface, we can issue transfer or balance transactions to an individual client. Once a client receives the transaction request from the user, the client executes it and displays on the screen "SUCCESS" or "INCORRECT" (for transfer transactions) or the balance (for balance transactions).

3. You should log all necessary information on the console for the sake of debugging and demonstration, e.g. Message sent to client XX. Message received from client YY. When the local clock value changes, output the current clock value. When a client issues a transaction, output its current balance before and after.

4. You should add some delay (e.g. 5 seconds) when sending a message. This simulates the time for message passing and makes it easier for demoing concurrent events.

5. Use message passing primitives TCP/UDP. You can decide which alternative and explore the trade-offs. We will be interested in hearing your experience.

4 Demo Case

For the demo, you should have 3 clients. Initially, they should read from the same content file and display the following initial information:
Balance:$10
Then the clients issue transactions to each other: A gives B $4, etc. You will need to maintain each client's balance (via checking the blockchain) and display the order of transactions.

5 Deadlines, Extension and Deployment

This project will be due 01/23/2020. We will have a short demo for each project For this project's demo, you can deploy your code on several machines. However it is also acceptable if you just use several processes in the same machine to simulate the distributed environment.