Winter20 CS271 Project2

January 28, 2020

In the last project, you developed a centralized blockchain whose access was controlled by mutual exclusion. Now we'll consider a replicated blockchain, where each client has a local copy of the blockchain and updates it according to messages received from the other clients.

1 Application Component

As before, each client starts with a balance of $10. A client cannot transfer more money than it thinks it has; however, a client might have more money than is recorded in its local log, due to the way that information propagates in the system. When a client sends money, it will record this transaction by updating its copy of the blockchain. A client who receives money will not know about it until it receives a message with a log in which this transaction is recorded.

A client must be capable of doing the following:

- Transfer money to another client.
- Send a message to another client.

In the first case (money transfer), the client need only update its local copy of the blockchain. In the second case (when sending a message), the client will transmit its local blockchain. The recipient client should update its local blockchain according to what it receives from the sender.

2 Implementation Detail Suggestions

1. Each client should maintain a local copy of the blockchain and 2D timetable.
2. Clients don't need to explicitly maintain a local log. Instead, the local copy of the blockchain will behave like the Wuu-Bernstein log, as described in the solution to the replicated log and dictionary problem discussed in class. Clients will then include a partial blockchain instead of a partial log in their messages.

3. It's OK if clients don't have identical copies of the blockchain, due to the order in which messages are received.

4. We won't do garbage collection, but each client should still keep track of the other sites via its 2DTT, so that clients don't have to send their entire log (blockchain) with every message.

5. Since the client has a local copy of the blockchain, it doesn't need to interact with any other clients in order to check its balance. However, the client should still be able to print its balance to the screen for demo purposes.

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, each client can transfer money or send a message to another client. Once a client receives an instruction from the user, it executes it and displays "SUCCESS" or "INCORRECT" (for transfers) or "MESSAGE SENT TO {Client ID}" (for general messages).

3. The user interface should allow each client to print its balance to the screen. A client should also be able to print its estimate of another client's balance. Clients will always know their own balances exactly, but a client's knowledge of another client's balance will be approximate and reflect knowledge since the last message exchange.

4. Only the client itself can decide to transfer money from its own account, e.g. Bob can't issue a transaction from Alice to Bob.

5. 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.

6. 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.

7. 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 and display each client's balance and local logs (blockchains).

5 Deadlines, Extension and Deployment

This project will be due 02/06/2020. 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.