CS 271 Winter2019
Programming Assignment III

Assigned: February 13th, 2019
Due On Demo Day: March 15th 2019–NO LATE DEMOS

PLEASE NOTE:

- Solutions have to be your own.
- It is an **PAIR** programming assignment.

1 Introduction

In the second programming assignment, you successfully implemented the money transfer function in a bank system. In this final project, you will implement a mechanism on the bank side(server side) to handle all the transactions made by clients using **Blockchain and Raft**.

To ensure that untrusted parties can come reach agreement on the state of a database, the bank uses **blockchain** as its database to store all transaction information.

The bank has *three* servers to keep track of all transactions made by clients. It uses **Raft** as its underlying consensus protocol to keep an updated transaction history on all of its servers to ensure proper blockchain replication, and at the same time, to ensure fault-tolerance from server crash failures.

2 Implementation Details

Since *Raft* is a *leader-based* approach to consensus, we decompose this project into **three parts**.

1. Leader Election
2. Normal Operations
3. Node Failure

In the first part, you will implement the leader election of the Raft protocol. In the second part, you will implement the normal operations of the Raft protocol along with blockchain as a database to store transaction information. And in the last part, you will deal with the leader failure of Raft protocol while at the same time,supporting normal operations.

2.1 Part I: Leader Election

- There will be **three** server states starting as **three followers** at the beginning.
- Follow the Raft leader election protocol so that within these three servers, there will be **exactly one leader**, and **two followers** after the leader election stage.

2.2 Part II: Normal Operations

In order for normal operations to execute, you must correctly implement a modified version of blockchain first, and then follow the Raft protocol.

2.2.1 Servers and Clients

- There are three servers.
- There are three clients A, B and C.
- Clients would have an estimated knowledge of the identity of the current leader when performing normal operations. They initiate new transactions and send them to their estimate of the current leader.

2.2.2 Modified Blockchain

- The modified blockchain is represented as a log (i.e, an array and not a linked list) and there will be **no direct pointers** to point to previous blocks.
- There are two major components of a block B: a **header** and a **list of two transactions**. (B-1) means the previous block.
  - The header consists of *four* components
    - current term
    - $H_{header}(B-1)$
    - $H_{listOfTxs}(B)$
    - The nonce
  - A List of transactions
    - txA
    - txB

  Details Explained:

- Current term represents the term being used in Raft.
- $H_{header}(B-1)$ is the SHA256 hash of the **header** of the **previous** block, which is the SHA256 of the string concatenation of term(B - 1), $H_{header}(B-2)$, $H_{listOfTxs}(B-1)$, and the nonce.
- $H_{listOfTxs}(B)$ uses the Merkel Tree data structure, which is a SHA256 hash of the concatenation of two SHA256 hashes of two transactions. Please see **Example1**, where X and Y are two transactions, represented in String, as will be explained next.
- The *nonce* is a random string such that: Taking the SHA256 of the **concatenation** of $H_{ListOfTxs}(H(X)||H(Y))$ of the current block **and** the nonce would give a resulting hash value with the last character being a digit (0-2). In order to do that you will create the nonce randomly. The length of the nonce is up to you. If the calculated hash value does not end with a digit between 0 and 2 (i.e. 0, 1 or 2)as its last character, you will have to try another nonce again. In other words, you need to create the nonce randomly until the rightmost character of the resulting hash value is a digit between (0-2). This is a simplification of the concept of Proof of Work (PoW) used in Bitcoin.)
- txA and txB are two transactions where each transaction is a string in the format: "A B amount" (which means A sends B an amount of 5). And there are **exactly two** transactions in each list of transactions(in each block).

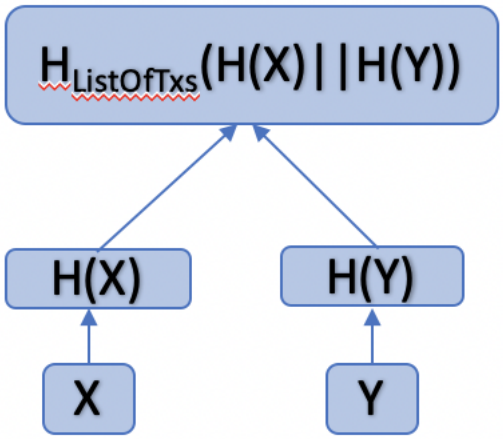

Figure 1: Example1

NOTE:

- Please see **Example2** for a visualization of a block B.
- In the first block, $H_{header}(B-1)$ would be "NULL".
- A block will be added to the blockchain **only if** there are two transactions(i.e. If there are odd number of transactions, then the last transaction wouldn't be added to the blockchain until another transaction comes along).

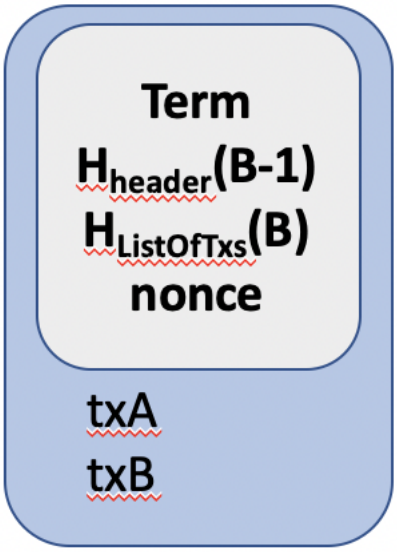

Figure 2: Example2

2.2.3 Raft Normal Operations

Follow the Raft normal operations protocol to ensure that all servers will maintain identical blockchain databases.

2.3 Part III: Node Failure

- Follower failure
  - This is the scenario when a non-leader node fails. After the follower fails, the program should still be able to perform normal operations.
- Leader failure
  - This is the scenario when the leader fails. After leader failure, Raft will begin a new term with leader election and then it should be able to perform normal operations.

3 Demo Flow

- Your program should first read a given input file to initialize the blockchain. This file will consist of a set of triplets (a,b,amount). Assume all clients start with 100 units.
- Your program should be able to **prompt user** for standard input of transactions.
- Your program should have a *PrintCurrentBalance* function which prints the balance of a given client.

- Your program should have a *PrintBlockchain* function which will print out the current blockchain.

  DemoFlow: The demo flow would most likely to be in the following way: You should have already read the input file, and you will be asked to print the current blockchain, and you will be interacting with the program(add more transactions), and you will be asked to print the current blockchain and current client balance at anytime.

NOTE:

1. We do not want any front end UI for this project. Your project will be run on the terminal and the input/output for the demo will use stdio.
2. 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

We will have a short demo for this project. It will be on **March 15th, 2019** in CSIL. Time details will be announced later. Please be ready with the working program at the time of your demo.