**CS 171 : Introduction to Distributed Systems**
**Programming Assignment 2**
**Due: Wednesday May 2, 2018**

For convenient access, polling stations are usually set across different locations. In this project, you are to design a voting system using a replicated dictionary. The voting system must make sure that the total number of ballots to each candidate is correct no matter where the voters cast their ballots. Garbage collection is also required due to the huge number of voter records. Once a record has been replicated over all stations, the corresponding log entries can be deleted.

## 1 Backend servers

Your voting system should consist of **three** servers to support two candidates, Alice and Bob. Each server should maintain a dictionary for the total ballot number for each candidate, a Log for voting records, and a two-dimensional time table (2DTT) for local knowledge about other servers. The dictionary should be of the form

$$\{A : n, B : m\}$$

, representing that A has n ballots and B has m ballots. In the beginning, both n and m are initialized to be zero. Voters can go to any server for casting their ballot to any candidate. On receiving ballots, the server will insert record

$$\{Vote : A\}$$

into the local Log, following the format

$$\{\{Vote : B\}, \{Vote : A\}, ..., \{Vote : B\}\}$$

Note that for expediancy and fast response, the vote is recorded **locally** with no immediate cross-server messages. However, for overall synchronization, servers will send a message to one of the other servers every three seconds, including both the Log and their 2DTT. On receipt of the message, the server should update its dictionary, Log, and 2DTT appropriately. Once finished updating, the server should do garbage collection immediately.

## 2 User Interface

Servers will take input from voters directly. There are four types of commands that voters can type: vote, printDict, printLog, and printTable.

1. Vote is a command for casting ballot to a candidate. The form of vote should be

   $$Vote, X$$

   , where X could be either A for Alice or B for Bob.

2. The purpose of printDict is to show how many ballots each candidate has received from the perspective of local server. Note that we do not require this number to be consistent with other servers all the time, especially when new ballots are coming into the system. However, the consistency is required after a reasonably long time period, since dictionary will be replicated across all servers eventually after all voters have finished voting. To use printDict, the user needs to simply type

   $$printDict.$$

3. For examining the system, we also need a command for printing local log and 2DTT. The command should be

   $$printLog$$

   and

   $$printTable$$

   respectively. There is no requirement on the format of printed 2DTT, as long as it is clear.

## 3 System Configuration

NOTE: We do not want any front end UI for this project. All the processes (servers and kiosks) will be run on the terminal and the input/output for these processes will use stdio.

1. The connection between servers is single direction instead of double direction. Fig. 1 gives an example. A could only send messages to B and receive message from C, while A can neither receive messages from B nor send message to C. In other words, servers cannot broadcast information to all other servers simultaneously and it takes time to pass a single message to everyone in the network.

2. Every three seconds, each server will send a message to the next server about its knowledge, following the algorithm for replicated dictionary problem. The purpose is to pass information to all servers periodically.

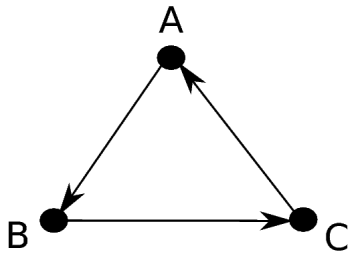


Figure 1: Example

3. When starting a server, it should connect to the other server. You can provide the servers IP, or other identification info that can uniquely identify each server. Or this could be done via a configuration file or other methods that are appropriate.

4. You should print all necessary information on the console for the sake of debugging and demonstration, e.g. Message received from server XX. Message sent to server YY.

5. You should add some delay (e.g. 2 seconds) when sending a message. This simulates the time for message passing and makes it easier for demoing concurrent events.

6. 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

For the demo, you should have 3 servers, one each for representing a polling station. We will have a short demo for each project. For PA2, the demo will be on **May 2, 2018** between 2-4 pm in CSIL. For this projects 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 servers.

## 5 Teams

Projects should be done in team of 2. You can use Piazza to form teams.