Paxos Project

CS 378H Due: 11:59pm TBD

1 Introduction

In the project you will create a chat room service. The service will consist of clients and servers each independent of one another. Servers will be used to maintain order in the chat room. They will take messages from the clients and run Paxos on each of the messages to determine the order of that message. Then the servers will tell the client what the order of the messages are so that each client displays the same chat log. Clients will send messages to the servers attempting to write to the chat log and will receive messages from the servers about committed messages in the chat log.

2 Technical Details

The implementation can be written in any language or languages you like. The two requirements are that no two processes can share memory and all processes be able to run concurrently. Other than that, the rest is up to you. We will not provide any starter Paxos implementation code. For the purposes of this assignment, all servers should be able to be leaders, act as acceptors, and have a local replica of the chat history.

2.1 Assumptions

You can assume that clients will know the server membership (i.e. where the servers are). The servers should be listening on specific ports for client requests. If the primary server dies, then the client should retransmit their requests to all replicas or wait until a new leader is elected.

2.2 Leader Failure Detection

One way for the servers to detect a leader failure is to have the leader send out heartbeats to other servers and the servers could time on those heartbeats.

No matter what, the chat room has to be consistent among all servers. Therefore decisions should only be made when there is at least a quorum of servers alive and at least a quorum agrees on a leader. Full leader election need not be implemented if you implement the "view change" model instead.

2.3 The Master Program

To help you test your implementation and assist with the evaluation, each submission must include a master program which will provide a programmatic interface with the chat room. The master program will keep track of and will also send command messages to all servers and clients. More specifically, the master program will read a sequence of newline delineated commands from standard input ending with EOF which will interact with the chat room and, when instructed to, will display output from the chat room to standard out. We will be using this master program to test your implementation of Paxos, not to trick you up on API edge cases. When you turn in the master program, please also turn in a one line file that contains the command to run your master program. The API for the master program is defined is section 4.

3 What to Turn In

1. Source code for your implementation of Paxos
2. A makefile that compiles your implementation on the CS Machines (if necessary)
3. A master program and a file named "COMMAND" that contains on one line the command to run your master program
4. A README file that details your implementation of Paxos, your name(s), uteid(s), and utcs id(s), major design decisions, instructions on how to use the chat room, as well as any other information you think is relevant to the grading of your service

Skeleton code for the master program, example COMMAND files, and sample test cases will be provided.

4 Master Program API Specification

Command	Summary
start [numberOfServers] [numberOfClients]	This command starts a chat room with a specified number of servers and clients. The servers and clients will be referred to by their index number (zero indexed).
sendMessage [index] message here	This command tells a client (specified by the index) to send a message to the chat room. The message will be every argument after the specified index. The call should be asynchronous. As you can guess there will be times when the grading script will attempt to send a message when there is less than a majority of servers and thus, no message should be committed until there is a majority of servers back in the system.
crashServer [index]	The will shutdown a server immediately.
restartServer [index]	This will restart a server that we have previously crashed.
allClear	This command should not return until the chat room system is idle. This combined with the timeBombLeader command will allow us to test how your implementation handles with failures in the middle of the consensus protocol.
timeBombLeader [numberOfMsgs]	This command tells the current leader to crash itself after sending the specified number of server side Paxos related messages. This excludes heartbeat messages if you use them.
printChatLog [index]	This command (which should block until completion) will print a client's view of the chat record in the format specified below. Our grading scripts will heavily rely on you following the exact format specified.

For each entry in the client's view of the chat log, print out a new line with the message in this format. Do not print the braces. The senderIndex is the index of the client who original sent the message and the sequence number is the Paxos sequence number (zero indexed) for the message. For NOP filled slots, do not print a line at all.

[sequenceNumber] [senderIndex]: message

5 Common Questions and Tips

1. It is useful to send all print results back through the master. This ensures that print requests occur atomically and do not get interleaved.
2. There is no stable storage requirement. Specifically there will always be one replica alive in the system that has all the chat history. Additionally *allClear* will be called every time a server is revived in order for that server to retrieve the state from an up to date replica.
3. We will always call *allClear* before *timeBombLeader* to ensure that leader election has finished.
4. We encourage people to round robin leaders to help simplify leader election.
5. Please do not print any more than the output specified, even if it is to *stderr*.
6. Please, please try running your code with the test harness we provide on a CS machine to make sure that it works and all threads are cleaned up appropriately.