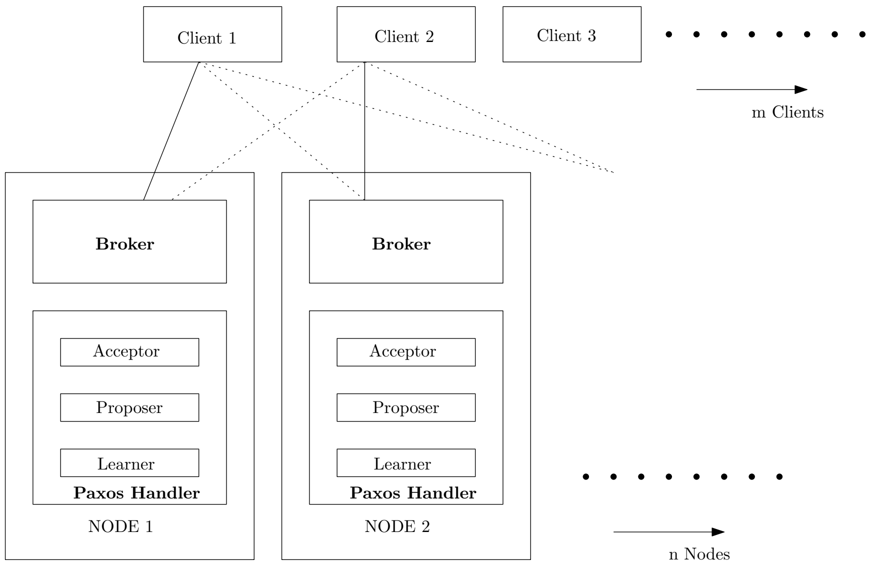
CSE 550 Introduction to Systems Research
Problem Set 2

Marco Tulio Correia Ribeiro, Shrainik Jain
1323300, 1323338

February 13, 2014

1 Implementation

The base system is a set of Nodes, each running a PaxosHandler instance and a Broker which acts as a middleware to interpret client requests. All the communication is done using the Apache Thrift Framework. Any client can contact Broker at any node, with a command of the form *Lock mutexId clientId* or *Unlock mutexNo clientId*. The Broker relays the commands to the PaxosHandler in the same Node.



1.1 Normal operation

PaxosHandler implements *leader Paxos*. Upon receiving a run command request, a node checks if it is the leader, if true, it starts an instance of Paxos with the value as the

command recieved. If the node recieving the client request is not the leader, it asks the leader to run the command.
Each node has an id. At the start, the node with id 0 is set as the leader. In addition, each node also maintains the following state variables:

- current_proposal_number: This starts with 0 for all nodes. Its like a seed to calculate the actual proposal number for Paxos instances. Proposal number is calculated as **current_proposal_number * 1000 + my_id**. Under normal operations, current_proposal_number is never changed.
- An array of all the values accepted so far, indexed by instance number.
- An array of the proposal number promised so far, indexed by instance number.

1.2 Scenario where a node fails

Node or communication failures are detected in the system when any node can't open a transport connection to some other node. In such a scenario **an exception is thrown, but its fine becuase we catch that exception and act accordingly**. Two possible situations could happen:

- Leader can't open a connection to some acceptor.
- A non leader can't open a connection to a leader while asking to start a new instance of Paxos.

The first situation is easily dealt with, the leader simply ignores the failure and contacts acceptor on some other node and keeps doing this till it gets a majority. The second situation means that some node can't communicate with the leader. This could either be becuase the leader is down or because the communication link is broken, in either case, the node which gets this exception, does a broadcast to all other nodes to elect a new leader whose id is old_leader + 1. Elect new leader algorithm just udpates the id of leader on each node and also increases the value of current_proposal_number by 1. Further, if a node sees that its the new leader, it calls PrepareFuture on a majority of nodes. PrepareFuture updates the list of promised proposal number for all future instances to **current_proposal_number * 1000 + new_leader_id**. This ensures that if the older leader comes up in the future, it will always see a proposal number promised to be greater than what it proposes and thus understand that it is not the leader anymore. When this happens the older leader figures out who the new leader is (Figuring out the new leader is easy enough since we maintain a convention of proposal numbers to end with the leader id).

2 Assumptions

- For the sake of the assignment, we assume that the number of nodes is less than 1000.
- Reconfiguration and recovery is not supported.

- Client does the error handling itself. i.e. if the there is an error in the communication between client and a node, the client implements the retry/ignore logic.

3 Running the code

To ease the starting of the servers and brokers, we are submitting a few scripts.

- *start_nodes.py*: Starts as many nodes (broker and paxos handlers) as given via argument. Usage:

  ```
  start_nodes.py -l NUM_LOCKS -n NUM_NODES
  ```

  This will start a bunch of nodes as servers running on different ports on the localhost. And will also output the port numbers for the broker and paxoshandler on each node. After this you can just run the following commands on the client:

  ```
  Broker-remote [-h host[:port]] Lock mutexNo ClientId
  ```

  Or

  ```
  Broker-remote [-h host[:port]] Unlock mutexNo ClientId
  ```

- *test-failed-message.sh*: Script to simulate failed messages. (Automatically calls start_nodes.py)
  This script starts 3 nodes and runs a few client requests and then issues an elect new leader (similar to what a node would do in case of message failure).

- *test-kill-leader.sh*: Script to kill a leader and see if the Paxos nodes are still working.
  This script starts 3 nodes and after running a few client requests, kills the leader, then tries to run a few other commands.

- *test-kill-2-leaders.sh*: Script to kill one leader after another to see if the Paxos nodes are still working.
  - Starts 5 nodes.
  - Runs a few client requests.
  - Kills the leader.
  - Runs some more client requests.
  - Kills the second leader.
  - Runs some more client requests.

- *test-kill-2-nodes.sh*: Script to kill non-leader nodes to see if the Paxos nodes are still working.
  - Starts 5 nodes.

  - Runs a few client requests.
  - Kills a non-leader.
  - Runs some more client requests.
  - Kills another non-leader.
  - Runs some more client requests.

- *test-gap.sh*
  - Starts 5 nodes.
  -
  -
  -
  -