Simulating the Raft Consensus Protocol

Rex Fernando

Dec 17, 2015

Goals

Problem: Comparison of Raft and Paxos

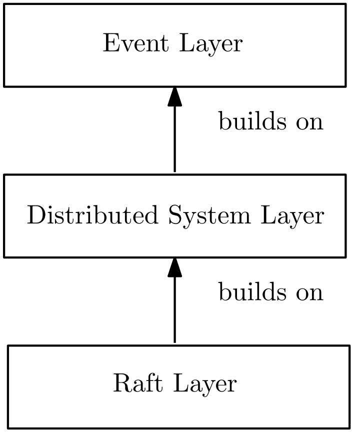
- Behavior in edge cases
- Ease of implementing

Proto-Goals

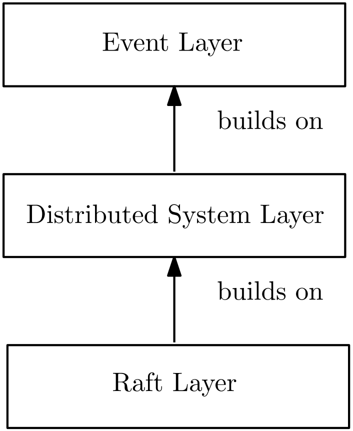
Problem: Build a useful simulation framework

- Should be able to simulate Raft and Paxos
- For now, only Raft
- Goal: generality

Solution/Outline



Solution/Outline



- Sampling of each piece
- Example Simulation Runs

Event layer

- Input: sequence of events, handler, starting state
- Handler: (State, Event) -> (Modified state, List of events to emit)
- Ouput: sequence of events, states ("filled out")

Event layer

- Input: sequence of events, handler, starting state
- Handler: (State, Event) -> (Modified state, List of events to emit)
- Ouput: sequence of events, states (“filled out”)

Example

Event layer

- Input: sequence of events, handler, starting state
- Handler: (State, Event) -> (Modified state, List of events to emit)
- Ouput: sequence of events, states (“filled out”)

Example

- Input
- Events: [Switch on at time 0, Switch off at time 1]
- Starting state: Light is off
- Handler:
  - If switch is switched on at time t, emit light-on event at t+0.01.
  - If switch is switched off at time t, emit light-off event at t+0.01.

Event layer

- Input: sequence of events, handler, starting state
- Handler: (State, Event) -> (Modified state, List of events to emit)
- Ouput: sequence of events, states ("filled out")

Example

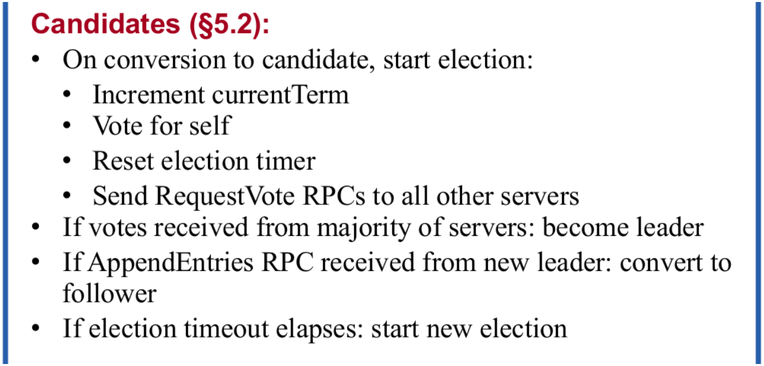
- Input
- Events: [Switch on at time 0, Switch off at time 1]
- Starting state: Light is off
- Handler:
  - If switch is switched on at time t, emit light-on event at t+0.01.
  - If switch is switched off at time t, emit light-off event at t+0.01.

- Output
- Events: [Switch on at time 0, Light on at time 0.01, Switch off at time 1, Light off at time 1.01]

Dist. System Layer

- System now consists of *n* machines
- Input:
- Sequence of events
- Handlers and starting states *for each machine in the system*
- *Global system behavior description*
  - Network
  - Crashes
  - Input into the system
- Ouput: sequence of events, states

Raft Layer



Candidates (§5.2):

- On conversion to candidate, start election:
  - Increment currentTerm
  - Vote for self
  - Reset election timer
  - Send RequestVote RPCs to all other servers
- If votes received from majority of servers: become leader
- If AppendEntries RPC received from new leader: convert to follower
- If election timeout elapses: start new election

Raft Layer

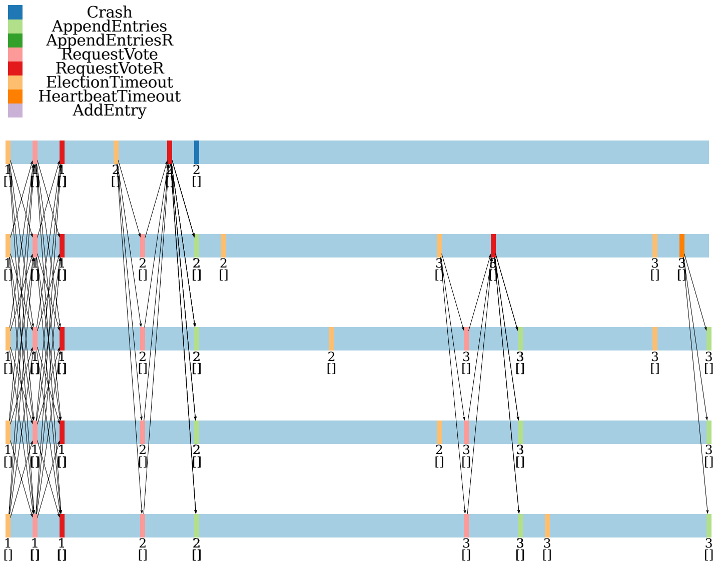
```haskell
-- Response handler
raftHandler (ReqVR term voteGranted) = do
  state <- get
  if term > (currentTerm state) then do
    becomeFollower term (-1)
    else if voteGranted then do
      put state { votesForMe = (votesForMe state)+1 }
      state <- get
      if (votesForMe state) >= quorum then
        becomeLeader
        else return ()
      else return ()
```

Crash after Elect

```haskell
global (Event time (Receive _ leader  _
                  (AppE _ _ _ _ _ _))) ms = do
  alreadyCrashed <- get
  if not alreadyCrashed then do
    crash leader
    put True
    else return ()
  sendAllMessages time ms

sendAllMessages time ms =
  mapM_ (flip send (time+delay)) ms
```

Crash after Elect

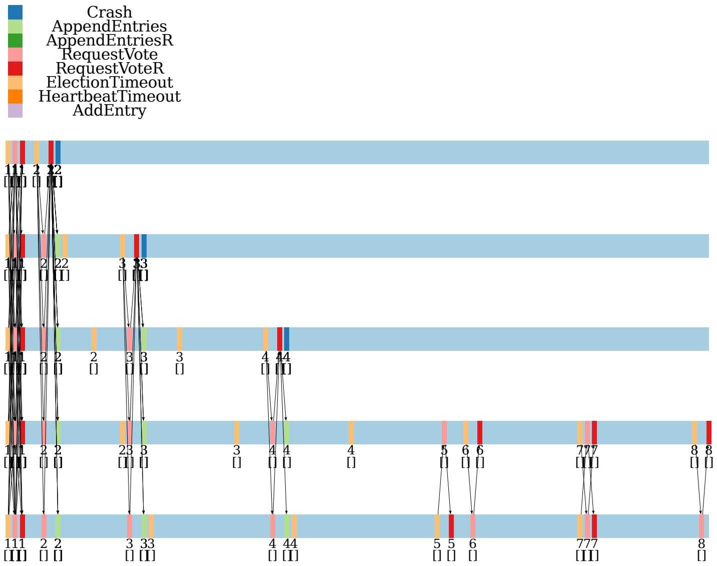


Crash after Elect 2

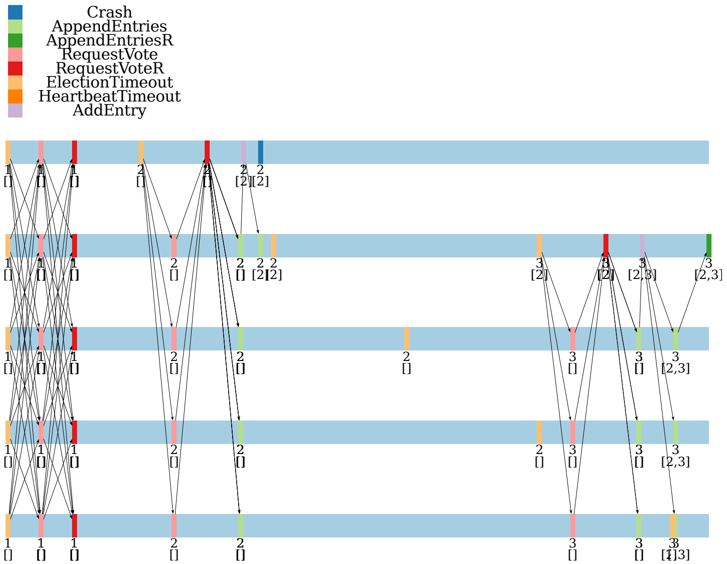
```
global (Event time (Receive _ leader  _
                   (AppE _ _ _ _ _ _))) ms = do
  alreadyCrashed <- get
  if not alreadyCrashed then do
    crash leader
    --put True
    else return ()
  sendAllMessages time ms

sendAllMessages time ms =
  mapM_ (flip send (time+delay)) ms
```

Crash after Elect 2



Crash during log write



Summary

Simulation framework

- Describe a distributed system (in this case Raft)
- Give custom “global behavior”
- Visualize results