Pure consensus in a world full of failure

Conrad Parker
fp-syd:20150325

Wither consensus?

Failures

- Non-Byzantine failures
- Byzantine failures
- Organizational failures

Not just sockets

- RobustIRC
- Unroutable networks
- Layer 3: BGP opaque data payloads

Not just storage

- Probabilistically Bounded Staleness
- Instrumentation
- Monte Carlo simulation

Abstractions

- Protocol: client/server and node/node RPC
- Storage: persistent logs
- Network: communications channel

A protocol abstraction

```
class Protocol p where

    type Request p :: *

    type Response p :: *

    step :: p -> Request p

         -> (p, Maybe (Response p))
```

A generic protocol server

```
serveOn :: (Protocol p, Serialize (Request p), Serialize (Response p))
        => PortID -> p -> IO ()
serveOn port p0 = do
    s <- listenOn port
    forever $ do
        (h, addr) <- S.accept s
        stream <- mkSocketStream h
        forkIO (loop stream p0 `finally` S.sClose h)
where
  loop stream p = do
    cmd <- Stream.runGet stream get
    let (p', m'rsp) = step p cmd
    case m'rsp of
        Just rsp -> Stream.runPut stream $ put rsp
        Nothing -> return ()
    loop stream p'
```

A store abstraction

- Log of state machine changes on each node

```
class Store s where

    type Value s :: *

    -- | Query the value at a given index
    query :: Index -> s -> m (Maybe (Value s, Term))

    -- | Store a value at a given index
    store :: (Foldable t, Monad m)
          => Int -> t (Value s) -> Term -> s -> m s

    -- | Mark values up to the given index as committed
    commit :: Monad m => Index -> s -> m s

    -- | Delete a given entry and all that follow it
    truncate :: Monad m => Index -> s -> m s
```

A simple client-server

- Client RPC: get, set values
- Sleep, Debug (dump state etc.)

```
step t cmd = case cmd of

          CmdSet k v ->
    let s' = store [v] (Term 0) (ts t) in
    (t{ts=s'}, Just $ RspSetOK k v)
CmdGet k -> let rsp = case query 0 (ts t) of
                          Just (v, _) -> RspGetOK k v
                          Nothing -> RspGetFail k
            in (t, Just $ rsp)
CmdSleep n -> (t, Nothing)
```

A replicated server

- Introduce a new Node communication

```
NodeSet k v
```

- Mirror client Set commands

Raft nodes

- Client talks with (who it thinks is) leader node
    - Non-leader responds with redirect message
- Leader translates into Raft RPC

Node state machine

- Nodes react based on:
  - Client requests (Get, Set)
  - Raft RPC from other nodes (AppendEntries, RequestVote)
  - Timeouts
- Reactions include
  - Sending out new Raft RPC requests
  - Sending Client and Raft RPC responses
  - Interacting with Store API

AppendEntries

```
step receiver (AE AppendEntries{..})

    -- Reply False if term < currentTerm

    | aeTerm < term = (receiver, Just . AER$ AppendEntriesResponse term False)

    | otherwise = do

        -- Reply False if log doesn't contain an entry at prevLogIndex

        -- whose term matches prevLogTerm

        t <- snd <$> Consensus.query prevLogIndex s

        if (t /= Just prevLogTerm)

          then return (receiver, AER$ AppendEntriesResponse term False)

          else do

            -- If an existing entry conflicts with a new one (same index but

            -- different terms), delete the existing entry and all that follow it.

            when (t /= aeTerm)

                truncate prevLogIndex s

            -- Append any new entries not already in the log

            store ix entries aeTerm s

            return (receiver, Just . AER$ AppendEntriesResponse aeTerm True)

where
```

RequestVote

```
-- Follower receiving RequestVote

step receiver@(RaftFollower p@RaftPersistentState{..} vol) (RV RequestVote{..})

    -- Reply False if term < currentTerm

    | rvTerm < currentTerm

      = (receiver, Just. RVR$ RequestVoteResponse currentTerm False)


    -- If votedFor is null or candidateId, and candidate's log is at

      -- least as up-to-date as receiver's log, grant vote

      | (votedFor == Nothing || votedFor == Just candidateId)

        && lastLogTerm <= currentTerm

        = (RaftFollower granted vol, Just . RVR$ RequestVoteResponse rvTerm True)

  where

    granted = p { votedFor = Just candidateId }
```

Next steps

- Protocol step function in a restricted monad that can only interact with storage and initiate timeouts
- Replay / instrumentation
- Membership changes
- http://github.com/kfish/raft