Raft
Implementing Distributed Consensus with Erlang
Outline

- Goals
- The consensus problem
- Outline RAFT algorithm
- Implementing in Erlang
Goals

What I want you to get out of this talk?

- Understand core ideas in RAFT
- Erlang / OTP as a tool for building systems
- Build your own implementation
Consensus

In a distributed system, agreement among multiple processes on a single data value, despite failures.

Once they reach a decision on a value, that decision is final.
Potential Use Case

- Configuration Management
- Distributed Transactions
- Distributed Lock Manager
- DNS and Resource Discovery
Raft is a consensus algorithm that is designed to be easy to understand.

Goals

- Design for understandability
- Strong leader
- Practical to implement
Messages

- RAFT only needs 2 messages.
- RequestVote includes term
- AppendEntries includes term and log entries
- Term acts as a logical clock
States

3 states a node can be in.

- Follower
- Candidate
- Leader
• Only a single leader within a cluster
• Receives commands from client
• Commits commands to the log
Follower

- Appends commands to log
- Votes for candidates
- Otherwise passive
Candidate

- Initiates Election
- Coordinates Votes
Leader Election

- Follower
- Candidate
- Leader

States and Transitions:
- Follower starts up
- Candidate times out and restarts election
- Candidate times out new election
- Candidate gets majority of votes
- Leader times out and step down
- Leader steps down

Process:
1. Follower starts up
2. Candidate times out and restarts election (loop)
3. Candidate times out new election
4. Candidate gets majority of votes
5. Leader times out and step down
6. Leader step down (loop)
Log Replication

Leader

AppendEntries
add 1, index 0

F1

add 1, index 0

F2
Log Replication

Leader

Ok

F1

Ok

F2

add 1

add 1

add 1

add 1
Log Replication

Leader
Executes command

F1

F2

Log Replication

- Leader
  - AppendEntries
    - add 4, index 1
  - add 1
  - add 4

- F1
  - add 1
  - Executes command

- F2
  - add 1
  - Executes command
  - add 4, index 1
RAFT Summary

- 2 types of messages, RequestVote and AppendEntries
- 3 states, Leader, Follower and Candidate
- Save Entries to persistent log
Erlang

- Functional language
- Fundamentally a concurrent language
- Actor model as basic abstraction
- No shared state between actors
- OTP behaviours like supervisors and gen_fsm
- Location independent message sending
Implementation Overview

- [github.com/tmcgilchrist/sloop](https://github.com/tmcgilchrist/sloop)
- [github.com/andrewjstone/rafter](https://github.com/andrewjstone/rafter)
- Each node has 2 supervised behaviours
- [gen_fsm](https://www.erlang.org/doc/man/gen_fsm.html) implementing the consensus protocol
- [gen_server](https://www.erlang.org/doc/man/sys_gen_server.html) wraps the log store
- passes erlang terms as messages
• state machine implements leader election and log replication

• each state is a function with multiple clauses

```haskell
follower(timeout, State) ->
    reset_timer(election_timeout()),
    request_votes(State),
    {next_state, candidate, State};
follower(_Event, Data) ->
    {next_state, follower, Data}.

candidate({vote_rpc, From}, State) ->
    Responses = store_vote(From),
    case quorum_reached(Responses, State) of
        true ->
            NewState = become_leader(State),
            {next_state, leader, NewState};
        false ->
            NewState = State#{state{responses=Responses}},
            {next_state, candidate, NewState}
    end;
candidate(_Event, Data) ->
    {next_state, candidate, Data}.
```
Supervisors

- sloop_sup
  - sloop_store
  - sloop_fsm
    - sloop_state
    - sender
Implementations

- raftconsensus.github.io
- github.com/tmcgilchrist/sloop
- github.com/andrewjstone/rafter
- github.com/goraft/raft
Summary

- Defined Distributed Consensus
- Looked at core ideas of RAFT
- Erlang suits distributed systems
- Map Erlang to RAFT
Thanks!