# Assignment 2: RAFT state machine

**Due date: Sat Feb 20 midnight. (two days before the midsem)**

**First checkpoint due Feb 6 (Sat. midnight).**

## The Big Picture

The figure below shows the file system using the services of a raft layer (the "node"). The Raft node embeds the Raft state machine, which in turn encodes the consensus algorithm.

The split between the Raft node and the state machine code is as follows. The state machine encodes the algorithm details, without worrying about disk logs, persistence, sockets, configuration files, URLs etc. The Raft node supplies that systemic framework. It is like the split between a processor and the motherboard that provides all ancillary support.

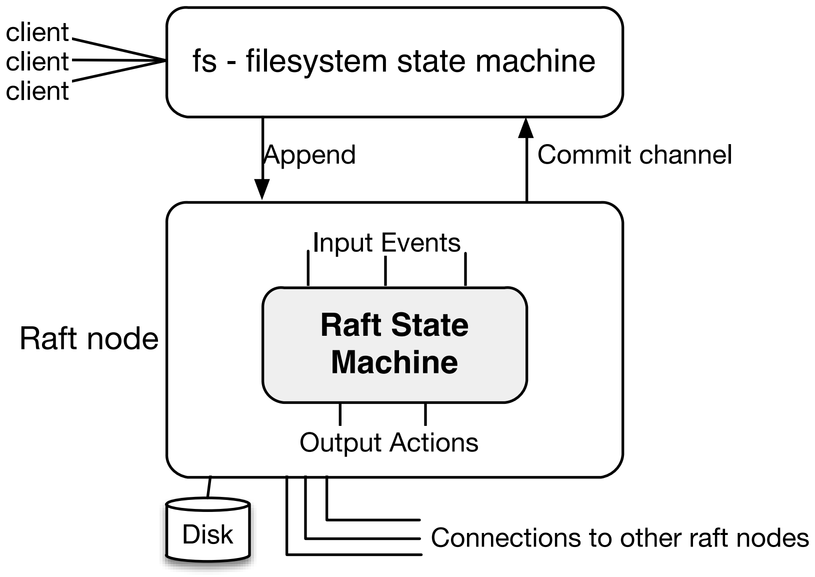

Figure 1: Raft state machine embedded in a Raft node'

**In this assignment, you will build the state machine (shaded) and test a single instance of it**. The next assignment will be to wrap it up in a suitable Raft node implementation, but don't worry about that for now.

The job of the state machine is to take input *events* (like 'timeout', 'AppendEntries msg' etc), and produce output *actions* (like 'send VoteReq to node 3', 'wake me after x milliseconds etc"). It does not perform the actions itself; that is the job of the Raft node in which this state machine is embedded.

The reason for this architecture is to make testing deterministic. The idea is to start the state machine in a particular state, throw events at it and see whether it changes to the right state and whether it produces actions as expected. This allows us to create scenarios that otherwise occur incredibly rarely in practice.

In this document, the term "state machine" is reserved for the Raft node's internal state machine. Do not confuse it with the application-level state machine, in this case the fs file system. When the paper refers to replicated state machines, it is fs they are referring to. In particular, you can ignore all references to `lastApplied` in the paper,since it tracks which events have been handed over to the layer above.

## State machine details.

The paper gives all the important state variables that the machine needs to manipulate. Your implementation may need others.

### Input events

The state machine will have to deal with the following events. In general, these events can come at any time, so regardless of which state the machine is in, it must react suitably to all events. All requests must be responded to – dropping an unexpected message is not acceptable.

1. `Append(data:[]byte)`: This is a request from the layer above to append the data to the replicated log. The response is in the form of an eventual Commit action (see next section).
2. `Timeout` : A timeout event is interpreted according to the state. If the state machine is a leader, it is interpreted as a heartbeat timeout, and if it is a follower or candidate, it is interpreted as an election timeout.
3. `AppendEntriesReq`: Message from another Raft state machine. For this and the next three event types, the parameters to the messages can be taken from the Raft paper.
4. `AppendEntriesResp`: Response from another Raft state machine in response to a previous AppendEntriesReq.
5. `VoteReq`: Message from another Raft state machine to request votes for its candidature.
6. `VoteResp`: Response to a Vote request.

**Output Actions**

1. `Send(peerId, event)`. Send this event to a remote node. The event is one of `AppendEntriesReq/Resp` or `VoteReq/Resp`. Clearly the state machine needs to have some information about its peer node ids and its own id.
2. `Commit(index, data, err)`: Deliver this commit event (index + data) or report an error (data + err) to the layer above. This is the response to the Append event.
3. `Alarm(t)`: Send a Timeout after `t` milliseconds.
4. `LogStore(index, data []byte)`: This is an indication to the node to store the data at the given index. Note that data here can refer to the client's

**First checkpoint due Sat 6.**

A document with the pseudocode for the state machine that outlines for each state, how it reacts to each input event. The state machine may totally ignore the event, or change some of its internal variables and/or produce some actions. Here is an example of the kind of documentation I expect to see. Keep it as close to code as you can, so that it is easily transferable to the next.

```
State : Follower

on VoteReq(from, term, candidateId, lastLogIndex, lastLogTerm)
   if  sm.term <= msg.term &&
      sm.votedFor == nil or msg.candidateId,
           if candidate log is at least as up-to-date as receiver's log,
                 sm.term = msg.term
                 sm.votedFor = msg.candidateId
                 action = Send(msg.from, VoteResp(sm.term, voteGranted = yes))

   Otherwise, reject vote:
        action = Send(msg.from, VoteResp(sm.term, voteGranted = no))
```

Make sure you cover all edge cases mentioned in the document. Feel free to add other actions or events as you see fit. But talk to me about it.

## Implementation choices

After the first checkpoint, you can convert the document into working code. Choose from a variety of implementations depending on the event delivery mechanism, generic or specialized event types and your interpretation of the spec. Examples:

- Event delivery as a method invocation. Processing an event results in a (possibly empty) array of actions.

```go
func (sm *StateMachine) ProcessEvent(Event ev) []Action {
      switch sm.status {
          case follower: followerEvent(ev)
          ...
}
```

- Event delivery and actions via channels. In the following example, there are multiple input channels and one output channel.

```
func (sm *StateMachine) eventLoop() {
      for {
          select {
              case  appendMsg := <- clientCh:
                     ....
                     actionCh <- for all peers p, Send{p, AppendEntries{..})
                     actionCh <- LogStore{..}
              case peerMsg := <- netCh:  // for all network messages from other raft instances
              case <- timeoutCh :
          }
     }
 }
```

- Typed vs untyped event channels or parameters.