1 Intro

1.1 Paxos vs Raft

- Raft separates leader election, log replication and safety
- Raft reduces the ways in which the servers can be different from one another
- 33 of 43 students could answer questions about Raft better than about Paxos

1.2 Novel features

- Strong leader - Raft's leader node is stronger than that of other algorithms
- Leader election - Raft uses randomized timers for elections
- Membership changes - Raft's mechanism for changing the set of servers in the cluster uses a new joint consensus approach where the majorities of two different configurations overlap during transitions. This allows the cluster to continue operating normally during configuration changes

2 Replicated state machines

Consensus algorithms typically have these properties:

- They ensure safety (Not returning to an incorrect result) for network delays, partitions, packet loss, duplication, and reordering
- They are fully functional (available) if any majority of servers is online
- They do not rely on timing (unsynchronized clocks and delays will can only affect availability)
- In most cases a command can be completed once a majority has responded

3 What's wrong with Paxos?

- Difficult to understand
- Poorly explained and documented
- Does not provide a foundation for a practical implementation
- Implementations begin with Paxos, discover its difficulty and develop a different architecture

4 Designing for understandability

Goals for the development of Raft:

- provide a complete and practical foundation for systems for reduced necessary design work
- Must be safe under all conditions
- Available during typical operating conditions
- most important: **understandability**
  - Problems broken down into easier questions
  - Reduce the number of possible states

5 The Raft consensus algorithm

Raft decomposes the consensus problem into 3 independent subproblems:

- Leader Election - a new leader must be chosen when an existing leader fails
- Log Replication - the leader accepts log entries and replicates them across the cluster
- Safety - If any node has committed an entry to the log, no other node may commit an entry to the same index

5.1 Raft basics

- Raft guarantees : figure 3
- All servers are always in one of three states:
  - Leader - Handles all client requests, falsely addressed requests are forwarded
  - Follower - Passive, create no requests, just respond to requests from leaders and candidates
  - Candidate - Used to elect new leaders
- Terms:
  - Time is divided into *terms* (last until leader is replaced)
  - Split votes end term with new elections called and no leader
  - Servers store a current term value
  - Current terms are exchanged every time servers communicate
  - Smaller current terms are updated to the larger value
  - Leaders and candidates revert to followers if they encounter a larger current term value
  - Requests are rejected if they have a stale term value

- Communication:
  - Servers communicate with Remote Procedure Calls (RPCs)
  - Two RPCs needed for consensus
    - RequestVote RPC - initiated by candidates
    - AppendEntries RPC - initiated by leaders to replicate log entries
    - See Log compaction for 3rd type (InstallSnapshot RPC)
  - RPCs are repeated if no response comes in too long a time

5.2 Leader election

- Raft uses heartbeats to trigger elections
- At startup servers are followers
- Followers remain followers as long as they receive valid RPCs
- Leaders send heartbeats(empty AppendEntry RPCs) to maintain their authority
- If a server's election timeout runs out before a heartbeat comes it initiates an election
- Elections occur as follows:
  1. A follower increments its term and transitions to the candidate state
  2. It votes for itself and sends RequestVote RPCs to all other servers in a cluster
  3. The candidate continues in this state until one of three conditions are met:
     (a) it wins the election
     (b) another server wins the election
     (c) a period of time elapses without a winner
  4. Once a candidate wins it sends a heartbeat to other servers to establish itself and prevent further elections
- Rules for elections:
  - Elections are won if a candidate receives votes from a majority of servers in the cluster for a given term
  - Servers will vote for at most one candidate per term
  - Servers vote on a first come first serve basis
  - If a candidate receives an AppendEntries RPC from another server claiming to be leader, it accepts the authority of the leader if the leader's term is equal or greater than its own term and returns to being a follower
- Split votes:

  - Split votes can occur if multiple followers become candidates at the same time
  - The term is ended and new electins with the next term are performed
  - To prevent further split votes election timeouts are randomized (doesn't just happen after a split vote)

5.3 Log replication

- The server services client requests
- The request contains a command to be executed by the replicated state machines
- The leader appends the command to the log and sends AppendEntries RPCs to the other servers to replicate the entry
- When the entry is safely replicated, the server applies teh change to its state machine and returns the result
- AppendEntries RPCs are sent continuously to followers until they are added to the log
- Logs store a state machine command and term number
- Term numbers in logs detect inconsistencies
- Each entry also has a log index
- The leader decides when it is safe to apply (or commit) a log entry
- Raft guarantees that committed entries are durable and will eventually be executed by all of the available state machines
- A log entry is committed once the leader that created the entry has replicated it on a majority of the servers
- The leader tracks the highest known index and send it with every AppendEntry RPC so other servers can commit entries and apply commands

5.3.1 Log Matching Property from Figure 3:

- If two entries in different logs have the same index and term then they store the same command.
- If two entries in different logs have the same index and term, then the logs are identical in all preceding entries.
  - When sending an AppendEntries RPC, the leader includes the index and term of the entry in its log that immediately precedes the new entries.
  - If the follower does not find an entry in its log with the same index and term, then it refuses the new entries.
  - Therefore, whenever AppendEntries returns successfully, the leader knows that the follower's log is identical to its own log up through the new entries.

5.3.2 Log inconsistency

- Inconsistencies occur when the leader fails or becomes unavailable
- Followers may be missing entries, have entries the leader doesn't or both
- The leader forces followers to duplicate (and overwrite) their logs to match the leaders'
- Process:
  1. The first common log is found
  2. All entries after that in the follower are deleted
  3. Starting after the first common index teh entries are sent to the follower to catch it up
- the leader maintains a *nextindex* to remember which entry need to be sent to the followers next
- *nextindex* is determined through AppendEntry RPCs and their consistency checks (decrementing until the check passes)

5.4 Safety