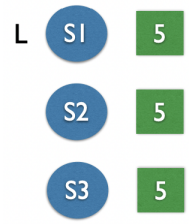
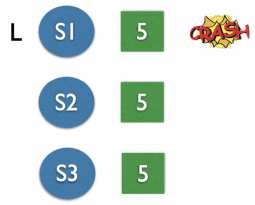
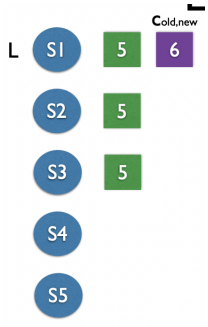
CAS CS 350
Lec 14

Raft

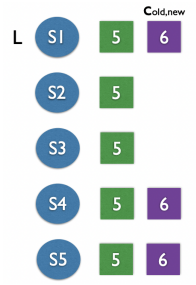
1. Overview
    a. Leader election
        i. Server becomes candidate → increase the term → vote for itself → send random election time out → send out vote request to other servers → waits until they get response (get majority of vote and become leader and send heartbeats to other candidates), wait until other candidate becomes leader (receive heartbeat from other candidate and become follower), if nothing happens, then restart the election
    b. Commit an entry on log → meaning
        i. If something is committed, you can reply to the client (committed entry must remain in the majority of the raft servers) at the same term it was initiated or subsequent command is committed
    c. If candidate receives a RPC that has a term higher than my term → do what?
        i. Transition to follower, change the term to the higher term, vote for that candidate
2. Configuration Changes
    a. What happens when:
        i. One or more servers cannot recover and we need a replacement?
        ii. We want to scale out/in by adding/removing servers?
    b. “Stop the world” and restart
        i. Bring the server down, bring the snapshot into a local disk, change the configuration, and restart the application
            1. System becomes unavailable for a long time (cannot receive client request) → want to avoid
    c. Can we do better?
        i. Two-phase approach: first phase disables old configuration, second phase enables new
            1. At some point in time, do buffer requests for a short time and store it in queue
            2. Enable second configuration when there are no more requests
            3. Allow second configuration to handle the requests in queue
            4. This pauses some part of execution and continue (no need to shutdown the servers and becomes more available)
        ii. Raft’s solution is “joint consensus”: allows different servers to transition to the new configuration at different times

3. Joint Consensus
    a. First phase: both configurations are active
        i. Two sets of servers that define separate majorities
        ii. Any server from either configuration can be a leader (still one leader)
        iii. Elections and commits require majorities from both configurations
    b. Second phase: all servers have switched to the new configuration
    c. Cluster configurations are treated as special entries
    d. A server uses the latest (possibly joint) configuration in its log, even if it is not committed (different servers can have different configurations here)
    e. Once a configuration is committed, all servers must use this configuration
4. Example
    a. 

    b. Three servers → and S1 is the leader
    c. 

    d. Leader crashes and there is a new leader
    e. 

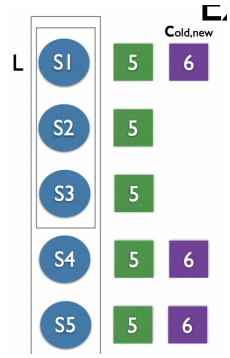
    f. Two more servers that enter the server (there is a new majority to 3 instead of 2)
    g. S1 add command 6 and asks request to followers to do the same

h. 

i. The command arrives to S4 and S5 first

j. Can it be committed? → No

   i. Does not appear in the majority in the old configuration (from S1, S2, and S3, there is a minority)

k. 

l. Re-configuration command appears in the new majority, not the old

m. 

n. Leader S1 crashes

o. 

p. S1 becomes the leader again after winning election with term 7 and receives a new client request

q. 7 cannot be committed → it appears only in minority (1 out of 5)

r. 6 cannot be committed until 7 is committed → because even though 6 meets majority, it meets majority at term 7 (wait for subsequent command to be committed first)

s. 

t. Leader sends append RPCs to the servers (command 7 still cannot be committed → not majority yet in the new configuration (second configuration))

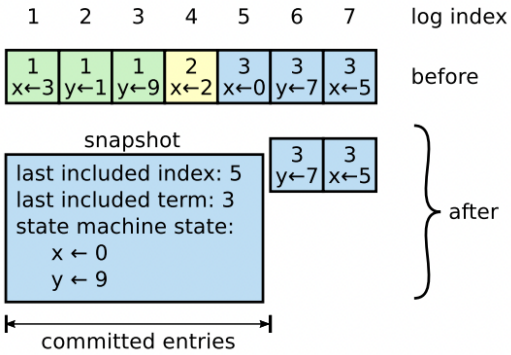
u. 

v. Re-configuration command is committed (appears in majority of both configurations) → 7 can be committed and therefore 6 is also committed

w. Server uses both configurations here

5. Persistence
    a. Server state breaks into two parts
        i. Election
        ii. Persistent state
6. Persistence and Performance
    a. Disk: 10ms write, SSD 0.1 ms – RPC << 1ms (in datacenters)
    b. So if every op needs a disk write then limited to speed of write
    c. Better performance
        i. Batch many client ops into each RPC
        ii. Faster storage non-volatile storage
7. Log compaction and snapshots
    a. Snapshots are required to prevent logs from growing indefinitely → prevent running out of memory
    b. Raft uses snapshotting to keep the size of the log bounded
    c. The entire current state is written to reliable storage as a snapshot

d. Snapshots use copy-on-write so that the server can still reply to requests during snapshotting
e. Snapshots can be periodic or based on an upper bound in the log size
f. Leader decides what is snapshotted or not
g. Snapshot only includes committed entries, taken asynchronously (do it independently of others)
h.



i. Each server periodically snapshots to bound resource consumption
j. Leader may need to use its snapshot to update stale clients instead of sending individual commands
k.

InstallSnapshot RPC

Invoked by leader to send chunks of a snapshot to a follower. Leaders always send chunks in order.

Arguments:

term	leader's term
leaderId	so follower can redirect clients
lastIncludedIndex	the snapshot replaces all entries up through and including this index
lastIncludedTerm	term of lastIncludedIndex
offset	byte offset where chunk is positioned in the snapshot file
data[]	raw bytes of the snapshot chunk, starting at offset
done	true if this is the last chunk

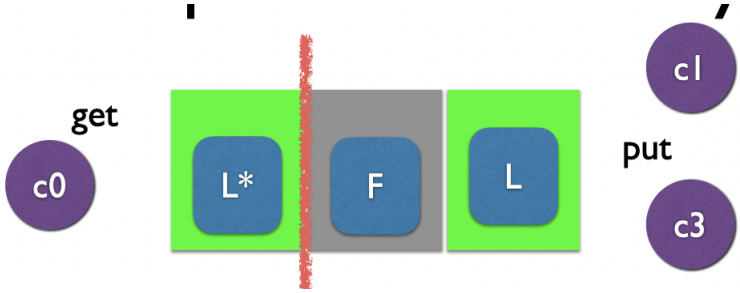
Results:

term	currentTerm, for leader to update itself

Receiver implementation:

1. Reply immediately if term < currentTerm
2. Create new snapshot file if first chunk (offset is 0)
3. Write data into snapshot file at given offset
4. Reply and wait for more data chunks if done is false
5. Save snapshot file, discard any existing or partial snapshot with a smaller index
6. If existing log entry has same index and term as snapshot's last included entry, retain log entries following it and reply
7. Discard the entire log
8. Reset state machine using snapshot contents (and load snapshot's cluster configuration)

8. Client Behavior Section 8
    a. Client loops: retry until success – leads to duplicate entries on leader change
    b. Duplicate detection: Raft can send to service repeat commands
        i. Service required to keep history of commands to reject duplicates
            1. Replica must maintain a duplicate table (as leaders can change)
9. Important: read-only
    a. 

    b. Can the leader serve read-only requests locally?
        i. No because it is a strong consistency model (linearizability)
            1. It means you cannot see stale data
    c. 

    d. If read's did not go to log and there is partition, we no longer need majority to reply so we do ;-) and why could this be a problem?
        i. There might be a server who thinks it is the leader and respond directly to client
10. Linearizability
    a. Raft supports linearizable semantics
    b. Clients cannot read stale data
11. Questions
    a. Which feature of the Raft algorithm facilitates linearizability in practice?
        i. They have to go through the leader everytime
        ii. GFS → read data from any replica (can see stale data)
        iii. In this case, all client request have to go through a single server
    b. Consider an implementation of a Raft service that is slow for whatever reason. A colleague of yours suggests that you increase the number of replicas. Will this affect the Raft service performance (latency/throughput) and, if so, how?
        i. The responses are concurrent
        ii. It improves fault-tolerance (Can tolerate more failures) → does not mean you improve latency

iii. Increasing the number of servers increases the replication level and provides “better” fault-tolerance but will not improve Raft’s performance because the leader needs to wait for more acks

c. How can we improve latency/throughput of a Raft service then?
    i.