CSCI 1380  Day 14: Raft
Last Class

Passive V. Active Replication

Today

* Picking btw Passive/Active Replication
* Raft (Passive )
  * node information
  * leader election (safety/liveness)
  * client protocol
  * Raft safety (committed data persists)
    * changes to leader election/log commitment
Passive

- Leader/followers
- P2Ps have different roles
- Code can be non-deterministic
- FE
- RM

Active

- All RM are equal
- FE
- RM
- Requires code to be deterministic
- Key issues: lots of messages & high overhead

Key issues: under failure it takes time to find new leaders & commit

Passive ⇒ data centers/clusters
Network latency ⇒ 41ms

Wide area network (Internet)
Network latency ⇒ 40-100ms
Geo distributed database/storage will use active replication

Can you tolerate the latency overhead of leader failures/leader election?
“The dirty little secret of the NSDI community is that at most five people really, truly understand every part of Paxos.” – NSDI reviewer

“There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system...the final system will be based on an unproven protocol.” – Chubby authors
**Consensus** = all nodes in the algorithm agree on some value

Chandy-Lamport
"agreement on what is in a snapshot"

"raft cluster" \( \Rightarrow \) \( N \) nodes that participate in protocol
1 leader
\( N-1 \) followers

"Node" \( \Rightarrow \) log + state machine + raft protocol code
\( \downarrow \) list of client requests

log of all requests
Leader Elections

1. Missed heartbeat
2. New term
3. Candidate
4. Vote for self
5. Request votes from all other nodes
6. If you get most votes

Note: Roles

Passively receiving request from leader & putting in logs

Because \( N \) is well defined every node can calculate majority

Leader

- Elections

1. Missed heartbeat
2. New term
3. Candidate
4. Vote for self
5. Request votes from all other nodes
6. If you get most votes

Node

- Holes

- Assume the leader is dead if no heartbeat

- If no heartbeat from leader, send request for votes
- If you get majority, step down if there's another leader with a higher term

- No majority within timeout

- No heartbeat from leader
Terms

1. elections: candidate is waiting

2. normal operations: candidate has become leader & can service request from client

3. Term with no leader elected

4. Split vote so no leader elected

5.

Cluster (N=3)

- Candidate A was leader
- Candidate A failed

- Candidate B
- Candidate C

If Node B & C detect failure at some time, then they both become candidates

Majority = 2 votes
How to ensure a leader (liveness)

(1) random delay into state transition (follower → candidate)!

(2) different weights (how does this impact)

ordering to break ties

make each follower use different T/O

Add randomness to avoid multiple notes becoming candidates at the same time!!

Each note randomly set Heartbeat Time Out => [Tp] [2T]

Time to send messages to everyone (broadcast)
Raft Election Properties

(1) **Liveness** ⇒ eventually you will have a leader
    ⇝ random timeouts

(2) **Safety** ⇒ at most one viable leader
    ⇝ each node only votes once
    + only the candidate with majority votes wins

Viable = process client requests

when there are partitions ⇒

n/w partition

these nodes can communicate with each other but not across partitions
How many failure can happen in your cluster?

At most 3 servers can fail in cluster.

\[ N = 2F + 1 \]
haft's clients are the front end servers

1. register
2. if random node is not leader then redirect
3. register with leader to get client ID
4. all requests include client ID + request ID
Client side: request retries → request
Leader side: duplicate suppression → response reply

1) duplicate suppression
   → checking log for request ID

2) responds with a cached response (response replay)

Which semantics requires the three techniques?

- at least once
- at most once
- exactly once

<table>
<thead>
<tr>
<th></th>
<th>at least once</th>
<th>at most once</th>
<th>exactly once</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Raft claims:
- exactly once as long as client doesn’t fail
- not practical
Log Commitment

- client request

These steps are after leader is found & you already registered with the leader

1. get request (make sure new)
2. put in local log
3. replicate followers
4. after a majority of nodes have update logs
5. then entry is considered committed
6. respond to client
Raft

1. Leader election (liveness/safety)
2. Client interactions
3. Log (log commit)
4. Node (states/transition/data structures)