Lecture 7.

RAFT

(+ Info About The Exam
  + A Little Bit About Partial Synchrony)

Midterm

- Next week, in-person, during class
- Covers everything (including Raft) we have looked at thus far
- Open book
  - No limit on how many pages, etc
  - No laptop, tablet or phone
  - Don't want to go around checking internet connectivity
  - Claim: Access to paper/notes has very little advantage
  - Understanding the "essence" more important
- Must work alone

Importante !!!!
Suggestion: Make sure you understand the things we really dived into & keep returning to

- Linearizability
- Why RSMs,
- "How" RSMs
- How to use RSM

- Fine to not know
details about the many proposals for exploiting partial synchrony
- Proof that linearizability is local

- If unsure about terminology to use when expressing answers
  - Just define them as you want
    - E.g., an event is X

- Write concise answers
  - More likely to be correct
  - Corollary: Do not hedge
- Most importantly: Try and have some fun

Regular Programming
Where we are

- RCM ← An abstraction for building replicated fault
**Why?** Because the proof said so?

- **Fault tolerance requirement**
  - Protocol must terminate even if 1 process fails

- **Asynchronous assumption**
  - Processes cannot distinguish between failed & slow processes
  
  Proof just delays messages repeatedly to prevent termination

**What if we add assumptions to async networks?**

- **Partial synchrony** — bound delay $\delta$
Why Does This Help?

**RAFT**

A Practical Consensus Protocol

Fast Path / No Failure

1. Decide Order
2. Replicate to Ensure Fault Tolerance

Failure Recovery

1. Make Sure Any Sufficiently Replicated Will Command is Not "Lost"
Fast Path

HAVE NOT TALKED ABOUT FAILURE MODEL YET

WANT TO TOLERATE $f$ FAILURES

How many replicas do we need?

(Just for log entries)
When is it safe to respond to Alice?

Leader recvd \( f+1 \) responses for an AE commit.

Rule: Any sufficiently replicated log entry should be eventually committed.
How Is This Used In Practice

1. Assume Commands Are Idempotent
2. Make Commands Idempotent

Rule: Any Sufficiently Replicated Log Entry Should Be Eventually Committed

Things To Do On Failure

- Follower fails
  - Nothing

- Leader fails
  - Decide what process should become leader (L')
  - Ensure L' log contains all sufficiently replicated entries

Why?

Ensuring L' log contains all sufficiently replicated entries
Ensuring $L'$ log contains an entry

**Approach 1.** $L'$ communicates and reconciles log entries from all live processes (two weeks from now)

**Approach 2.** $L'$ only elected if it contains all sufficiently replicated entries (RAFT today)

- **Leader Fails**
  - **Decide what process should become leader ($L'$)**

Note this is just consensus

Why?

Aim: Choose a process $L'$ as leader s.t.

$L'$ has all sufficiently replicated log entries
Definition: A log entry is "sufficiently replicated" if and only if replicated on $f+1$ replicas.

Steps:
1. Process $P$ suggests it wants to be leader includes entries it knows about in message.
2. Any process $Q$ on receiving message from $P$ votes iff:
   1. $Q$ has not voted for another process this term.
   2. $P$'s log has all entries $Q$ has.
3. $P$ becomes leader if it receives enough votes.

Note: "Enough votes" $\Rightarrow P$ must hear from at least $1/(f+1)$ replicas.

$\Rightarrow$ But $f$ can fail.
**Require:** \(2f+1\) Process To Survive \(f\) Failures + Votes From \(f+1\) Replicas

<table>
<thead>
<tr>
<th>Rep</th>
<th>Alive</th>
<th>Voted</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✔</td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>✔</td>
<td>X</td>
</tr>
<tr>
<td>C</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rep</th>
<th>Alive</th>
<th>Voted</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✔</td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>✔</td>
<td>X</td>
</tr>
</tbody>
</table>

Another approach's trade-off replication & votes

3\(f+1\) processes

\(f+1\) to commit

2\(f+1\) to elect

Why?

**Quorum Intersection**
Including "all entries P knows about" when requesting votes / checking P has all entries Q has.

Problem?
- Message size
- Compute time

How addressed

* Term

- Elections might not terminate

Why?
RECONFIGURATION

- Use a joint consensus (both old + new servers)
  - Commit = quorum in old + why?
  - Quorum in new

Re-use log for safety

Commit
Switching: Why?

Quorum Intersection

- Prevent Split Brain

\[ O \cap L \cap C \cap \omega \]

\[ A \cap B \cap C \cap D \]

\[ 3 \cap C \]

\[ 3 \cap 3 \]