ATTRIBUTION

- These slides incorporate material from:
  - Diego Ongaro and John Ousterhout
ANNOUNCEMENTS

Change in HW 8 command-line posted to course website
- Takes port number in addition to node number

No class Thursday
- Can use time to meet with group for HW 8

Correcting one HW 5 test case regarding deleting an already deleted file

Recall: Primary-Backup

• **Mechanism:** Replicate and separate servers

• **Goal #1:** Provide a highly reliable service

• **Goal #2:** Servers should behave just like a single, more reliable server
Extend PB for high availability

- Primary gets ops, orders into log
- Replicates log of ops to backup
- Backup executes ops in same order
- Backup takes over if primary fails

- But what if network partition rather than primary failure?
  - “View” server to determine primary
  - But what if view server fails?
    - “View” determined via consensus!

PB high availability via 2PC

1. \( C \rightarrow P: \) “request <op>”
2. \( P \rightarrow A, B: \) “prepare <op>”
3. \( A, B \rightarrow P: \) “prepared” or “error”
4. \( P \rightarrow C: \) “result exec<op>” or “failed”
5. \( P \rightarrow A, B: \) “commit <op>”

“Okay” (i.e., op is stable) if written to > ½ backups
View changes on failure

1. Backups monitor primary

2. If a backup thinks primary failed, initiate View Change (leader election)

3. Intuitive safety argument:
   - View change requires $f+1$ agreement
   - Op committed once written to $f+1$ nodes
   - At least one node both saw write and in new view

4. More advanced: Adding or removing nodes (“reconfiguration”)
RAFT OVERVIEW

1. Leader election (last week)
2. Normal operation (basic log replication)
3. Safety and consistency after leader changes
4. Neutralizing old leaders
5. Client interactions
6. Reconfiguration

Log Structure

- Log entry = < index, term, command >
- Log stored on stable storage (disk); survives crashes
- Entry committed if known to be stored on majority of servers
  - Durable / stable, will eventually be executed by state machines
Normal operation

- Client sends command to leader
- Leader appends command to its log
- Leader sends AppendEntries RPCs to followers

Once new entry committed:
- Leader passes command to its state machine, sends result to client
- Leader piggybacks commitment to followers in later AppendEntries
- Followers pass committed commands to their state machines

Normal operation

- Crashed / slow followers?
  - Leader retries RPCs until they succeed

- Performance is optimal in common case:
  - One successful RPC to any majority of servers
Log Operation: Highly Coherent

- If log entries on different server have same index and term:
  - Store the same command
  - Logs are identical in all preceding entries

- If given entry is committed, all preceding also committed

Log Operation: Consistency Check

- AppendEntries has <index,term> of entry preceding new ones
- Follower must contain matching entry; otherwise it rejects
- Implements an induction step, ensures coherency
Leader Changes

- New leader’s log is truth, no special steps, start normal operation
  - Will eventually make follower’s logs identical to leader’s
  - Old leader may have left entries partially replicated

- Multiple crashes can leave many extraneous log entries

<table>
<thead>
<tr>
<th>log index</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>s₂</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>s₃</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s₄</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s₅</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Safety Requirement

**Once log entry applied to a state machine, no other state machine must apply a different value for that log entry**

- **Raft safety property**: If leader has decided log entry is committed, entry will be present in logs of all future leaders

- Why does this guarantee higher-level goal?
  1. Leaders never overwrite entries in their logs
  2. Only entries in leader’s log can be committed
  3. Entries must be committed before applying to state machine

**Committed → Present in future leaders’ logs**
Picking the Best Leader

• Elect candidate most likely to contain all committed entries
  – In RequestVote, candidates incl. index + term of last log entry
  – Voter V denies vote if its log is “more complete”: (newer term) or (entry in higher index of same term)
  – Leader will have “most complete” log among electing majority

Committing Entry from Current Term

• Case #1: Leader decides entry in current term is committed
• Safe: leader for term 3 must contain entry 4
Committing Entry from Earlier Term

- **Case #2**: Leader trying to finish committing entry from earlier

- Entry 3 not safely committed:
  - $s_5$ can be elected as leader for term 5 (how?)
  - If elected, it will overwrite entry 3 on $s_1$, $s_2$, and $s_3$

New Commitment Rules

- **For leader to decide entry is committed**:
  1. Entry stored on a majority
  2. $\geq 1$ new entry from leader’s term also on majority

- Example: Once e4 committed, $s_5$ cannot be elected leader for term 5, and e3 and e4 both safe
Challenge: Log Inconsistencies

Leader changes can result in log inconsistencies

Possible followers

Leader for term 8

Possible followers:

1 2 3 4 5 6 7 8 9 10 11 12

(a) 1 1 1 4 4 5 5 6 6 6 6
(b) 1 1 1 4
(c) 1 1 1 4 4 5 5 6 6 6 6
(d) 1 1 1 4 4 5 5 6 6 6 7 7
(e) 1 1 1 4 4 4 4
(f) 1 1 1 1 1 2 2 2 3 3 3 3

Leader for term 7

Followers:

1 2 3 4 5 6 7 8 9 10 11 12

(a) 1 1 1 4
(b) 1 1 1 2 2 2 3 3 3 3

Repairing Follower Logs

- New leader must make follower logs consistent with its own
  - Delete extraneous entries
  - Fill in missing entries

- Leader keeps nextIndex for each follower:
  - Index of next log entry to send to that follower
  - Initialized to (1 + leader’s last index)

- If AppendEntries consistency check fails, decrement nextIndex, try again
Repairing Follower Logs

Leader for term 7

Before repair

After repair

Neutralizing Old Leaders

Leader temporarily disconnected

→ other servers elect new leader

→ old leader reconnected

→ old leader attempts to commit log entries

• Terms used to detect stale leaders (and candidates)
  – Every RPC contains term of sender
  – Sender’s term < receiver:
    • Receiver: Rejects RPC (via ACK which sender processes…)
  – Receiver’s term < sender:
    • Receiver reverts to follower, updates term, processes RPC

• Election updates terms of majority of servers
  – Deposed server cannot commit new log entries
Client Protocol

- **Send commands to leader**
  - If leader unknown, contact any server, which redirects client to leader

- **Leader only responds after command logged, committed, and executed by leader**

- **If request times out (e.g., leader crashes):**
  - Client reissues command to new leader (after possible redirect)

- **Ensure exactly-once semantics even with leader failures**
  - E.g., Leader can execute command then crash before responding
  - Client should embed unique ID in each command
  - This client ID included in log entry
  - Before accepting request, leader checks log for entry with same ID