ANNOUNCEMENTS

For PA3:

- Only one uploader will run (so no need to handle conflicts)
- You should always download the files (don’t scan/read local directories)
- Fixed typo in writeup about the ‘dd’ command

---

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>term</th>
<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></td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s₂</td>
<td></td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>s₃</td>
<td></td>
<td>1</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s₄</td>
<td></td>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s₅</td>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></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

Restrictions on commitment

Restrictions on leader election

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

41

42
• Case #1: Leader decides entry in current term is committed

• **Safe:** leader for term 3 must contain entry 4

• 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
Repairing Follower Logs

**Leader for term 7**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>(b)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**Followers**

- **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**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

**Before repair**

|   | 1 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 3 |

**After repair**

|   | 1 | 1 | 1 | 4 | 4 | 4 | 4 | 5 | 5 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |

47

48
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
WEB SIMULATOR/DEMO

https://raft.github.io/raftscope/index.html