Distributed Systems

To survive failures you need a raft...
Log Safety Requirement

Once in a log, no other node must apply a different value for that log entry

• Raft safety property:
  • If a leader has decided that a log entry is committed, that entry will be present in the logs of all future leaders

• This guarantees the safety requirement
  • Leaders never overwrite entries in their logs
  • Only entries in the leader’s log can be committed
Committing Entry from Current Term

• Case #1/2: Leader decides entry in current term is committed

1 2 3 4 5 6

s1 1 1 2 2 2
s2 1 1 2 2
s3 1 1 2 2
s4 1 1 2
s5 1 1

Leader for term 2
AppendEntries just succeeded
Can’t be elected as leader for term 3

• Safe: leader for term 3 must contain entry 4

leader for term 3 must be in this set
Log Safety Requirement

Once a log, no other node must apply a different value for that log entry

• Raft safety property:
  • If a leader has decided that a log entry is committed, that entry will be present in the logs of all future leaders

• This guarantees the safety requirement
  • Leaders never overwrite entries in their logs
  • Only entries in the leader’s log can be committed
Committing Entry from Earlier Term
• Case #2/2: Leader is trying to finish committing entry from an earlier term
• Entry 3 not safely committed:
  • $s_5$ can be elected as leader for term 5
  • If elected, it will overwrite entry 3 on $s_1$, $s_2$, and $s_3$!

If committing from an earlier term: then never commit, until at least one req. in current term is also committed.
New Commitment Rules

• For a leader to decide an entry is committed:
  • Must be stored on a majority of servers
  • At least one new entry from leader’s term must also be stored on majority of servers

• Once entry 4 committed:
  • s5 cannot be elected leader for term 5
  • Entries 3 and 4 both safe

Combination of election rules and commitment rules makes Raft safe
Safety Changes to Protocol Summarized

- **Voting changes:**
  - **Original:** vote for first candidate to contact you
  - **Modification:** deny vote if the candidate has less complete log
    - Candidate is at a lower term than you
    - Candidate is at same term but lower index

- **Commitment changes:**
  - **Original:** majority of servers reply
  - **New:** Must ensure that at least one entry from current term is committed before prior logs are committed
Log Inconsistencies

Leader changes can result in log inconsistencies:

<table>
<thead>
<tr>
<th>log index leader for term 8</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>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>(e)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f)</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></td>
</tr>
</tbody>
</table>

Possible followers:

- Missing Entries
- Extraneous Entries
Repairing Follower Logs

- New leader must make follower logs consistent with its own
  - Delete extraneous entries
  - Fill in missing entries
- Leader keeps \textbf{nextIndex} for each follower:
  - Index of next log entry to send to that follower
  - Initialized to \((1 + \text{leader’s last index})\)
- When AppendEntries consistency check fails, decrement \textbf{nextIndex} and try again:
Repairing Logs, cont’d

- When follower overwrites inconsistent entry, it deletes all subsequent entries:

```
log index: 1 2 3 4 5 6 7 8 9 10 11
leader for term 7: 1 1 1 4 4 5 5 6 6 6
follower (before): 1 1 1 2 2 2 3 3 3 3 3
follower (after): 1 1 1 4
```
Partitioned Leader

- A client talking to a partitioned leader could be delayed forever.
  - Solution: leader will step down after a number of rounds of heartbeats with no response from majority
Neutralizing Old Leaders

• Deposed leader may not be dead:
  • Temporarily disconnected from network
  • Other servers elect a new leader
  • Old leader becomes reconnected, attempts to commit log entries

• Terms used to detect stale leaders (and candidates)
  • Every RPC contains term of sender
  • Node with lower term steps down (becomes follower)
    • If sender’s term is older, RPC is rejected, sender reverts to follower and updates its term
    • If receiver’s term is older, it reverts to follower, updates its term, then processes RPC normally

• Election updates terms of majority of servers
  • Deposed server cannot commit new log entries
Raft

• Proposed by Ongaro and Ousterhout in 2014

• Five components
  • Leader election
  • Log replication
  • Safety
  • Client protocol
  • Membership changes

• Assumes crash failures (so no byzantine failures)
• No dependency on time for safety
  • But depends on time for availability
• Tolerates \((N-1)/2\) failures
Client Protocol

• **Send commands to leader**
  - If leader unknown, contact any server
  - If contacted server not leader, it will redirect to leader

• **Leader does not respond until command has been logged, committed, and executed by leader’s state machine**

• **If request times out (e.g., leader crash):**
  - Client reissues command to some other server
  - Eventually redirected to new leader
  - Retry request with new leader
Client Protocol, cont’d

• What if leader crashes after executing command, but before responding?
  • Must not execute command twice

**Solution: client embeds a unique id in each command**
  • Server includes id in log entry
  • Before accepting command, leader checks its log for entry with that id
  • If id found in log, ignore new command, return response from old command

• Result: **exactly-once semantics** as long as client doesn’t crash
  • Enforces **linearizability** (will see in upcoming lecture)
What if clients can crash?

- Servers maintain a session for each client
  - Keep track of latest sequence number processed for client, and response

- Generalizes for multiple outstanding requests
  - Server keeps set of \( \text{seq, resp} \) for client
  - Client includes with request lowest seq with no response
  - Server can discard smaller sequence numbers

- Must expire sessions
  - All replicas must agree on when to do this
  - Raft uses leader timestamp, committed to log
Alive clients with expired sessions

• How to distinguish between client which exited from client which just took too long?

• Require clients to register with the leader when starting a session
  • RegisterClient RPC
  • Leader returns unique ID to the client
  • Client uses this ID in subsequent request

• If server receives request for non-existing session...
  • Return an error. Current implementation crashes the client, forcing restart
Raft

• Proposed by Ongaro and Ousterhout in 2014

• Five components
  • Leader election
  • Log replication
  • Safety
  • Client protocol
  • Membership changes

• Assumes crash failures (so no byzantine failures)
• No dependency on time for safety
  • But depends on time for availability
• Tolerates \((N-1)/2\) failures
Configuration Changes

• Cannot switch directly from one configuration to another: conflicting majorities could arise

• Switching from N=3 to N=5

See the paper for details
Consensus Use-Cases
Assumptions!

• Each program is a state machine
  • Deterministic
  • Given initial state + sequence of events
    • Terminates at same state

• Replicated State Machine (RSM)

• Implications of RSM
  • Each server can independently process events
  • AND reach same conclusion
    • ONLY if events are total ordered

Given identical initial states, applying updates in same sequence results in same final state
Server Recovery (SR) with RSM

• On node failure
  • Start new replica
  • Feed it sequence of events

• Interesting questions:
  • How to detect failures?
  • How to determine sequence of events?
Log Replication

MetaData Management (Service Discovery)
<table>
<thead>
<tr>
<th>Project</th>
<th>Consensus System</th>
<th>SR</th>
<th>LR</th>
<th>SS</th>
<th>BO</th>
<th>SD</th>
<th>GM</th>
<th>LE</th>
<th>MM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFS</td>
<td>Chubby</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Borg</td>
<td>Chubby/Paxos</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>etcd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Megastore</td>
<td>Paxos</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Spanner</td>
<td>Paxos</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Bigtable</td>
<td>Chubby</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hadoop/HDFS</td>
<td>ZooKeeper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HBase</td>
<td>ZooKeeper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hive</td>
<td>ZooKeeper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Configurator</td>
<td>Zeus</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Cassandra</td>
<td>ZooKeeper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Accumulo</td>
<td>ZooKeeper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BookKeeper</td>
<td>ZooKeeper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hedwig</td>
<td>ZooKeeper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Kafka</td>
<td>ZooKeeper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Solr</td>
<td>ZooKeeper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Giraph</td>
<td>ZooKeeper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hama</td>
<td>ZooKeeper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mesos</td>
<td>ZooKeeper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CoreOS</td>
<td>etcd</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OpenStack</td>
<td>ZooKeeper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Neo4j</td>
<td>ZooKeeper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- server replication (SR),
- log replication (LR),
- synchronization service (SS),
- barrier orchestration (BO),
- service discovery (SD),
- group membership (GM),
- leader election (LE),
- metadata management (MM)

Redundant servers (for processing)

- access to resources
- across servers
- members of a group

https://www.cse.buffalo.edu/tech-reports/2016-02.pdf
When should you use Raft-style Replication?

Server A

New replica server

Sequence of events

Server Replication?

Log replication

Metadata Management