In Search of an Understandable Consensus Algorithm

Diego Ongaro  John Ousterhout
Stanford University

http://raftconsensus.github.io
What is Consensus?

- Agreement on shared state (single system image)
- Recovers from server failures autonomously
  - Minority of servers fail: no problem
  - Majority fail: lose availability, retain consistency

- Key to building consistent storage systems
Replicated State Machines

- Replicated log $\Rightarrow$ replicated state machine
  - All servers execute same commands in same order
- Consensus module ensures proper log replication
- System makes progress as long as any majority of servers are up
- Failure model: fail-stop (not Byzantine), delayed/lost messages
How Is Consensus Used?

- Top-level system configuration

- Replicate entire database state
Paxos Protocol

- Leslie Lamport, 1989
- Nearly synonymous with consensus

“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
Raft’s Design for Understandability

- We wanted the **best** algorithm for building real systems
  - Must be correct, complete, and perform well
  - Must also be **understandable**
- “What would be easier to understand or explain?”
  - Fundamentally different decomposition than Paxos
  - Less complexity in state space
  - Less mechanism
Raft User Study

Quiz Grades

Survey Results

- Paxos much easier
- Paxos somewhat easier
- Roughly equal
- Raft somewhat easier
- Raft much easier
Raft Overview

1. Leader election
   - Select one of the servers to act as cluster leader
   - Detect crashes, choose new leader

2. Log replication (normal operation)
   - Leader takes commands from clients, appends them to its log
   - Leader replicates its log to other servers (overwriting inconsistencies)

3. Safety
   - Only a server with an up-to-date log can become leader
RaftScope Visualization
Core Raft Review

1. Leader election
   - Heartbeats and timeouts to detect crashes
   - Randomized timeouts to avoid split votes
   - Majority voting to guarantee at most one leader per term

2. Log replication (normal operation)
   - Leader takes commands from clients, appends them to its log
   - Leader replicates its log to other servers (overwriting inconsistencies)
   - Built-in consistency check simplifies how logs may differ

3. Safety
   - Only elect leaders with all committed entries in their logs
   - New leader defers committing entries from prior terms
Randomized Timeouts

- How much randomization is needed to avoid split votes?

- Conservatively, use random range ~10x network latency
<table>
<thead>
<tr>
<th>Implementation</th>
<th>Language</th>
<th>Author/Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>go-raft</td>
<td>Go</td>
<td>Ben Johnson (Sky) and Xiang Li (CoreOS)</td>
</tr>
<tr>
<td>kanaka/raft.js</td>
<td>JS</td>
<td>Joel Martin</td>
</tr>
<tr>
<td>hashicorp/raft</td>
<td>Go</td>
<td>Armon Dadgar (HashiCorp)</td>
</tr>
<tr>
<td>rafter</td>
<td>Erlang</td>
<td>Andrew Stone (Basho)</td>
</tr>
<tr>
<td>ckite</td>
<td>Scala</td>
<td>Pablo Medina</td>
</tr>
<tr>
<td>kontiki</td>
<td>Haskell</td>
<td>Nicolas Trangez</td>
</tr>
<tr>
<td>LogCabin</td>
<td>C++</td>
<td>Diego Ongaro (Stanford)</td>
</tr>
<tr>
<td>akka-raft</td>
<td>Scala</td>
<td>Konrad Malawski</td>
</tr>
<tr>
<td>floss</td>
<td>Ruby</td>
<td>Alexander Flatten</td>
</tr>
<tr>
<td>CRaft</td>
<td>C</td>
<td>Willem-Hendrik Thiart</td>
</tr>
<tr>
<td>barge</td>
<td>Java</td>
<td>Dave Rusek</td>
</tr>
<tr>
<td>harryw/raft</td>
<td>Ruby</td>
<td>Harry Wilkinson</td>
</tr>
<tr>
<td>py-raft</td>
<td>Python</td>
<td>Toby Burress</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

- Consensus widely regarded as difficult
- Raft designed for understandability
  - Easier to teach in classrooms
  - Better foundation for building practical systems
- Paper/thesis covers much more
  - Cluster membership changes (simpler in thesis)
  - Log compaction (expanded tech report/thesis)
  - Client interaction (expanded tech report/thesis)
  - Evaluation (thesis)
Questions

raftconsensus.github.io