THE RAFT CONSENSUS ALGORITHM

DIEGO ONGARO AND JOHN OUSTERHOUT

STANFORD UNIVERSITY

APRIL 2015

raftconsensus.github.io
MOTIVATION

• Goal: shared hash table (state machine)
• Put it on a single machine attached to network
  ■ Pros: easy, consistent
  ■ Cons: prone to failure
• With Raft, keep consistency yet deal with failures
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 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 up
• Failure model: fail-stop (not Byzantine), delayed/lost msgs
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 an algorithm optimized 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

![Graph showing Raft grade vs Paxos grade.](image)

- Raft then Paxos: +
- Paxos then Raft: 

![Bar chart showing number of participants.](image)

- Paxos much easier: Red
- Paxos somewhat easier: Black
- Roughly equal: White
- Raft somewhat easier: Blue
- Raft much easier: Blue

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 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 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
## RAFT IMPLEMENTATIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Primary Authors</th>
<th>Language</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>etcd/raft</td>
<td>Blake Mizerany, Xiang Li and Yicheng Qin (CoreOS)</td>
<td>Go</td>
<td>Apache 2.0</td>
</tr>
<tr>
<td>go-raft</td>
<td>Ben Johnson (Sky) and Xiang Li (CMU, CoreOS)</td>
<td>Go</td>
<td>MIT</td>
</tr>
<tr>
<td>hashicorp/raft</td>
<td>Armon Dadgar (hashicorp)</td>
<td>Go</td>
<td>MPL-2.0</td>
</tr>
<tr>
<td>copycat</td>
<td>Jordan Halterman</td>
<td>Java</td>
<td>Apache2</td>
</tr>
<tr>
<td>LogCabin</td>
<td>Diego Ongaro (Stanford, Scale Computing)</td>
<td>C++</td>
<td>ISC</td>
</tr>
<tr>
<td>akka-raft</td>
<td>Konrad Malawski</td>
<td>Scala</td>
<td>Apache2</td>
</tr>
<tr>
<td>kanaka/raft.js</td>
<td>Joel Martin</td>
<td>Javascript</td>
<td>MPL-2.0</td>
</tr>
<tr>
<td>rafter</td>
<td>Andrew Stone (Basho)</td>
<td>Erlang</td>
<td>Apache2</td>
</tr>
<tr>
<td>OpenDaylight</td>
<td>Moiz Raja, Kamal Rameshan, Robert Varga (Cisco), Tom Pantelis (Brocade)</td>
<td>Java</td>
<td>Eclipse</td>
</tr>
<tr>
<td>liferaft</td>
<td>Arnout Kazemier</td>
<td>Javascript</td>
<td>MIT</td>
</tr>
<tr>
<td>skiff</td>
<td>Pedro Teixeira</td>
<td>Javascript</td>
<td>ISC</td>
</tr>
<tr>
<td>ckite</td>
<td>Pablo Medina</td>
<td>Scala</td>
<td>Apache2</td>
</tr>
<tr>
<td>willemt/raft</td>
<td>Willem-Hendrik Thiart</td>
<td>C</td>
<td>BSD</td>
</tr>
</tbody>
</table>

Copied from Raft website, probably stale.
LOGCABIN

- C++, permissive license
- Started as research platform for Raft at Stanford
- Working with Scale Computing to make production-ready
- 1.0 release later this month
  - Rolling upgrades for 1.0
- Looking for more users!

github.com/logcabin
CONCLUSION

• Consensus widely regarded as difficult
• Raft designed for understandability
  ▪ Easier to teach in classrooms
  ▪ Better foundation for building practical systems
• Pieces needed for a practical system:
  ▪ Cluster membership changes (simpler in dissertation)
  ▪ Log compaction (expanded tech report/dissertation)
  ▪ Client interaction (expanded tech report/dissertation)
  ▪ Evaluation (dissertation: understandability, correctness, leader election & replication performance)
QUESTIONS

raftconsensus.github.io

raft-dev mailing list