An Introduction to Consensus with Raft

Diego Ongaro  John Ousterhout
Stanford University

http://raftconsensus.github.io
Distributed Systems

availability or consistency
Inside a Consistent System

- **TODO:** eliminate single point of failure
- An ad hoc algorithm
  - “This case is rare and typically occurs as a result of a network partition with replication lag.”
  - Watch out for @aphyr
    - OR -
- A consensus algorithm (built-in or library)
  - Paxos, Raft, …
- A consensus service
  - ZooKeeper, etcd, consul, …
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

Servers
Why Is Consensus Needed?

- Key to building consistent storage systems
- Top-level system configuration
  - Which server is my SQL master?
  - What shards exist in my storage system?
  - Which servers store shard X?
- Sometimes used to replicate entire database state (e.g., Megastore, Spanner)
Goal: Replicated Log

- 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
Paxos Protocol

- Leslie Lamport, 1989
- Nearly synonymous with consensus
- Hard to understand
  
  “The dirty little secret of the NSDI community is that at most five people really, truly understand every part of Paxos ;-).” – Anonymous NSDI reviewer

- Bad foundation for building systems
  
  “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
User study

Quiz Grades

Survey Results
Raft Overview

1. Leader election
   - Select one of the servers to act as 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 elect leaders with all committed entries in their logs
Server States

- At any given time, each server is either:
  - **Follower**: completely passive replica (issues no RPCs, responds to incoming RPCs)
  - **Candidate**: used to elect a new leader
  - **Leader**: handles all client interactions, log replication
    - At most one viable leader at a time
Terms

- **Time divided into terms:**
  - Election
  - Normal operation under a single leader
- **At most one leader per term**
- **Each server maintains current term value**
- **Key role of terms:** identify obsolete information
Leader Election

Leaders send **heartbeats** to maintain authority.

Upon **election timeout**, start new election:

- Increment current term
- Change to Candidate state
- Vote for self

- Send **Request Vote** RPCs to all other servers, wait until either:
  1. Receive votes from majority of servers:
     - Become leader, send heartbeats to all other servers
  2. Receive RPC from valid leader:
     - Return to follower state
  3. No-one wins election (election timeout elapses):
     - Increment term, start new election
Leader Election Visualization

- The Secret Lives of Data
  [http://thesecretlivesofdata.com](http://thesecretlivesofdata.com)
- Visualizes distributed algorithms, starting with Raft
- Project by Ben Johnson (author of go-raft)
Randomized Timeouts

- If we choose election timeouts randomly,

- One server usually times out and wins election before others wake up
Raft Paper

- Log replication
- Client interaction
- Cluster membership changes
- Log compaction

  - June 19-20 in Philadelphia
  - Draft on Raft website
# Raft Implementations

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Language</th>
<th>Author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>kanaka/raft.js</td>
<td>JS</td>
<td>Joel Martin</td>
</tr>
<tr>
<td>go-raft</td>
<td>Go</td>
<td>Ben Johnson (Sky) and Xiang Li (CoreOS)</td>
</tr>
<tr>
<td>hashicorp/raft</td>
<td>Go</td>
<td>Armon Dadgar (HashiCorp)</td>
</tr>
<tr>
<td>LogCabin</td>
<td>C++</td>
<td>Diego Ongaro (Stanford)</td>
</tr>
<tr>
<td>ckite</td>
<td>Scala</td>
<td>Pablo Medina</td>
</tr>
<tr>
<td>peterbourgon/raft</td>
<td>Go</td>
<td>Peter Bourgon</td>
</tr>
<tr>
<td>rafter</td>
<td>Erlang</td>
<td>Andrew Stone (Basho)</td>
</tr>
<tr>
<td>barge</td>
<td>Java</td>
<td>Dave Rusek</td>
</tr>
<tr>
<td>py-raft</td>
<td>Python</td>
<td>Toby Burress</td>
</tr>
<tr>
<td>ocaml-raft</td>
<td>OCaml</td>
<td>Heidi Howard (Cambridge)</td>
</tr>
</tbody>
</table>

...
Best Logo: go-raft

by Brandon Philips (CoreOS)
Summary

- Consensus is key to building consistent systems
- Design for understandability
- Raft separates leader election from log replication
  - Leader election uses voting and randomized timeouts


- Paper draft, other talks
- 10 to 50+ implementations
- raft-dev mailing list

Diego Ongaro  @ongardie