



# The Raft Consensus Algorithm

Diego Ongaro and John Ousterhout

November 2015



Source code available at <https://github.com/ongardie/raft-talk>.

Unless otherwise noted, this work is:

© 2012-2015 Diego Ongaro, © 2012-2014 John Ousterhout.

Licensed under the [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/).

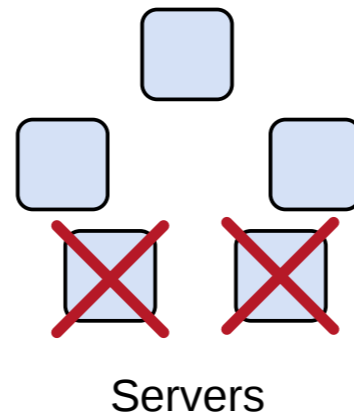


# Motivation

- Goal: shared key-value store (state machine)
- Host 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

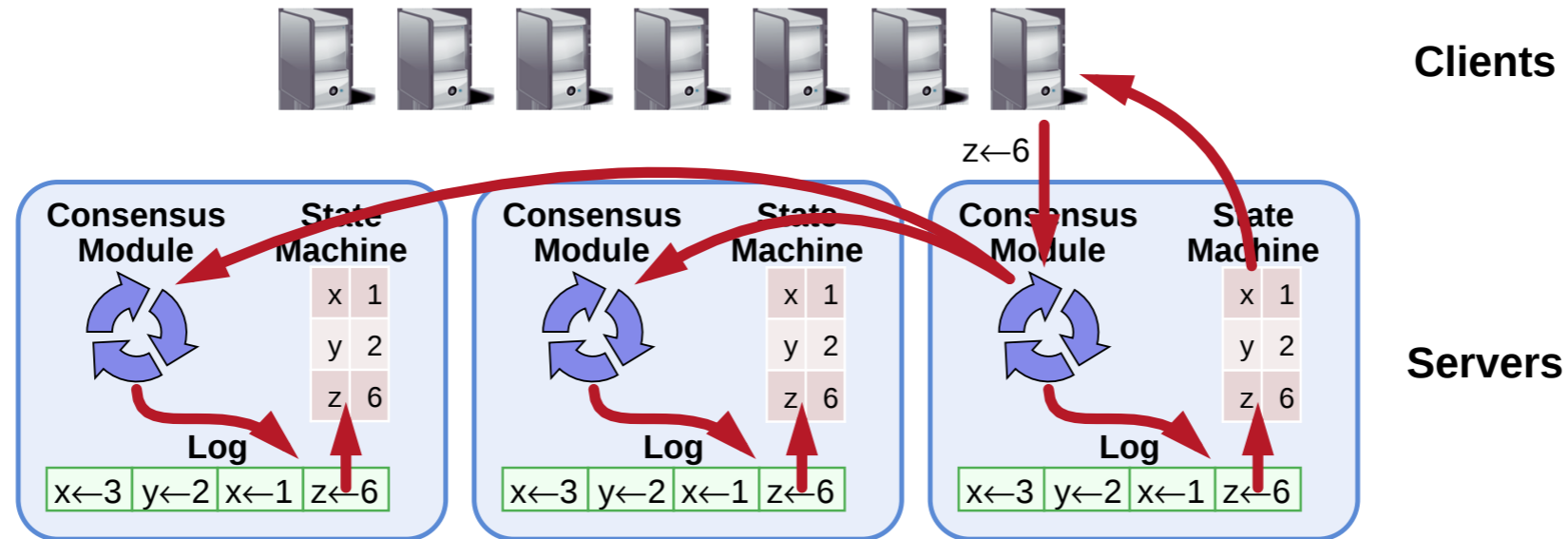
- Agreement on shared state (single system image)
- Recovers from server failures autonomously
  - Minority of servers fail: no problem
  - Majority fail: lose availability, retain consistency



- Key to building consistent storage systems

# Replicated State Machines

Typical architecture for consensus 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

# 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 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

or <https://raft.github.io/raftscope-replay/>



# 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

# 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

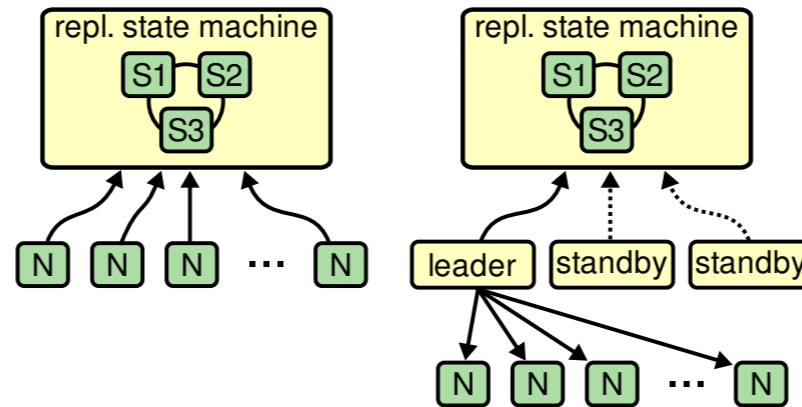
[raft.github.io](https://raft.github.io)

[raft-dev mailing list](#)

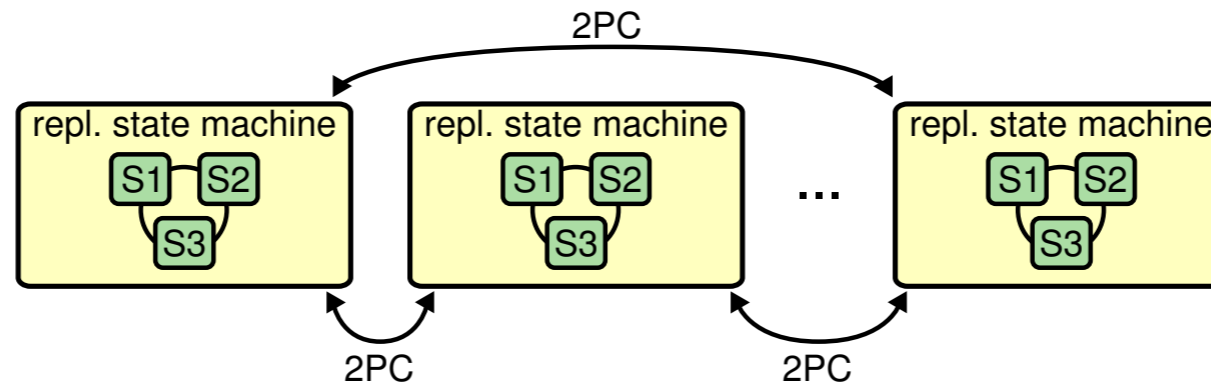


# How Is Consensus Used?

Top-level system configuration



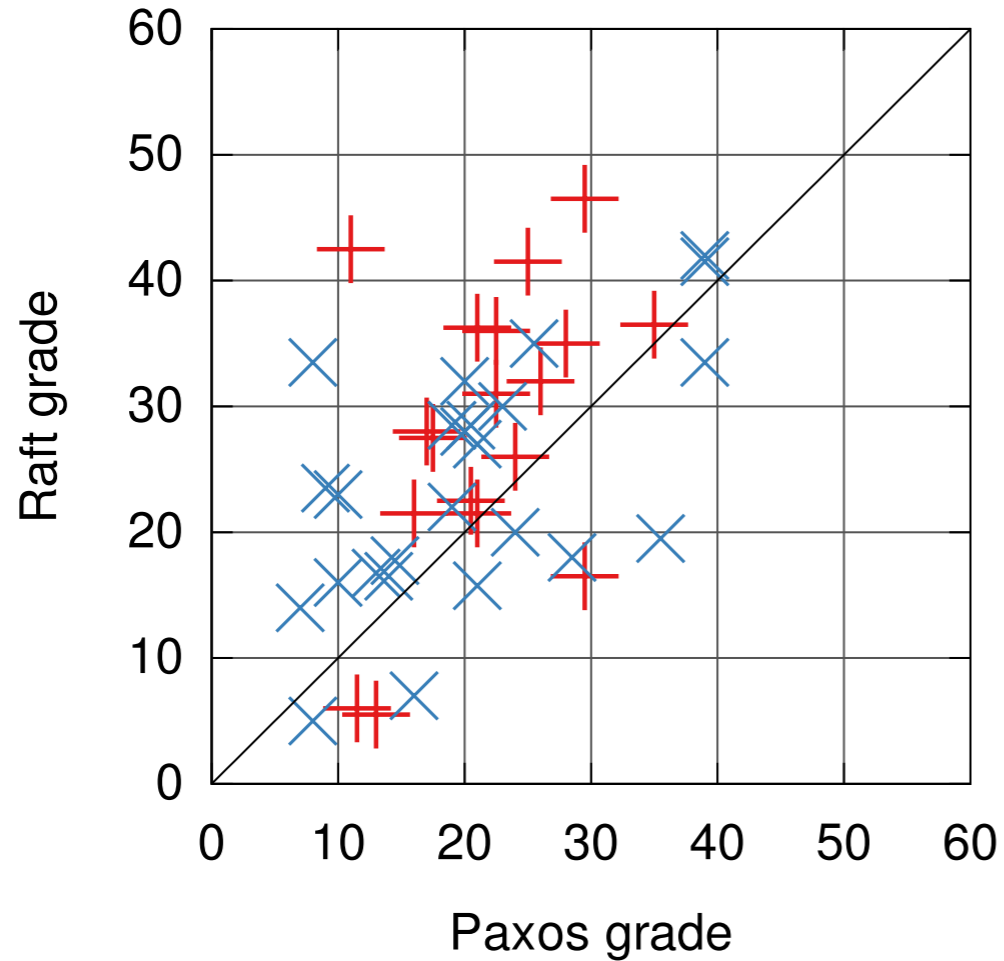
Replicate entire database state



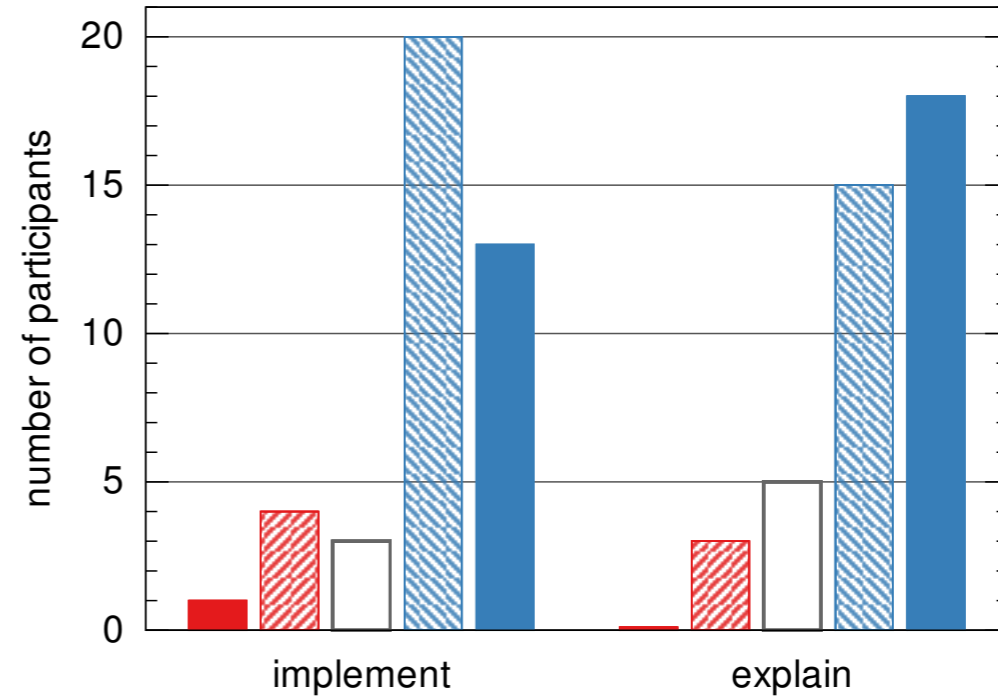
# Raft

- Algorithm for implementing a replicated log
- System makes progress as long as any majority of servers up
- Failure model: fail-stop (not Byzantine), delayed/lost msgs
- Designed for [understandability](#)

# Raft User Study



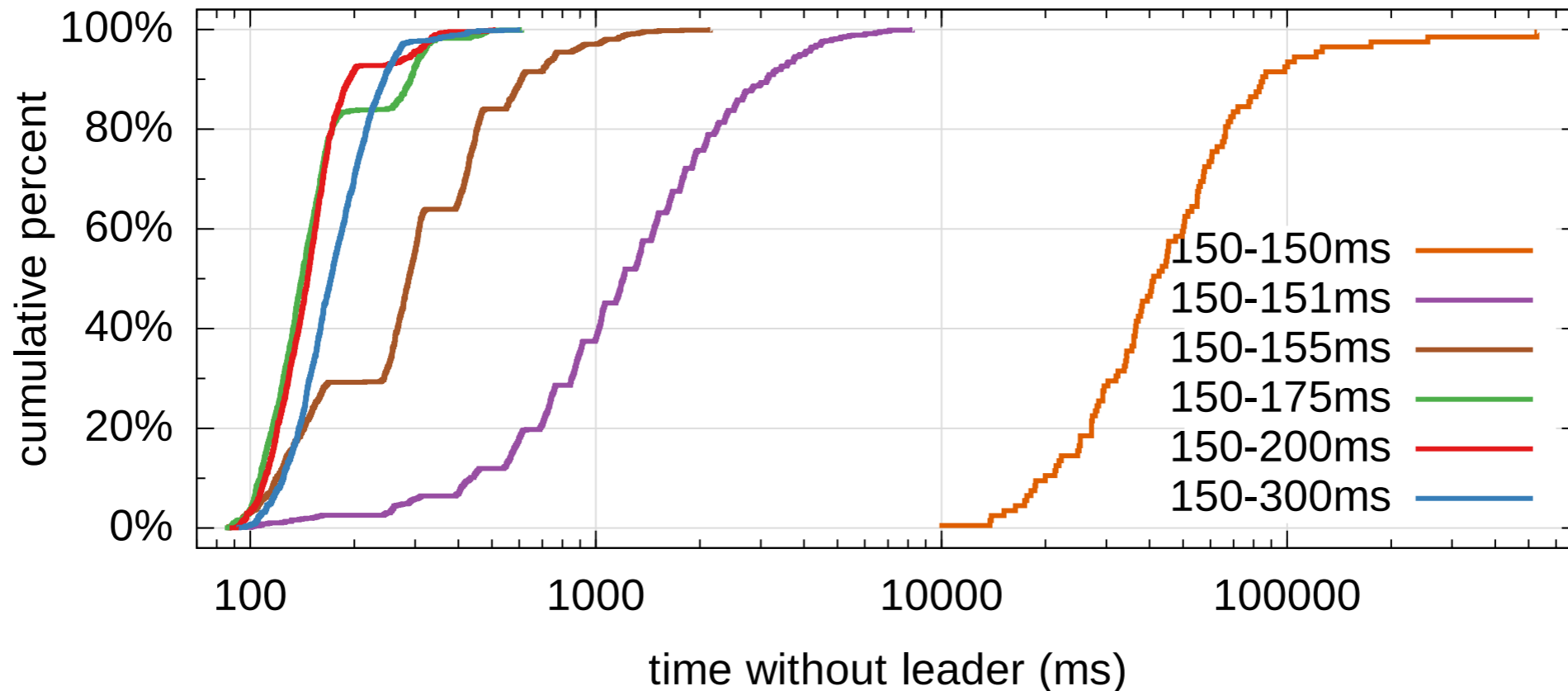
Raft then Paxos +  
Paxos then Raft X



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

# Randomized Timeouts

- How much randomization is needed to avoid split votes?



- Conservatively, use random range  $\sim 10x$  network latency

# Raft Implementations



Name	Primary Authors	Language	License
<a href="#">RethinkDB/clustering</a>		C++	AGPL
<a href="#">etcd/raft</a>	Blake Mizerany, Xiang Li and Yicheng Qin	Go	Apache 2.0
<a href="#">LogCabin</a>	<a href="#">Diego Ongaro</a> (Stanford)	C++	ISC
<a href="#">go-raft</a>	<a href="#">Ben Johnson</a> (Sky) and <a href="#">Xiang Li</a> (CMU, CoreOS)	Go	MIT
<a href="#">hashicorp/raft</a>	<a href="#">Armon Dadgar</a> (hashicorp)	Go	MPL-2.0
<a href="#">hoverbear/raft</a>	<a href="#">Andrew Hobden</a> , Dan Burkert	Rust	MIT
<a href="#">ckite</a>	<a href="#">Pablo Medina</a>	Scala	Apache2
<a href="#">verdi/raft</a>	James Wilcox, Doug Woos, Pavel Panchekha, Zach Tatlock, Xi Wang, Mike Ernst, and Tom Anderson (University of Washington)	Coq	BSD
<a href="#">OpenDaylight</a>	Moiz Raja, Kamal Rameshan, Robert Varga (Cisco), Tom Pantelis (Brocade)	Java	Eclipse
<a href="#">zraft_lib</a>	Gunin Alexander	Erlang	Apache2
<a href="#">kanaka/raft.js</a>	<a href="#">Joel Martin</a>	Javascript	MPL-2.0
<a href="#">akka-raft</a>	<a href="#">Konrad Malawski</a>	Scala	Apache2
<a href="#">rafter</a>	<a href="#">Andrew Stone</a> (Basho)	Erlang	Apache2
<a href="#">floss</a>	<a href="#">Alexander Flatter</a>	Ruby	MIT
<a href="#">willemt/raft</a>	<a href="#">Willem-Hendrik Thiart</a>	C	BSD
...	...	...	...

Copied from Raft website, probably stale.

# LogCabin

- Started as research platform for Raft at Stanford
- Developed into production system at Scale Computing
- Network service running Raft replicated state machine
- Data model: hierarchical key-value store, kept in memory
- Written in C++ ([gcc 4.4's C++0x](#))

