Consensus:
Bridging Theory and Practice

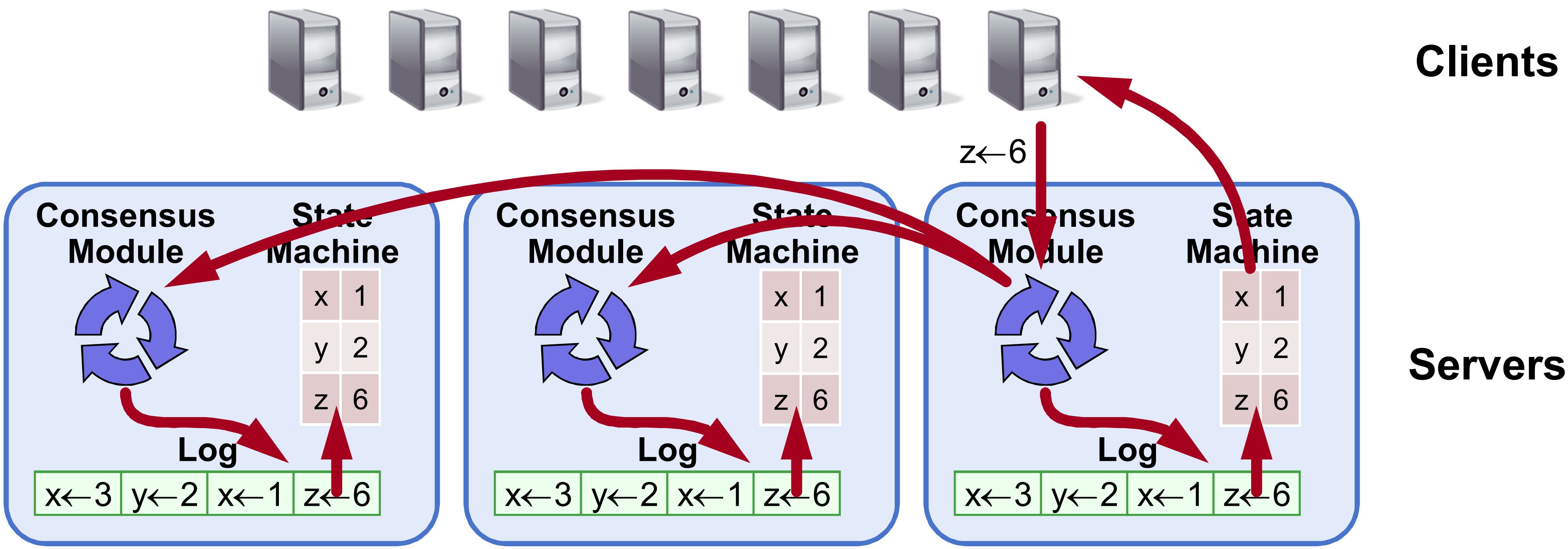
Diego Ongaro

PhD Defense

Introduction

- **Consensus: agreement on shared state**
  - Store state consistently on several servers
  - Must be available even if some servers fail
- **Needed for consistent, fault-tolerant storage systems**
  - Top-level system configuration
  - Sometimes used to replicate entire database state
- **Consensus is widely regarded as difficult**
- **Raft: consensus algorithm designed for understandability**

Replicated State Machines



- **Replicated log ⇒ 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**

Motivation: Paxos

- *"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

Motivation: Paxos (2)

- **Leslie Lamport, 1989**
- **Theoretical foundations**
- **Hard to understand:**
  - Can’t separate phase 1 and 2, no intuitive meanings
- **Bad problem decomposition for building systems**
  - Too low-level
  - Implementations must extend published algorithm

Contributions

Understandability

1. Raft algorithm, designed for understandability
   - Strong form of leadership
   - Leader election algorithm using randomized timeouts
2. User study to evaluate understandability

Completeness

3. Proof of safety and formal spec for core algorithm
4. Cluster membership change algorithm
5. Other components needed for complete and practical system

Design for Understandability

- **Key considerations**
  - How hard is it to explain each alternative?
  - How easy will it be for someone to completely understand the approach and its implications?
- **General techniques**
  - Decomposing the problem
  - Reducing state space complexity

Raft Components

1. **Leader election**
   - Select one of the servers to act as cluster leader
2. **Log replication (normal operation)**
   - Leader takes commands from clients, appends them to its log
   - Leader replicates its log to other servers
3. **Safety**
   - Tie above components together to maintain consistency

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

Topics for Practical Systems

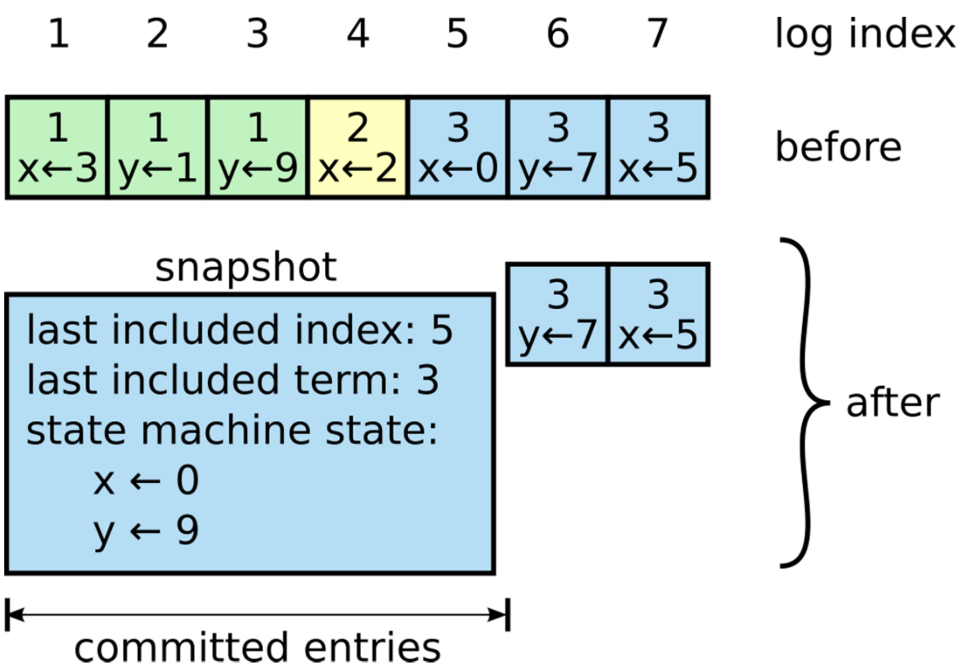
1. **Cluster membership changes**
2. **Log compaction**
3. **Client interaction**

Cluster Membership Changes

- **Grow/shrink cluster, replace nodes**
- **Agreement on change requires consensus**
- **Raft's approach**
  1. Switch to *joint configuration*: requires majorities from both old and new clusters
  2. Switch to new cluster
- **Overlapping majorities guarantee safety**
- **Continues processing requests during change**

Other Topics for Complete Systems

- **Log compaction: snapshotting**



- **Client interaction**
  - How clients find the leader
  - Optimizing read-only operations

Evaluation

1. **Understandability**
   - Is Raft easier to understand?
2. **Leader election performance**
   - How quickly does the randomized timeout approach elect a leader?
3. **Correctness**
   - Formal specification in TLA+
   - Proof of core algorithm's safety
4. **Log replication performance**
   - One round of RPC from leader to commit log entry (same as Multi-Paxos, ZooKeeper)

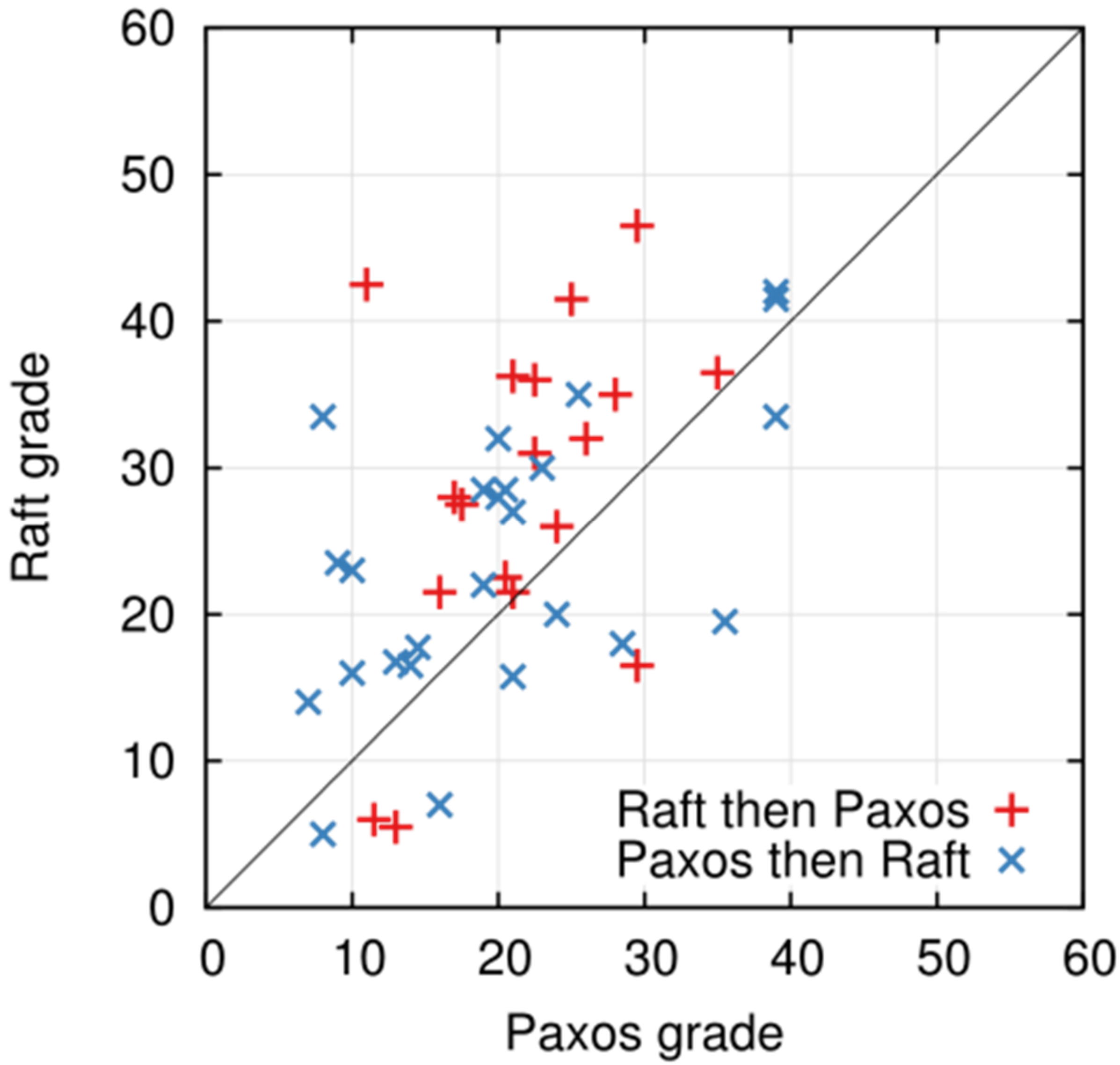
User Study Intro

- **Goal: evaluate Raft's understandability quantitatively**
- **Two classrooms of students**
- **Taught them both Raft and Paxos**
- **Quizzed them to see which one they learned better**
- **Each student:**

1. Raft lecture and quiz	1. Paxos lecture and quiz
2. Paxos lecture and quiz	2. Raft lecture and quiz
3. Short survey	3. Short survey

- **Considered programming assignment: less data**

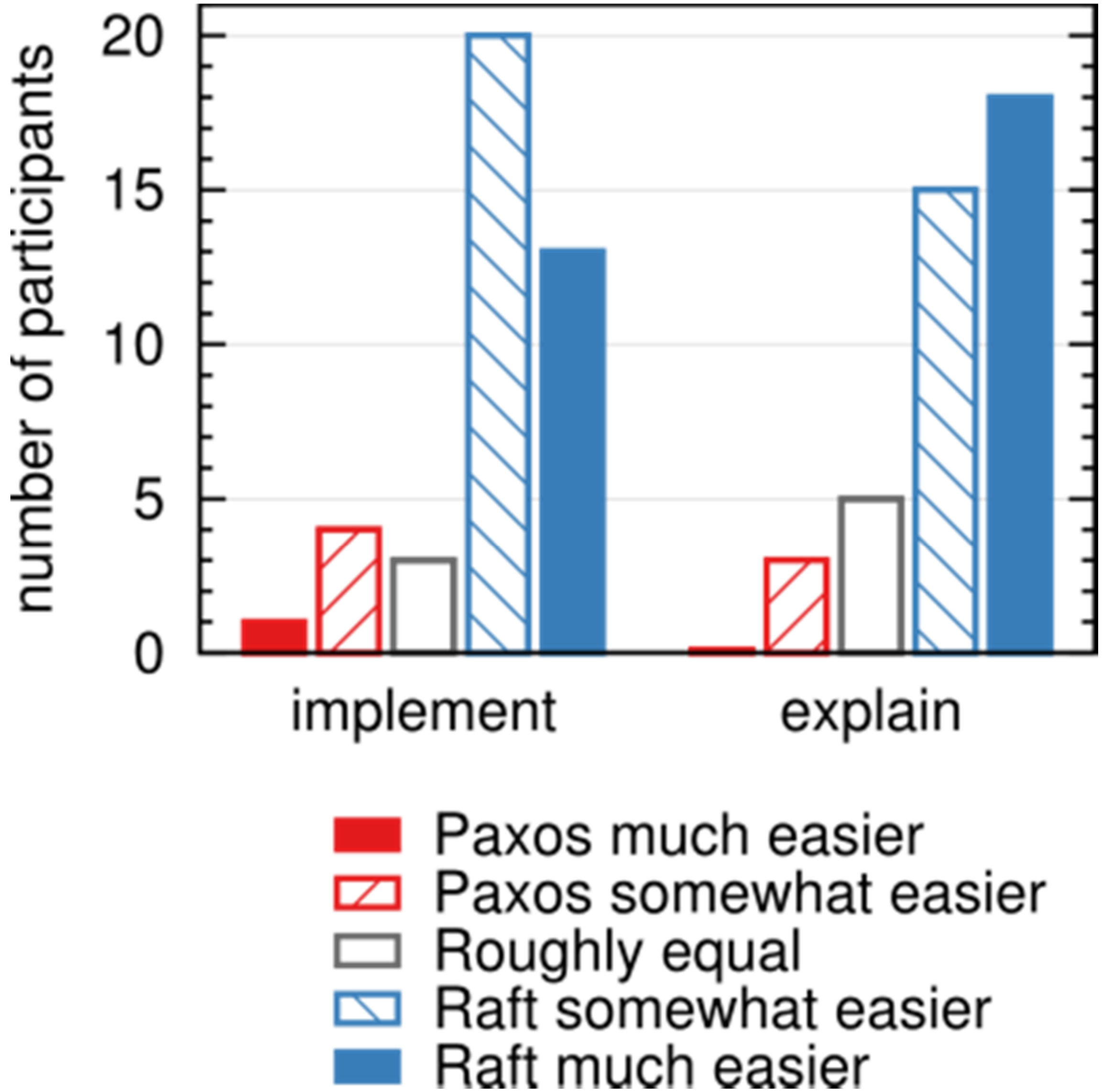
Quiz Results



- **43 participants**
- **33 scored higher on Raft**
- **15 had some prior Paxos experience**
- **Paxos mean 20.8**
- **Raft mean 25.7 (+23.6%)**

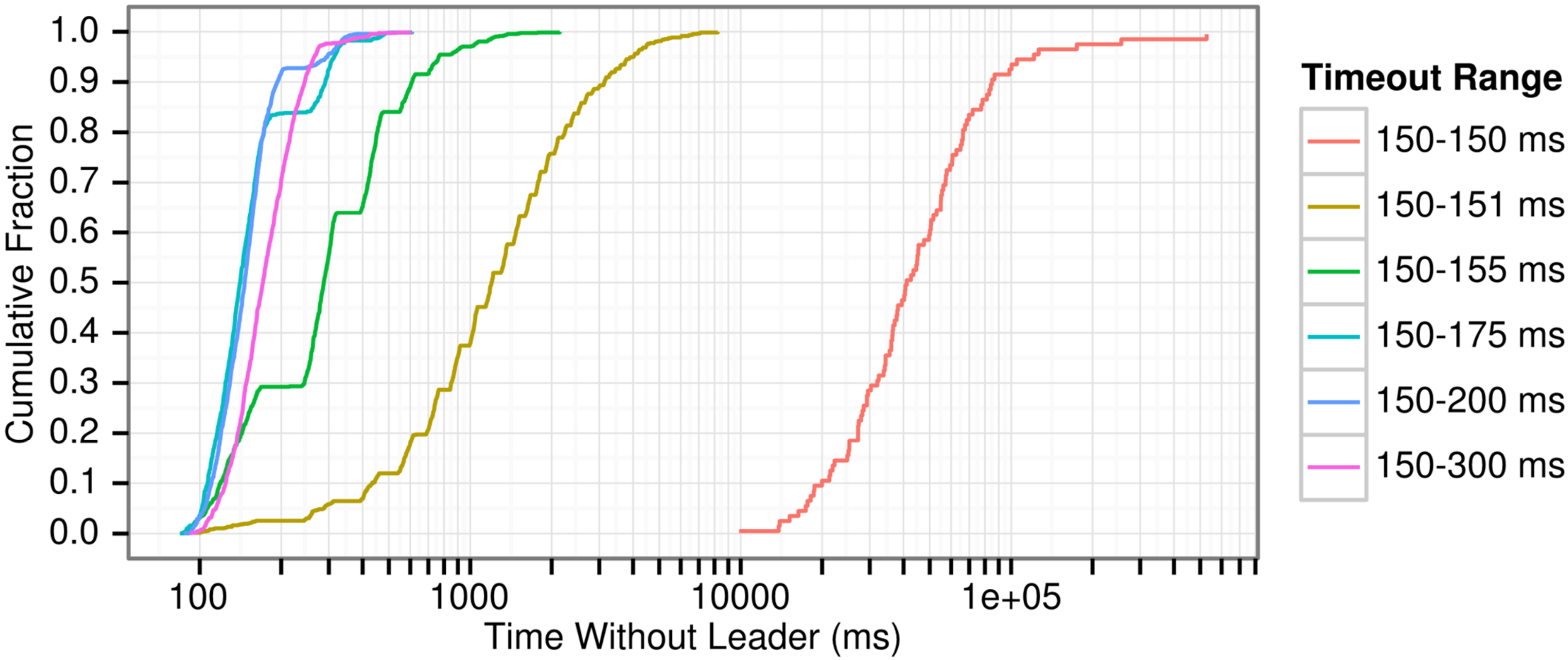
Survey Results

- Which would be easier to implement in a correct and efficient system?
- Which would be easier to explain to a CS grad student?
- For each question, 33 of 41 said Raft



Randomized Timeouts

- **How much randomization is needed to avoid split votes?**



- **Conservatively, use random range ~10x network latency**

Raft Implementations

go-raft	Go	Ben Johnson (Sky) and Xiang Li (CoreOS)
kanaka/raft.js	JS	Joel Martin
hashicorp/raft	Go	Armon Dadgar (HashiCorp)
rafter	Erlang	Andrew Stone (Basho)
ckite	Scala	Pablo Medina
kontiki	Haskell	Nicolas Trangez
LogCabin	C++	Diego Ongaro (Stanford)
akka-raft	Scala	Konrad Malawski
floss	Ruby	Alexander Flatten
CRaft	C	Willem-Hendrik Thiart
barge	Java	Dave Rusek
harryw/raft	Ruby	Harry Wilkinson
py-raft	Python	Toby Burress
…		

Raft Implementations

go-raft	Go	Ben Johnson (Sky) and Xiang Li (CoreOS)
kanaka/raft.js	JS	Joel Martin
hashicorp/raft	Go	Armon Dadgar (HashiC
rafter	Erlang	Andrew Stone (Basho)
ckite	Scala	Pablo Medina
kontiki	Haskell	Nicolas Trangez
LogCabin	C++	Diego Ongaro (Stanfor
akka-raft	Scala	Konrad Malawski
floss	Ruby	Alexander Flatten
CRaft	C	Willem-Hendrik Thiart
barge	Java	Dave Rusek
harryw/raft	Ruby	Harry Wilkinson
py-raft	Python	Toby Burress
	...	

go-raft logo by Brandon Philips

Related Work

- **Paxos**
  - Theoretical, difficult to apply
  - "*Our Paxos implementation is actually closer to the Raft algorithm than to what you read in the Paxos paper...*"
    – Sebastian Kanthak, Spanner
- **Viewstamped Replication, ZooKeeper**
  - Both leader-based
  - *Ad hoc* in nature: did not fully explore design space
  - More complex state spaces: more mechanism
    - Each uses 10 message types, Raft has 4
  - ZooKeeper widely deployed but neither widely implemented

Summary: Contributions

Understandability

1. Raft algorithm, designed for understandability
   - Strong form of leadership
   - Leader election algorithm using randomized timeouts
2. User study to evaluate understandability

Completeness

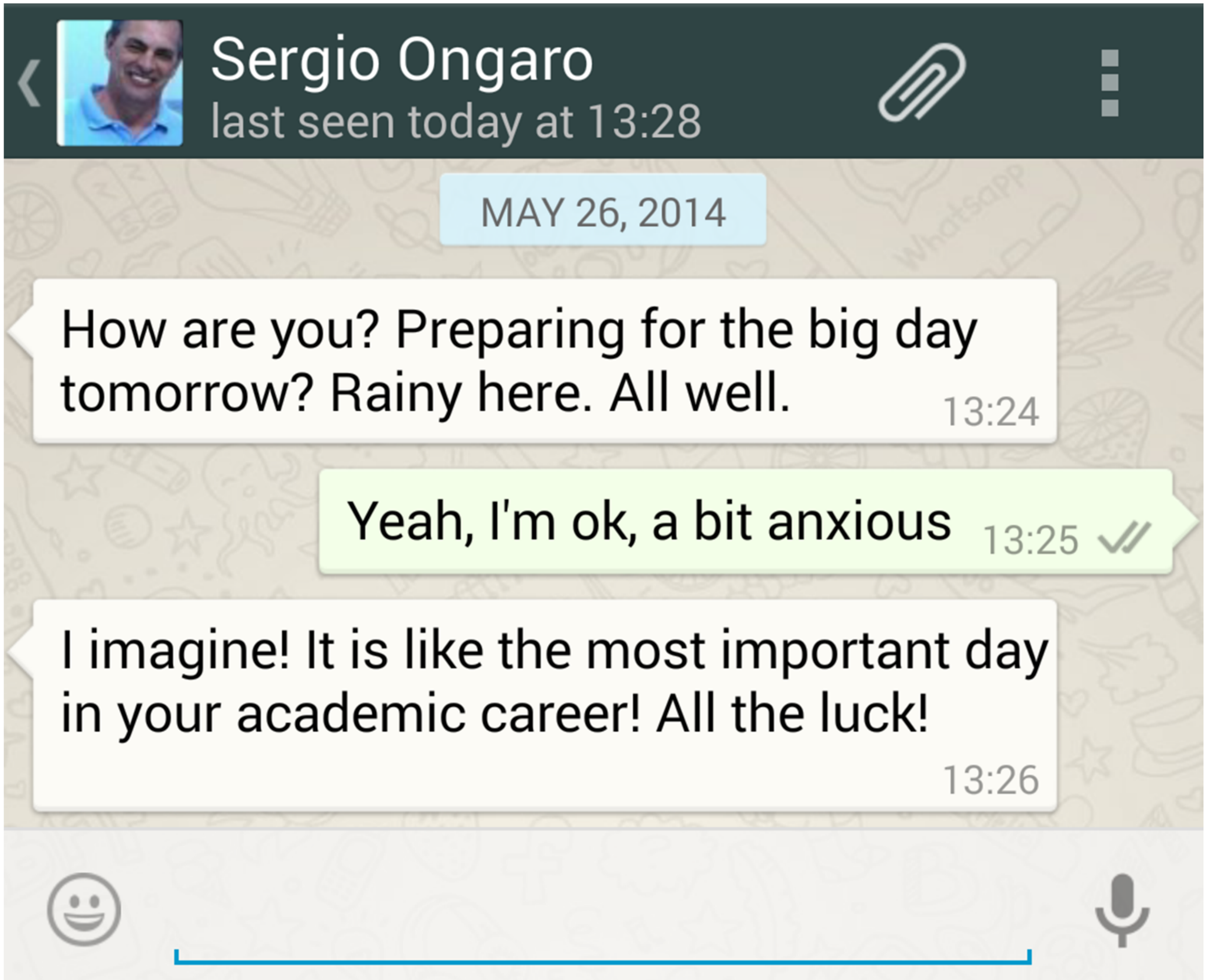
3. Proof of safety and formal spec for core algorithm
4. Cluster membership change algorithm
5. Other components needed for complete and practical system

Conclusions

- **Consensus widely regarded as difficult**
- **Hope Raft makes consensus more accessible**
  - Easier to teach in classrooms
  - Better foundation for building practical systems
- **Burst of Raft-based systems is exciting**
  - Renewed interest in building consensus systems
  - More off-the-shelf options becoming available
- **Understandability should be a primary design goal**

Acknowledgements

Acknowledgements



Questions

raftconsensus.github.io