Raft
Understandable Consensus Algorithm

Presentation for Seminar

Muayyad Saleh Alsadi
(20158043)

Based on Raft Paper
“In Search of an Understandable Consensus Algorithm”
Diego Ongaro and John Ousterhout, Stanford University

Introduction

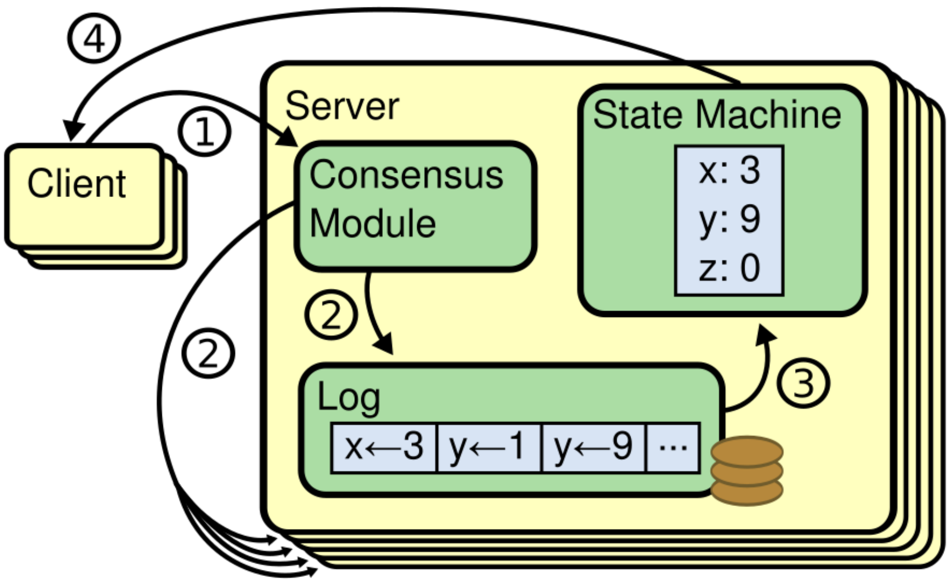
Raft is a consensus algorithm for managing a replicated log. It produces a result equivalent to multi-Paxos, and it is as efficient as Paxos but with completely different structure meant to be understandable as the most key objective.

What is a consensus algorithm?

"Consensus algorithms allow a collection of machines to work as a coherent group that can survive the failures of some of its members. Because of this, they play a key role in building reliable large-scale software systems."

Replicated State Machine

a collection of servers compute identical copies of the same state and can continue operating even if some of the servers are down. Replicated state machines are used to solve a variety of fault tolerance problems in distributed systems.



Why not Paxos?

“ 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”

Quoted from Google's Chubby implementers

Why not Paxos?

#1 Single-decree Paxos is dense and subtle: it is divided into two stages that do not have simple intuitive explanations and cannot be understood independently. Because of this, it is difficult to develop intuitions about why the single-decree protocol works. The composition rules for multi-Paxos add significant additional complexity and subtlety.

Why not Paxos?

#2 it does not provide a good foundation for building practical implementations. One reason is that there is no widely agreed-upon algorithm for multi-Paxos. Lamport’s descriptions are mostly about single-decree Paxos; he sketched possible approaches to multi-Paxos, but many details are missing.

Raft
Properties and key features

Novel features of Raft:

- **Strong leader:** Log entries only flow from the leader to other servers. This simplifies the management of the replicated log and makes Raft easier to understand.
- **Leader election:** beside heartbeats required in any consensus algorithm, Raft uses randomized timers to elect leaders. This helps resolving conflicts simply and rapidly.
- **Membership changes:** changing the set of servers in the cluster uses a new joint consensus approach where the majorities of two different configurations overlap during transitions. This allows the cluster to continue operating normally during configuration changes.

Raft
Properties and key features

consensus algorithm properties

- **Safety:** never returning an incorrect result in any condition despite network delays, partitions, and packet loss, duplication, and reorder
- **Fault-tolerant:** the system would be available and fully-functional in case of failure of some nodes.
- **Does not depend on time consistency:**
- **Performance** is not affected by minority of slow nodes

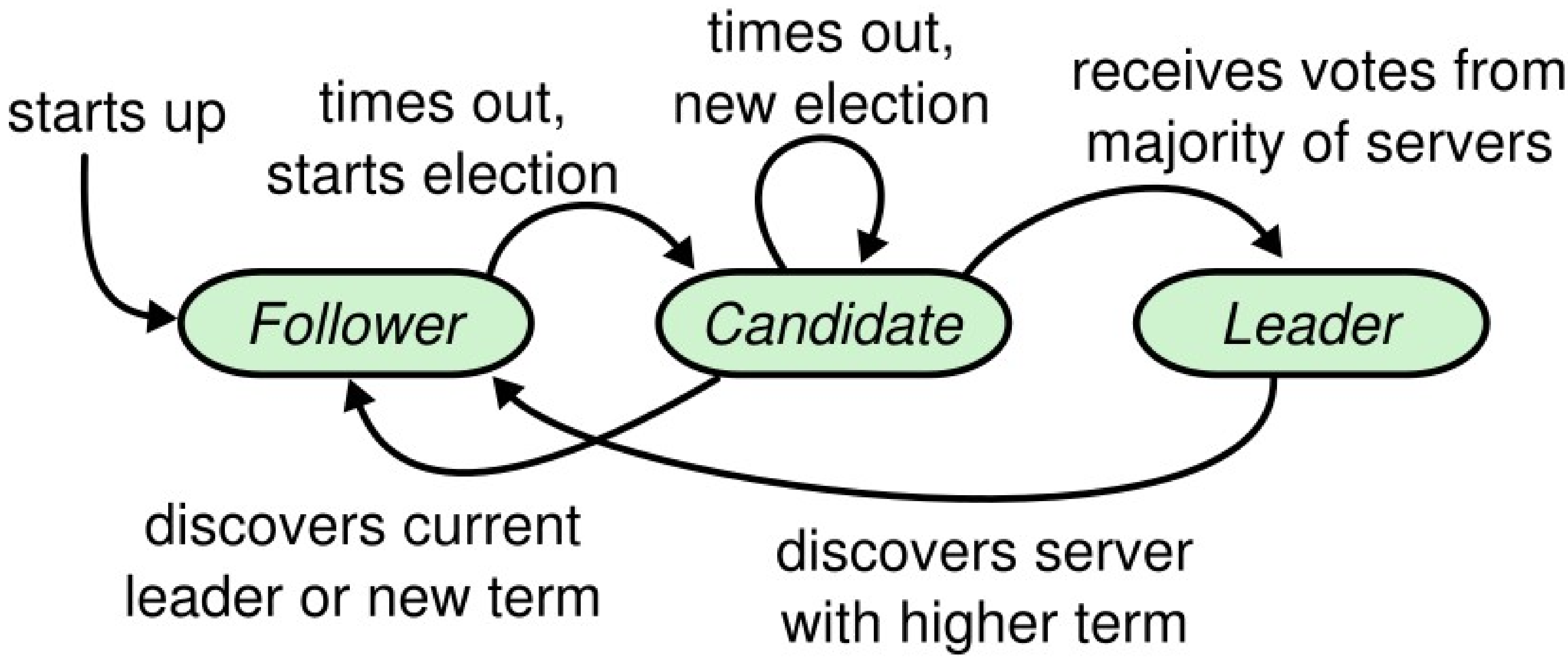
Raft
Sub-problems

- **Leader election:** system should continue to work even if its leader fails, by electing a new leader
- **Log replication:** leader takes operations from clients and replicate them into all nodes. Forcing them all to agree.
- **Safety:** assert that the 5 guarantees (next slide) are all satisfied at any time

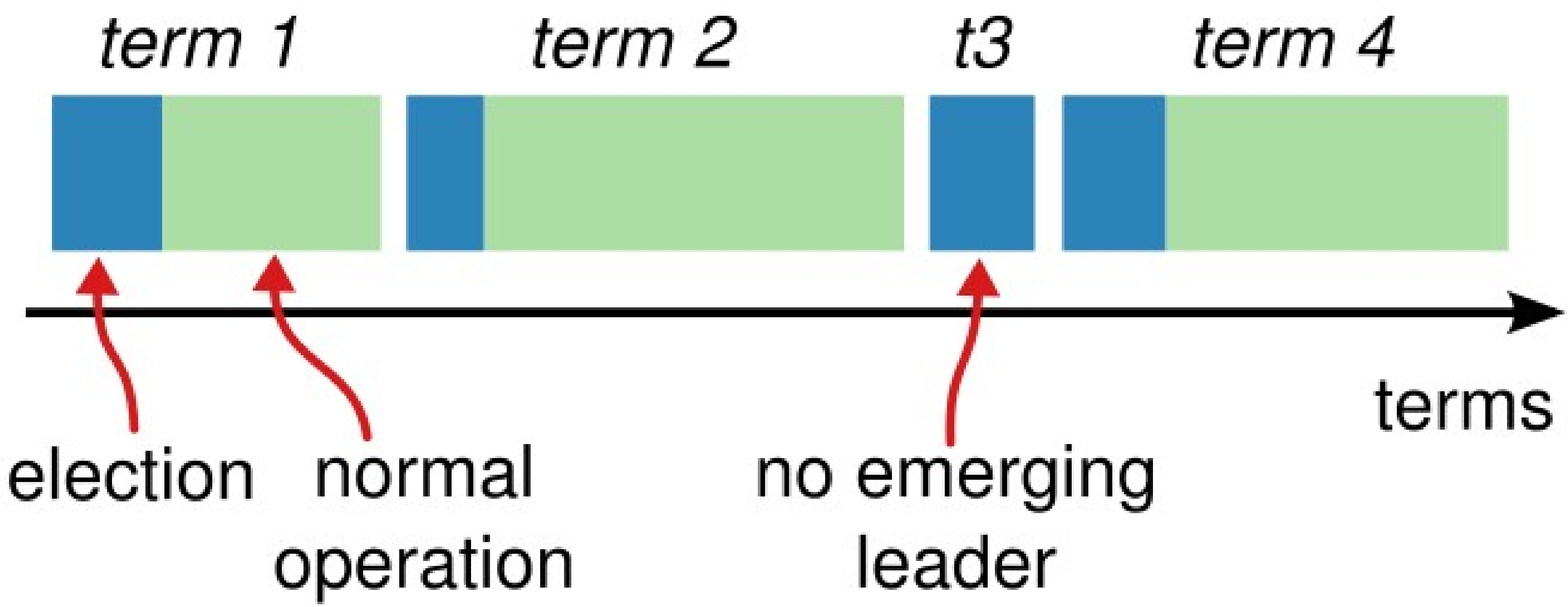
Raft
The 5 guarantees

1) **Election Safety:** at most one leader can be elected in a given term.
2) **Leader Append-Only:** a leader never overwrites or deletes entries in its log; it only appends new entries.
3) **Log Matching:** if two logs contain an entry with the same index and term, then the logs are identical in all entries up through the given index.
4) **Leader Completeness:** if a log entry is committed in a given term, then that entry will be present in the logs of the leaders for all higher-numbered terms.
5) **State Machine Safety:** if a server has applied a log entry at a given index to its state machine, no other server will ever apply a different log entry for the same index.

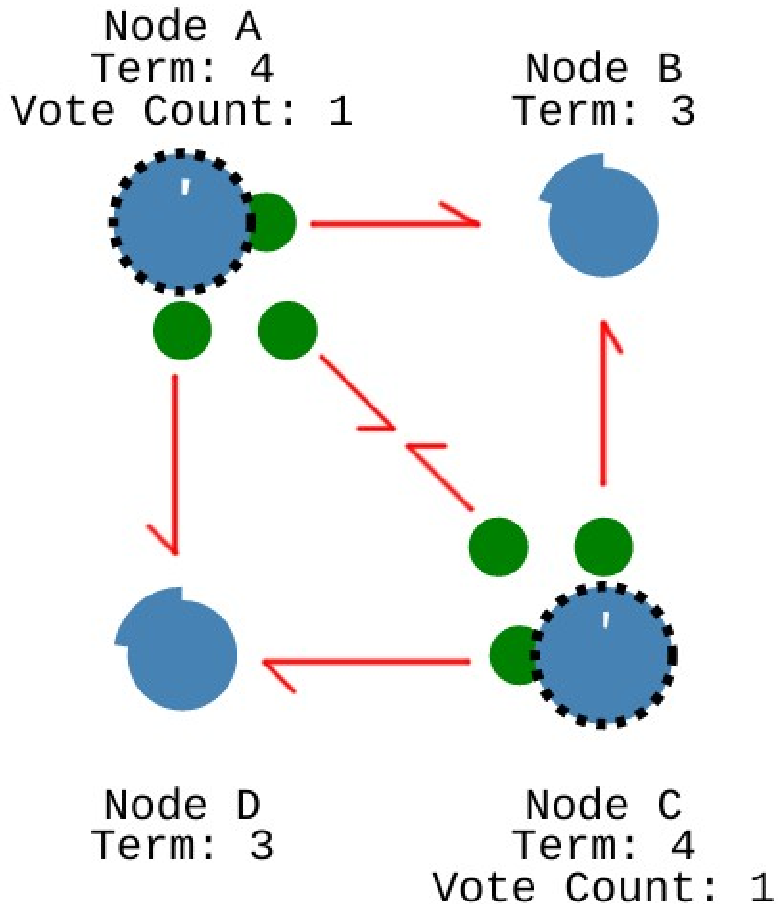
Raft
The Basics: The 3 node states



Raft
The Basics: Terms



Raft
The Basics: no emerging leader term



Raft
The Basics: Heartbeat RPC used to append

AppendEntries RPC

Invoked by leader to replicate log entries (§5.3); also used as heartbeat (§5.2).

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

Raft
The Basics: Request Vote

RequestVote RPC

Invoked by candidates to gather votes (§5.2).

Arguments:

term	candidate's term
candidateId	candidate requesting vote
lastLogIndex	index of candidate's last log entry (§5.4)
lastLogTerm	term of candidate's last log entry (§5.4)

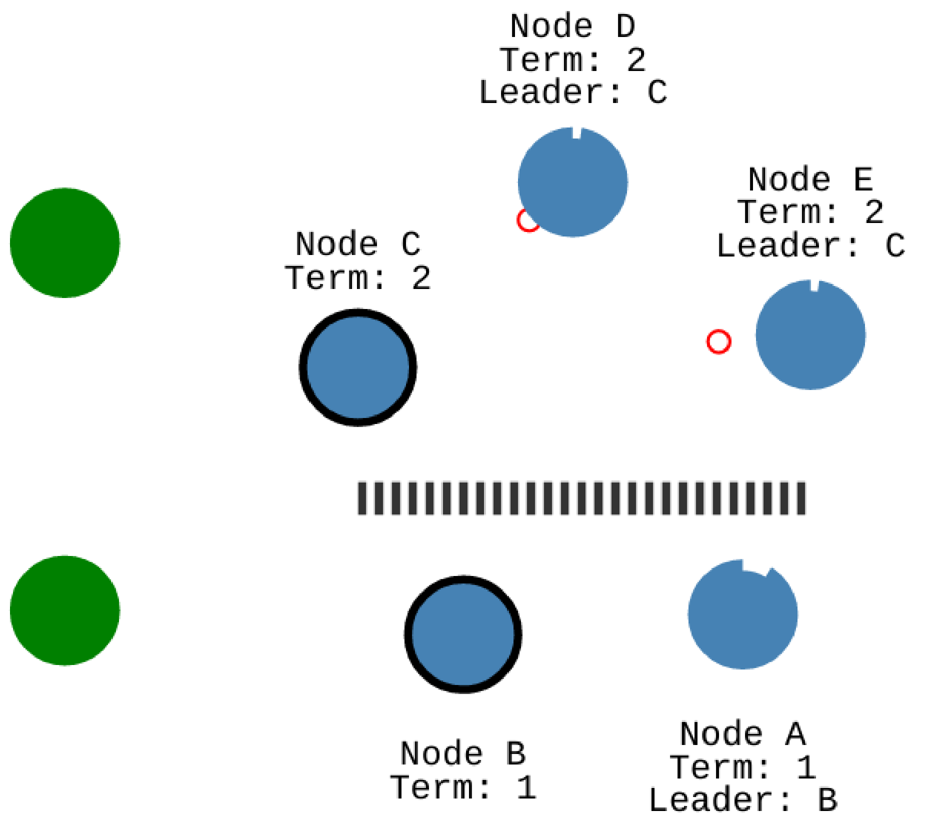
Results:

term	currentTerm, for candidate to update itself
voteGranted	true means candidate received vote

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. If votedFor is null or candidateId, and candidate's log is at least as up-to-date as receiver's log, grant vote (§5.2, §5.4)

Raft
Partition tolerance



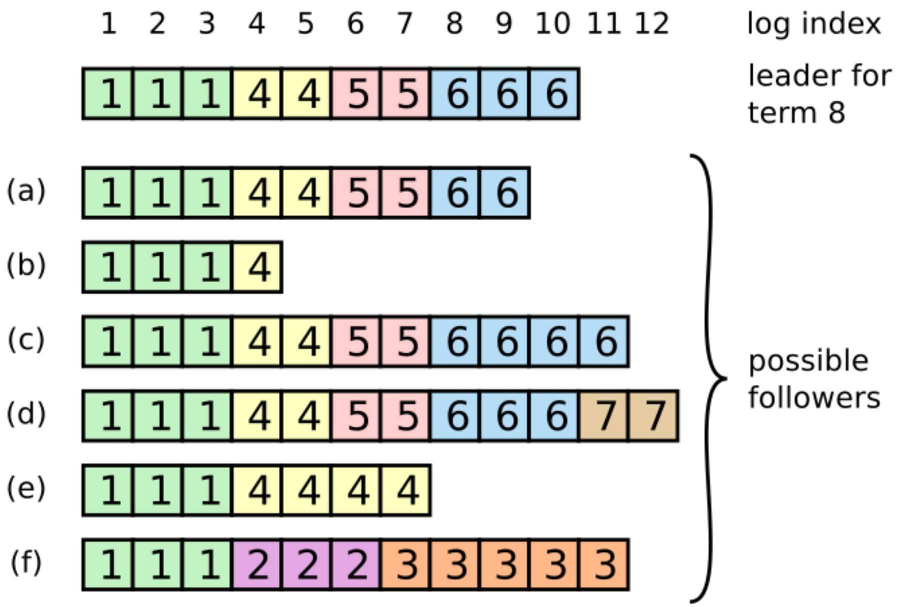
Node B consider itself a lead (from old term) but it can reach Consensus so it won't affect consistency

Raft
Follower Cases

A follower might have missed some entries as in a-b.

Or have extra "uncommitted" entries as in c-f (that should be removed)

F was a leader of term 2 that committed some changes then failed

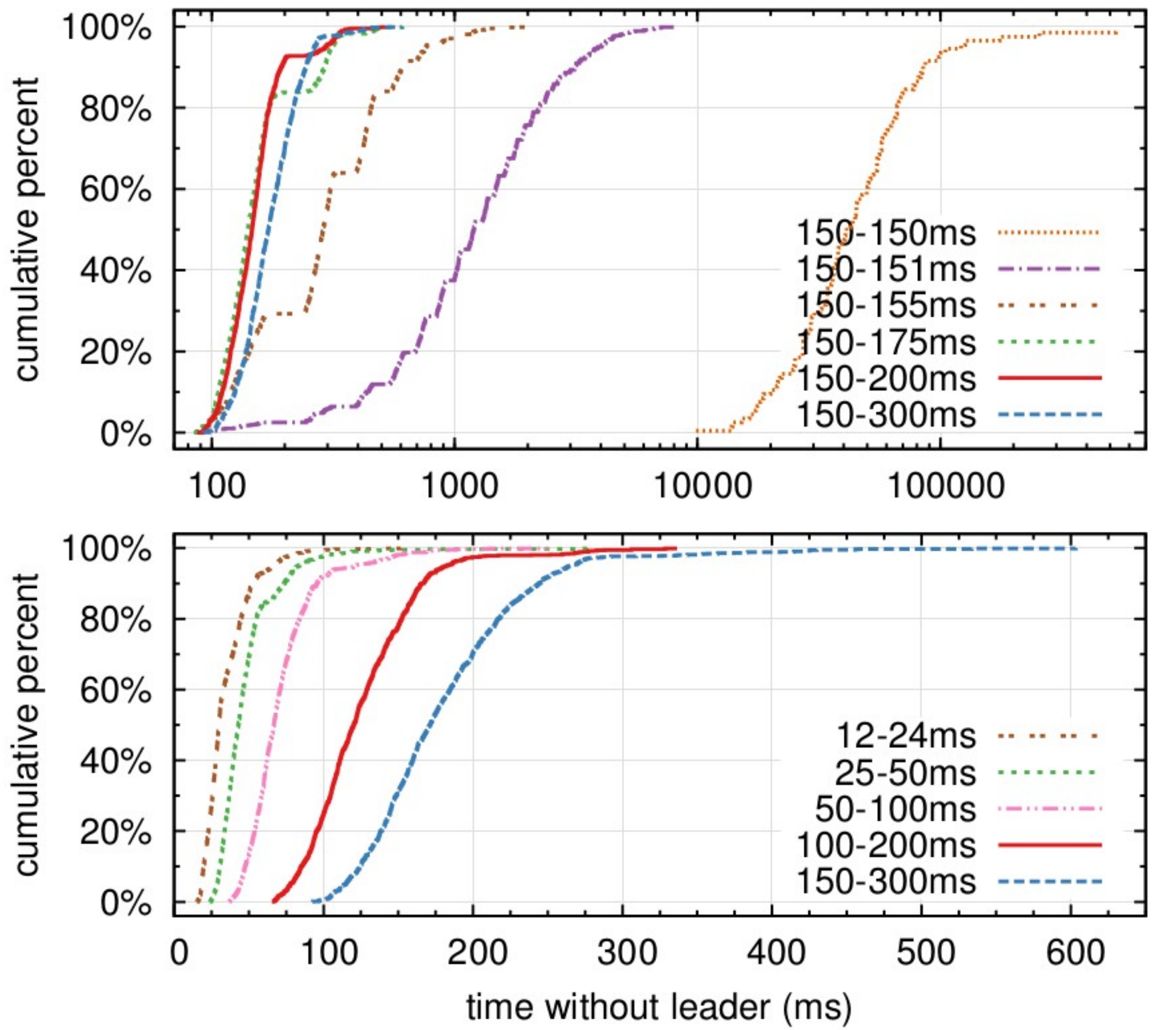


Raft
Working conditions

$$broadcastTime \ll electionTimeout \ll MTBF$$

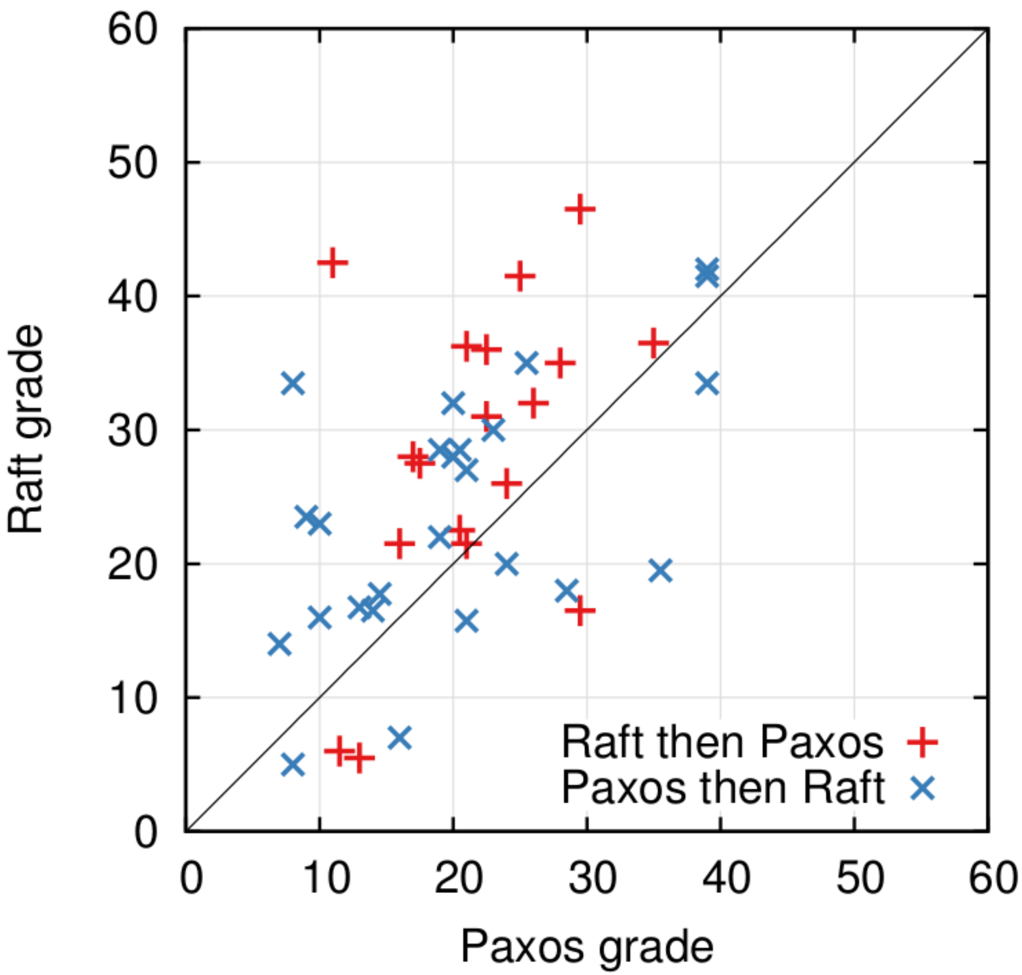
Where MTBF: is mean time between failures

Raft
Tuning for time without leader



Time without leader By tuning random timeout range

Raft
How Understandable?



Raft Resources

- The Paper
- Visualization of Raft operations
- Etcd - implementation
- Consul – another implementation
- http://raftconsensus.github.io/
- CoreOS/Fleet uses Etcd
- Google Kubernetes - a distributed system
- Mesos / Mesosphere– a distributed system