RAFT

Ankush Phulia
2014CS50279

Motivation

- “The Part-Time Parliament”, 1989 paper by Leslie Lamport describes the Paxos algorithm for distributed consensus
    - “My attempt at inserting some humor into the subject was a dismal failure. People who attended my lecture remembered Indiana Jones, but not the algorithm”
    - Took 6 years to publish - few people got it!

- “Paxos Made Simple”, 2001
    - “At PODC 2001, I got tired of everyone saying how difficult it was to understand the Paxos algorithm”
    - “In 2015, Michael Dearderuff of Amazon informed me that one sentence in this paper is ambiguous, and interpreting it the wrong way leads to an incorrect algorithm. Dearderuff found that a number of Paxos implementations on Github implemented this incorrect algorithm”

Motivation

- Paxos is considered by many to be a subtle algorithm that is famously difficult to fully grasp
  - RAFT’s Authors propose that the same guarantees and efficiency can be obtained by an algorithm that also gives better understandability

- Paxos does not lend itself well to implementation - implemented algorithm can deviate far from theoretical description - theoretical proof may often not apply in practice anymore
  - Paxos Made Live - An Engineering Perspective

- Multiple variants - which one?

Preliminaries

- Distributed Consensus
- (Multi-)Paxos
- Assumptions

Distributed Consensus

- Consensus requires the following conditions to be met:
  - Agreement - all correct processes/nodes arrive at the same value (safety)
  - Validity - value chosen is one that was proposed by a correct node (non-triviality)
  - Termination - all correct nodes eventually decide on a value (liveness)

- FLP Impossibility - consensus impossible in asynchronous distributed systems with the presence of only one faulty process
  - A system that cannot reach consensus is a system that cannot make progres
  - Some executions cannot reach consensus in bounded time

(Multi-)Paxos

- Multi-Paxos is chaining together a series of instances of Paxos to achieve consensus, on a series of values such as a replicated log
  - “Standard” Paxos considers the agreement over a single value

- Multi-Paxos uses leaders, long-term proposers who are able to omit the propose message when they were the last node to commit a value. This series of values is recorded by each node

Assumptions

- Network communication between nodes is unreliable
    - Network delays, partitions, packet loss, duplication, re-ordering
- Asynchronous environment - no upper bound exists for the delay of messages
- Byzantine failures cannot occur
- Statically configured with each node with full knowledge of all other nodes
- Cluster membership cannot change dynamically;

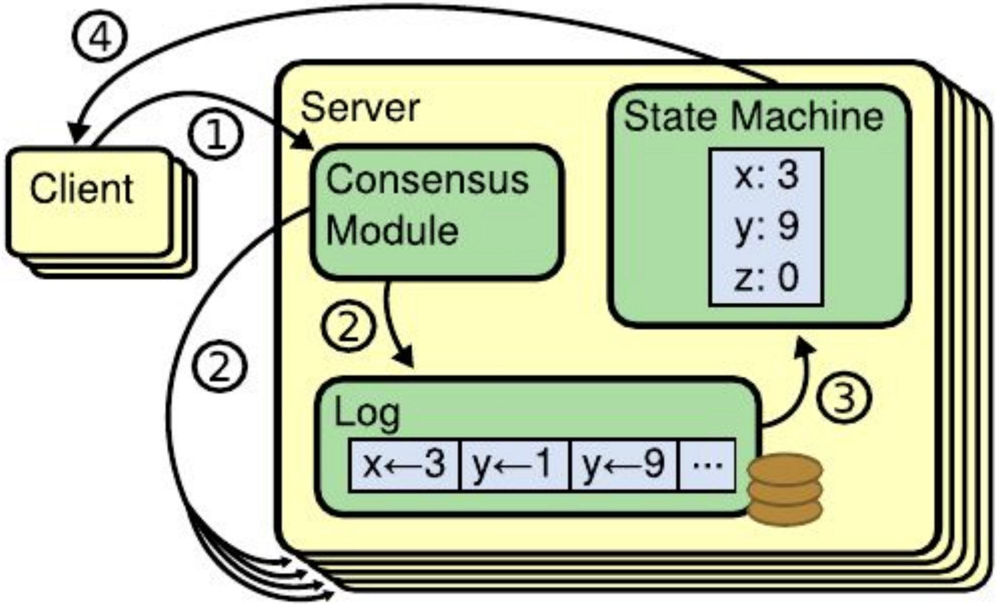
The Algorithm

- Basics
- Leader Election
- Log Replication
- Membership Changes

Basics

- Reliable, Replicated, Redundant, And Fault-Tolerant

- Replicated State Machines
  - A replicated state distributed across the cluster
  - Each node/process must agree upon the same series of state transitions



Leader Election

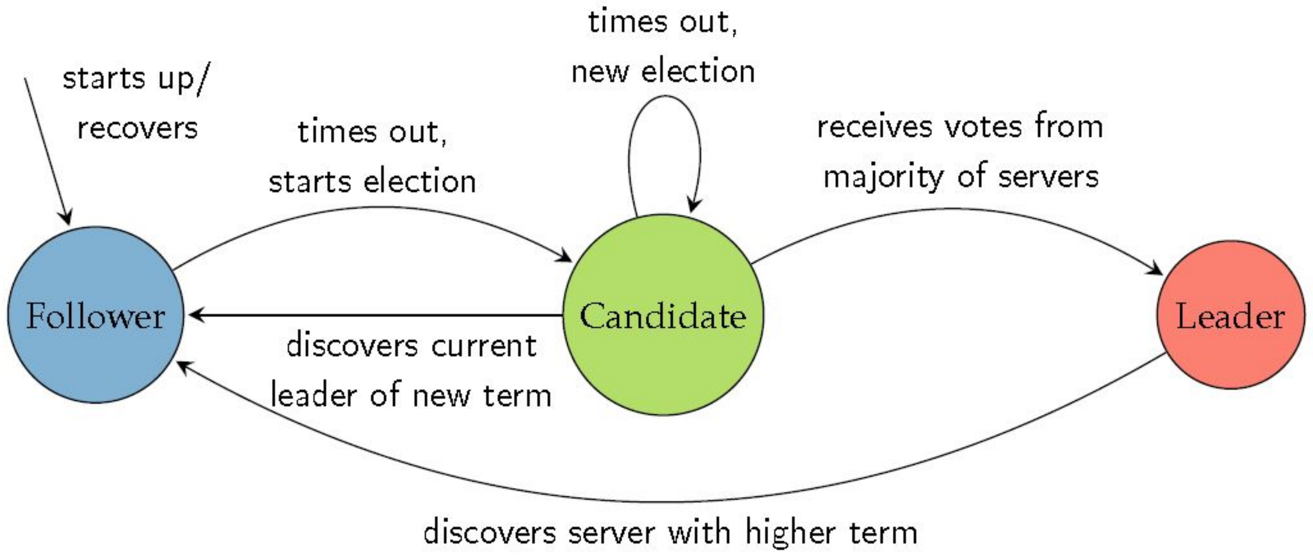


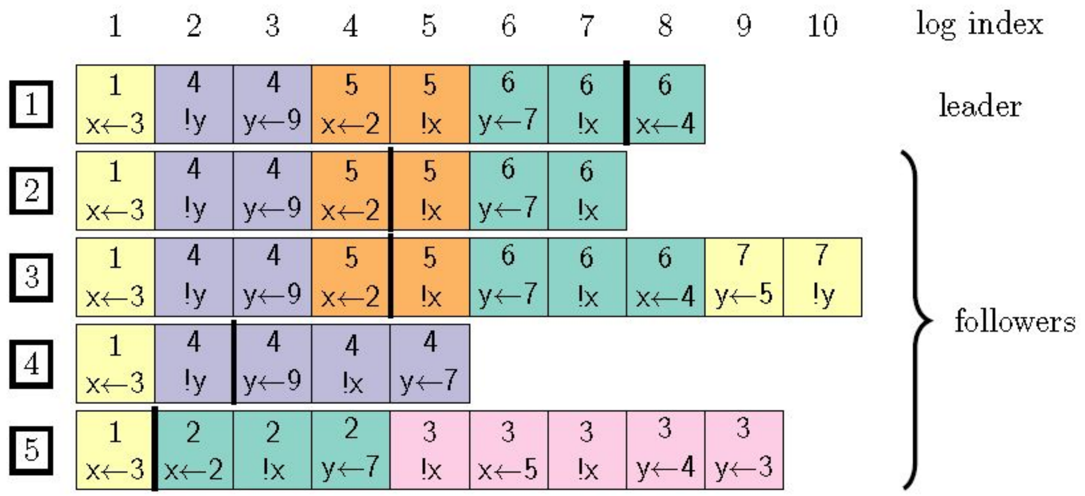
Figure 2.2: State transition model for Raft consensus.

Log Replication

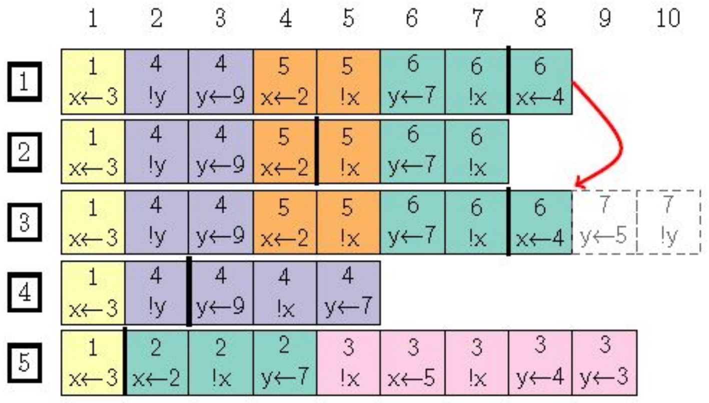
- On receipt of a command, leader assigns a term and index to the command
- Leader tries to replicate the command to the logs of a strict majority of nodes
  - Sends the previous entry as well, to ensure follower updates correctly
  - Needs ACK from nodes to be sure of success

- Extra conditions for log safety -
  - Node will only grant a vote to another if its log is at least as up-to-date
  - Leader can only commit an entry from a previous term if it has already committed an entry from current term

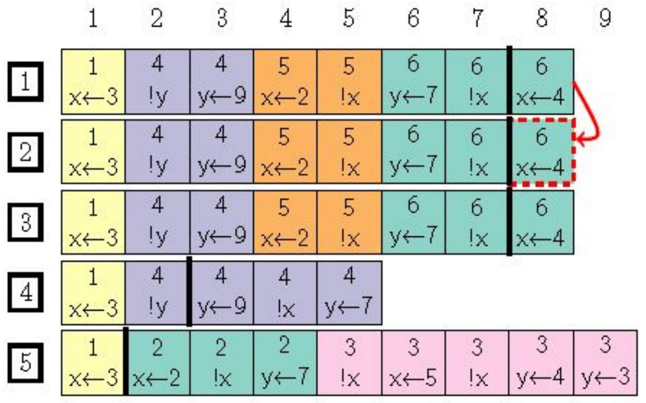
Log Replication


1 2 3 4 5 6 7 8 9 10
log index
leader
followers
1
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x | 6 x←4
2
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x
3
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x | 6 x←4 | 7 y←5 | 7 !y
4
1 x←3 | 4 !y | 4 y←9 | 4 !x | 4 y←7
5
1 x←3 | 2 x←2 | 2 !x | 2 y←7 | 3 !x | 3 x←5 | 3 !x | 3 y←4 | 3 y←3

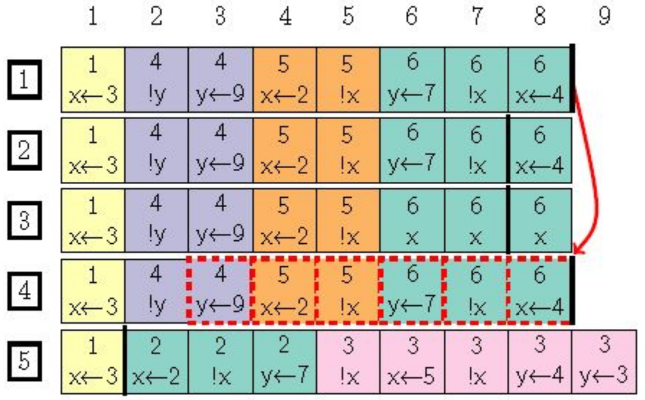
(a) Initial state of logs


1 2 3 4 5 6 7 8 9 10
1
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x | 6 x←4
2
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x
3
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x | 6 x←4 | 7 y←5 | 7 !y
4
1 x←3 | 4 !y | 4 y←9 | 4 !x | 4 y←7
5
1 x←3 | 2 x←2 | 2 !x | 2 y←7 | 3 !x | 3 x←5 | 3 !x | 3 y←4 | 3 y←3

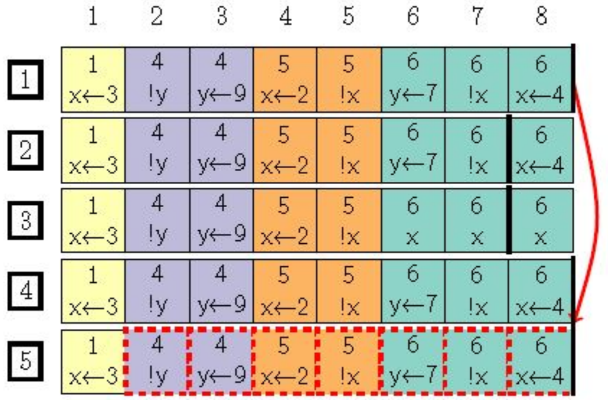
(b) bringing node 3 up-to-date


1 2 3 4 5 6 7 8 9
1
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x | 6 x←4
2
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x | 6 x←4
3
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x | 6 x←4
4
1 x←3 | 4 !y | 4 y←9 | 4 !x | 4 y←7
5
1 x←3 | 2 x←2 | 2 !x | 2 y←7 | 3 !x | 3 x←5 | 3 !x | 3 y←4 | 3 y←3

(c) bringing node 2 up-to-date


1 2 3 4 5 6 7 8 9
1
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x | 6 x←4
2
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x | 6 x←4
3
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 x | 6 x | 6 x
4
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x | 6 x←4
5
1 x←3 | 2 x←2 | 2 !x | 2 y←7 | 3 !x | 3 x←5 | 3 !x | 3 y←4 | 3 y←3

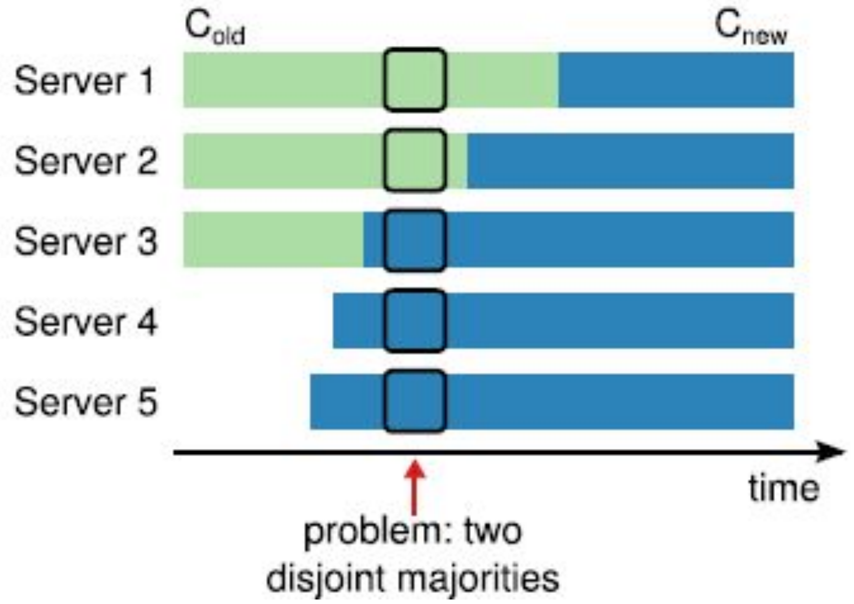
(d) bringing node 4 up-to-date


1 2 3 4 5 6 7 8
1
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x | 6 x←4
2
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x | 6 x←4
3
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 x | 6 x | 6 x
4
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x | 6 x←4
5
1 x←3 | 4 !y | 4 y←9 | 5 x←2 | 5 !x | 6 y←7 | 6 !x | 6 x←4

(e) bringing node 5 up-to-date

Membership Changes

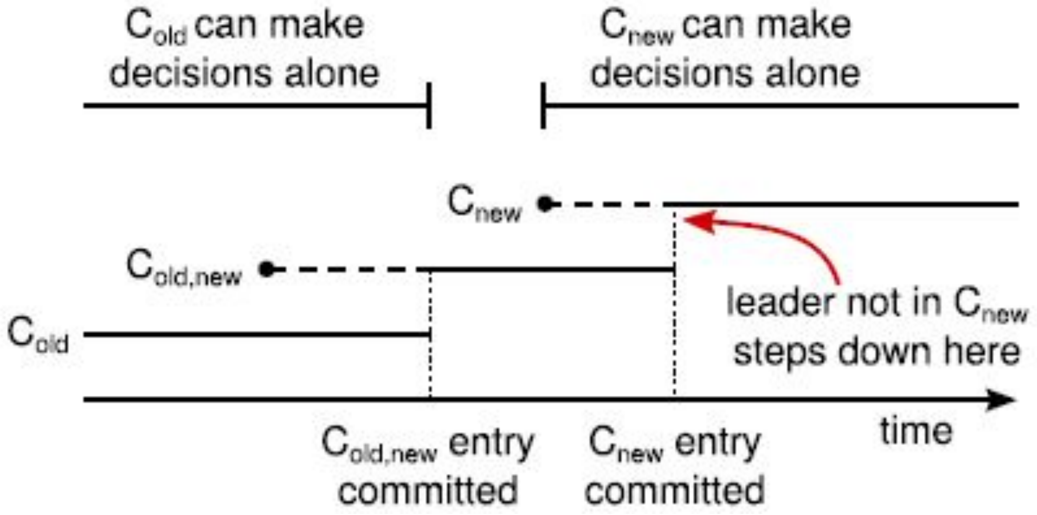
- Direct switch?
- Different machines switch at different times
- Some can keep acting according to old config



Membership Changes

- Two rounds for change
  - One to tell everyone there is a new config - add new machines
  - Another round to remove older ones

- Different majorities needed for commit
  - Over half of the older ones
  - Over half of the older AND new ones



C_{old} can make decisions alone
C_{new} can make decisions alone
C_{new}
$C_{old,new}$
C_{old}
leader not in C_{new} steps down here
$C_{old,new}$ entry committed
C_{new} entry committed
time

Properties

- Safety
- Liveness
- CAP

Safety

- Elections -
  - Each server gives only one vote - 2 candidates can't have majority
  - Votes need to be persisted on disk - in case of fail-recover

- Commitment - For log entries that are
  - Replicated on a majority of servers
  - From the current leader's term or the items preceding an AppendEntries RPC (may also be a heartbeat/ empty RPC) from the current term

- Log Safety - once a log entry has been applied to one state machine, no other command may be applied to a different state machine for that specific log entry (specified by term and index).

Safety

- Picking the Best Leader - processes deny the vote if own log is more complete.
    - This is the case if the voting server has either participated in a higher term or the same term, but having a higher lastIndex (and therefore a longer log)

- Making Logs Consistent - Append call will be tried again until eventually the last item in the log is reached and the log filled up to mirror the leader's log
    - Note that leaders log is authoritative, it remove extra entries

Safety

- Neutralizing Old Leaders - Terms are the mechanism by which the protocol takes care of stale leaders. All messages include the term of the sender and the comparison of sender’s to receiver’s term leads to the updating of the more out-of-date
    - In case of an older sender’s term, the sender reverts to follower or, in case of an older receiver’s term, the receiver reverts to follower
    - Any election necessarily updates the terms within a majority of servers
    - Once a leader is stale, it is impossible for it to commit any new log entries.

- Client interaction - Only once a command has been logged, committed, and executed on the state machine of the leader, will it then respond.
    - Every command is paired with a unique command id, respond once

Liveness

- Elections - New random timeout within each given interval for each process
    - Lower bound of the chosen interval considerably larger than the broadcast time

- Client Interactions - If the leader is unknown or dead, client may contact any of the servers, which will then redirect to the (new) leader, where the request will then be retried

- Cluster reconfiguration - cluster remains up during the entire reconfiguration process and progress is guaranteed by the same mechanisms as the rest of the protocol

C, A or P ?

- Strongly Consistent

- Partition Tolerant

- Available? Not Always
    - Leader Elections
    - Majority of the servers go down

VS. Paxos

In my experience, all distributed consensus algorithms are either

1. *Paxos*
2. *Paxos with extra unnecessary cruft, or*
3. *Broken*

- Mike Burrows

“Strong” Leadership

- Raft concentrates much of functionality with leader (interaction with clients, initiation of communication, availability)
    - Log entries flow out of the leader only, keeps it log management simple

- Leader election is an essential part of the consensus
    - Not deferred to some oracle
    - For Paxos, this is more of optimisation - consensus achieved without it
    - But this cause two-phases for basic consensus and a separate leader election

“Strong” Leadership - The flipside

- Raft’s strong leadership approach simplifies the algorithm, but precludes some performance optimizations

- Egalitarian Paxos (EPaxos) can achieve higher performance under some conditions with a leaderless approach
  - Any server can commit a command with just one round of communication as long as other commands that are proposed concurrently commute with it
  - EPaxos balances load well between servers and is able to achieve lower latency than Raft

Performance

- Raft has no performance benefits compared to a full-featured Paxos implementation

- Leader Election - converges thanks to random time-outs, original paper for Paxos mentions nothing in this regard

- Replication - when an established leader is replicating new log entries, a single round-trip from the leader to half the cluster is needed (optimal) in RAFT

Beyond RAFT

- RAFT Assumptions
- Scalability
- Scope

Assumptions

- Network communication between nodes is unreliable
    - Network delays, partitions, packet loss, duplication, re-ordering
- Asynchronous environment - no upper bound exists for the delay of messages
- **Byzantine failures cannot occur**
- Statically configured with each node with full knowledge of all other nodes
- **Cluster membership cannot change dynamically**

VS. Blockchains

- Blockchains are different
  - Byzantine Fault Tolerance - Byzantine Paxos, even less scalable
  - Dynamic network changes
  - Anonymous
  - More scalable than RAFT/Paxos

- There are similarities though
  - Asynchronous, Immutable
  - Leader takes decisions (elected vs. random)

Atomic Broadcasts

- Completely asynchronous and unordered network => Need the full complexity of Paxos/RAFT
- Totally ordered, atomic broadcast at network level =>Rreplica consistency is trivial
  - Providing totally ordered atomic broadcast requires moving the coordination overhead to a different layer
- Look for certain properties that are stronger than completely asynchronous and unordered, but weaker than totally ordered atomic, AND can be implemented very efficiently within the network
- Ordered Unreliable Multicast
- “Just say NO to Paxos overhead: replacing consensus with network ordering” - Li. et al. OSDI ‘16

QUESTIONS?