Raft Consensus Algorithm

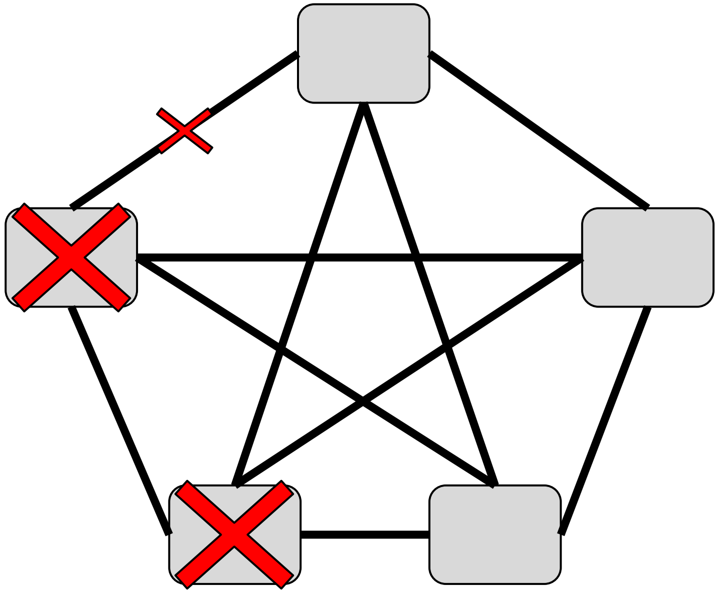
Joseph Batchik

Outline

- Basics of the Raft Protocol
- Overview of a Replicated State Machine
- Leader Election Process
- Log Replication
- Client Interactions
- Example

Consensus, what is?

- Agreement on shared state
- Achieve overall system reliability in the presence of a number of faulty processes or connections
- Multiple variations
- Example:
    - Google Chubby
    - Apache Zookeeper
    - CoreOS etcd



Overview of Raft

- Paper by Diego Ongaro and John Ousterhout (Stanford)
    - “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 its structure is different from Paxos; this makes Raft more understandable than Paxos”
- Designed to be more understandable than Paxos

Overview of Raft

- Sequential Consistency
  - "...the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program" - Leslie Lamport
-

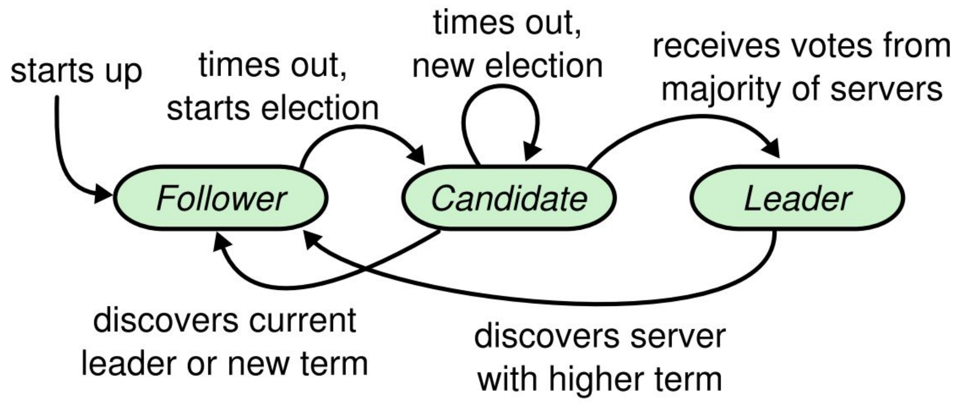
Overview of Raft

- Strong Leader
  - Log entries only flow from the leader to followers
- Leader Election
  - heartbeat between leader and followers
  - Randomized timers
- Membership Changes
  - joint consensus approach

Raft Basics

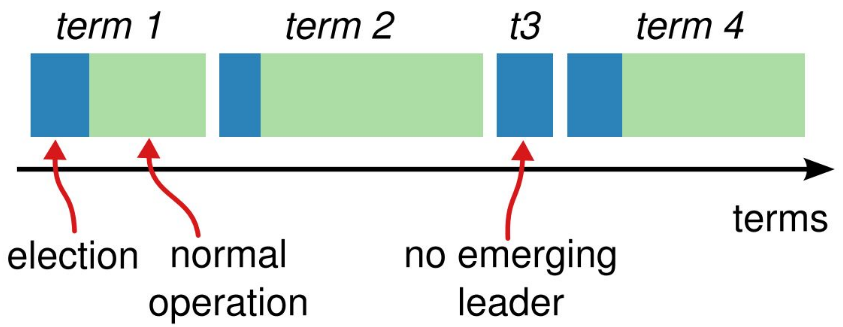
Server States

- **Leader**
  - only 1 per cluster
  - handles all client requests
  - replicates logs to followers
- **Followers**
  - starting state
  - are passive, simply reply to requests from leader
- **Candidate**
  - used to elect new leaders


starts up
times out, starts election
times out, new election
receives votes from majority of servers
Follower
Candidate
Leader
discovers current leader or new term
discovers server with higher term

Election Terms

- Raft divides time into **terms** of arbitrar
  length
- 1 leader per term
- act as a logical clock
- each server stores what it thinks is the
  current term
- current term passed with every message



The RPCs

RequestVote RPC

- used during leader elections
- used by candidates

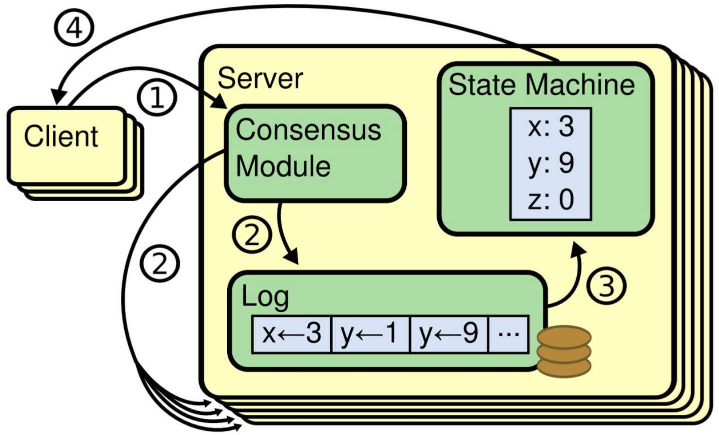
AppendEntries RPC

- used to replicate logs & send heartbeats
- sent by leader to followers

Replicated State Machines

Replicated State Machines

- The state machine approach is a general method for achieving fault tolerance and implementing decentralized control in distributed systems.
- Allows a collection of servers to:
  - maintain identical copies of the same data
  - continue operation in the face of partial failures
- uses a replicated log



Replicated Log

- Each server stores a log of **commands**
- Consensus algorithm ensures all logs are the same
- State machine executes entries in the same order
- Entries have to be **deterministic**

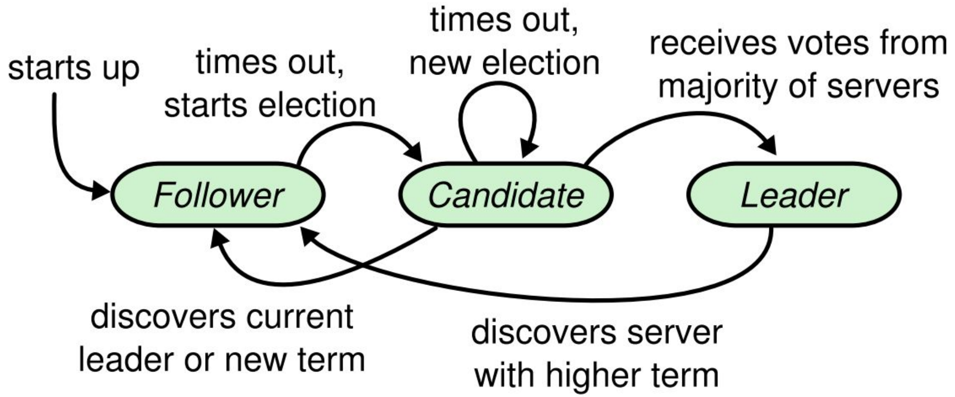
Replicated Log

- Client sends a command to one of the servers
- Server adds the command to its local log
- Server forwards the new log entry to every other server
- The state machine processes all commands up to the new entry and returns the result

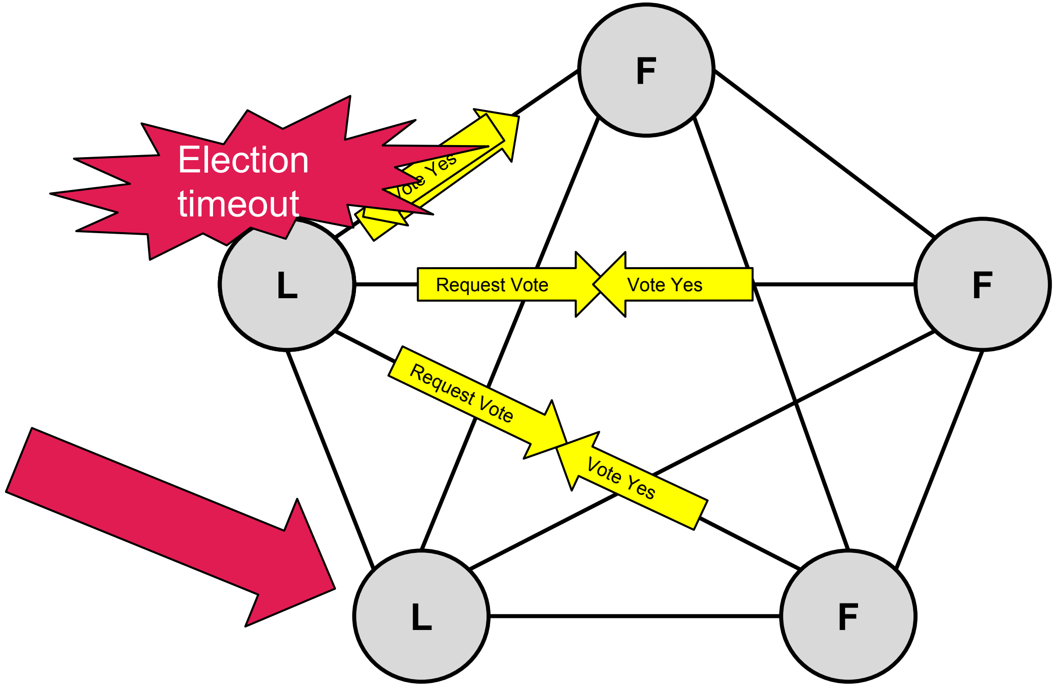
Leader Election

Leader Elections

- Leaders send periodic heartbeats to all followers
- Servers start as a **follower**
  - stay in that state as long as it receives a heartbeat from the leader
  - If a follower receives no heartbeat in a certain time, it starts the election process
- randomized election timeouts


starts up
times out, starts election
times out, new election
receives votes from majority of servers
Follower
Candidate
Leader
discovers current leader or new term
discovers server with higher term

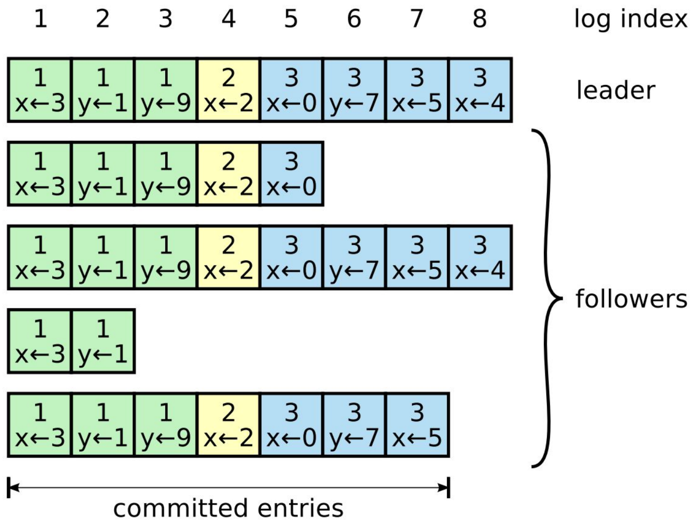
Basic Example



Log Replication

Log Replication Overview

- Clients submit commands to the leader
- Leader appends command to local log and replicated to all followers
- applies the command once it has been safely replicated
- Guaranteed that committed logs are durable



Replication Messages

```
struct Entry {
    1: required Term term;
    2: required i32 index;
    3: required EntryType type;
    4: optional binary command;
    5: optional list<admin.Server> newConfiguration;
    6: optional Configuration configurationType;
}

struct AppendEntries {
    1: required Term term;          // leader's term
    2: required ServerId leaderId;  // so follower can redirect clients
    3: required i32 prevLogIndex;   // index of log entry immediately preceding new ones
    4: required Term prevLogTerm;   // term of prevLogIndex entry
    5: required list<Entry> entires; // log entries to store (empty for heartbeat; may send more than one for efficiency)
    6: required i32 leaderCommit;   // leader's commitIndex
}

struct AppendEntriesResponse {
    1: required Term term;     // currentTerm, for leader to update itself
    2: required bool success;  // true if follower contained entry matching prevLogIndex and prevLogTerm
    3: optional i32 lastIndex; // the index of the last element in the node's log
}
```

Client Interactions

Client Interactions

- Clients of Raft send all of their requests to the leader
- Needs to deal with leader / client crashes
- Clients assign unique IDs to every command
- Prevents duplicating execution

```scala
case class Put(override val client: String, override val id: Int, key: String, value: String) extends Command(client, id)
case class Delete(override val client: String, override val id: Int, key: String) extends Command(client, id)
case class CAS(override val client: String, override val id: Int, key: String, current: String, newVal: String) extends Command(client, id)
case class Append(override val client: String, override val id: Int, key: String, value: String) extends Command(client, id)

case class GetResult(override val id: Int, value: Option[String]) extends Result(id)
case class PutResult(override val id: Int, overridden: Boolean) extends Result(id)
case class DeleteResult(override val id: Int, deleted: Boolean) extends Result(id)
case class CASResult(override val id: Int, replaced: Boolean) extends Result(id)
case class AppendResult(override val id: Int, newValue: String) extends Result(id)
```

Actually Useful

The Raft consensus algorithm can be found in production across industry

RethinkDB

KUDU

etcd

Log Cabin

github.com/logcabin

Questions?

Example Time