How Unreliable Computers Can Usually Agree (Sort Of)
A Brief Tour of the Raft Algorithm

Laura Hampton
@incunabulista

OSCON 2018
Introduction to Consensus
Computers are terrible

- Computers prone to failure
- Unreliable networks
- Availability
- How to avoid involving people when things fail
Effective Consensus Algorithms

- Consistency between participant state machines
- Tolerant of failure of 1 or more participants
- Tolerant of unreliable networks and network partition
Features of Consensus Algorithms

- Agreement
- Validity
- Termination
CAP Theorem

- Consistency
- Availability
- Partition Tolerance
What Could Possibly Go Wrong?

- Fail-stop
- Fail-recover
- Network partition
- Byzantine failure
Byzantine Failure
2-Phase Commit
2-Phase Commit - Leader Fail-Stop
A PAGER GOES OFF
3-Phase Commit
3-Phase Commit: Fail-Recover
A PAGER GOES OFF
Raft in Theory
Uses for Raft
Election and Log Propagation

Election  Log Propagation
Raft Cluster

FOLLOWER
TERM: 0

FOLLOWER
TERM: 0

FOLLOWER
TERM: 0

FOLLOWER
TERM: 0

FOLLOWER
TERM: 0
Leader Election

CANDIDATE
TERM: 1

FOLLOWER
TERM: 1

FOLLOWER
TERM: 1

FOLLOWER
TERM: 1

FOLLOWER
TERM: 1

FOLLOWER
TERM: 1
Log Replication

CLIENT

LEADER

FOLLOWER

FOLLOWER

FOLLOWER

FOLLOWER
Vote Message

Vote(
    status = self.status,
    sender_id = self.id,
    sender_log_length = len(self.log),
    recipient_id=recipient,
    term = self.term,
    vote=None #None to request, 1 vote yes, 0 vote no
)
Add Entries Message

AddEntry(
    status = self.status,
    sender_id = self.id,
    recipient_id = recipient,
    term = self.term,
    commit = False,
    success = None,#False to reject, True if logs match
    last_log_index = self.last_log_index,
    last_log_entry = self.last_log_entry, #None for heartbeat
    new_log_entry = self.new_log_entry #None for heartbeat
)
Log Safety

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server U</td>
<td>10</td>
<td>7</td>
<td>14</td>
<td>33</td>
<td>52</td>
<td>81</td>
<td>82</td>
<td>98</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>Server T</td>
<td>10</td>
<td>7</td>
<td>14</td>
<td>33</td>
<td>52</td>
<td>81</td>
<td>82</td>
<td>98</td>
<td>17</td>
<td>2</td>
</tr>
</tbody>
</table>
Leader Completeness

CANDIDATE
LOG: [1]

FOLLOWER
LOG: [1, 2]

FOLLOWER
LOG: [1, 2]

FOLLOWER
LOG: [1, 2]

FOLLOWER
LOG: [1]
Split Vote

CANDIDATE
VOTES: 2

FOLLOWER

CANDIDATE
VOTES: 2

FOLLOWER
Network Partition
Log Compaction

SNAPSHOT
TERM:  3
INDEX:  8
Membership Changes

Old config makes decisions

Old config

Old, new config

New config

New config makes decisions

Leader not in new config steps down here
Success!
A Brief Trip to a Greek Island
Raft in Practice
Setting up a Cluster

- Reliability
- Maintenance schedule
- Risk
- Performance
- Cost
Performance Considerations
Where to Locate Replica Nodes
Raft and etcd
Raft as Implemented in MongoDB

- Priority
- Arbiters
- Chaining
- Delayed members
Priority

LEADER
PRIORITY: 3
VOTES: 1

FOLLOWER
PRIORITY: 1
VOTES: 1

FOLLOWER
PRIORITY: 1
VOTES: 1

FOLLOWER
PRIORITY: 2
VOTES: 1

FOLLOWER
PRIORITY: 0
VOTES: 0
Arbiters

ARBITER
VOTES: 1

CANDIDATE
PRIORITY: 3
VOTES: 1

FOLLOWER
PRIORITY: 1
VOTES: 1

FOLLOWER
PRIORITY: 1
VOTES: 1

FOLLOWER
PRIORITY: 2
VOTES: 1

FOLLOWER
PRIORITY: 0
VOTES: 0
Chaining
Delayed Participants

- **Leader**
  - Priority: 3
  - Votes: 1

- **Follower**
  - Priority: 2
  - Votes: 1

- **Follower**
  - Priority: 1
  - Votes: 1

- **Follower**
  - Priority: 2
  - Votes: 1

- **Delayed**
  - Priority: 0
  - Votes: 1
Resources


Raft visualization with explanations: http://thesecretlivesofdata.com/raft/

Thank you