In Search of an Understandable Consensus Algorithm

Diego Ongaro and John Ousterhout

Papers We Love (pwl) meetup Zurich
Animesh Trivedi
Why I have chosen Raft?

Claim to fame: Understandable (not necessarily easy!) [consensus] algorithm

Open-source [https://raft.github.io/](https://raft.github.io/), multiple implementations

- C, C++, Java, Go, Rust, Scala, Haskell, Python, PHP, JavaScript, Dash, D, Ruby, more?

Taught at various universities

- Stanford, Berkeley, Princeton, UWash, Brown, MIT, Harvard, Duke, IIT and more

Used in production code

- etcd, CockroachDB, InfluxDB, Apache Kudu, Apache Ratis
In Search of an *Understandable* Consensus Algorithm*

Diego Ongaro and John Ousterhout

Papers We Love (pwl) meetup Zurich
Animesh Trivedi

*The thesis version*
What is a Consensus

Reaching an agreement in a distributed setting when (this is the failure model):

- There are no global clocks
- Machines operate at different speeds
- Machines can fail (fail-stop)
- The network can fail
- Messages can be reordered, dropped, or replicated
- Messages can be take arbitrary long to deliver

A practical solution must ensure that

1. Safety: A single proposed value is chosen;
2. Liveness: Some progress is made eventually;

(also known as The FLP Theorem)
Applications of Consensus

building a storage service

key-value store

client machines

+ Pros: easy, and consistent
- Cons: Failure?
Applications of Consensus

building a **highly reliable** storage service

key-value store

client machines

+ Pros: easy, and consistent
- Cons: Failure?

client machines

consensus to reason about the state of the replicated system
State-Machine Replication using Log

state machine  consensus

state machine  consensus

state machine  consensus
State-Machine Replication using Log

client request

consensus

state machine

log

log

log
State-Machine Replication using Log

1 Reaching a consensus

client request
State-Machine Replication using Log

1. Reaching a consensus
2. Appending to the log

client request
State-Machine Replication using Log

1. Reaching a consensus
2. Appending to the log
3. Executing the command

client request → client response
Raft: In a Nutshell

What is Raft? An algorithm for managing a replicated log of decisions.

How does it do it? It chooses a leader which makes decisions about the log status and forces other servers to follow its decisions.

What is that good for? One can use replicated Raft log to implement the replicated state machine.

Is Raft safe? Yes, it is proven to be safe. Its TLA+ specification is available.
Raft: Basic Commit Operation

$(2f + 1)$ config for $f$ failures (fail-stop, non-Byzantine), and use RPCs for messages.
Raft: Basic Commit Operation

$(2f + 1)$ config for $f$ failures (fail-stop, non-Byzantine), and use RPCs for messages

1. Client request comes in
Raft: Basic Commit Operation

$(2f + 1)$ config for $f$ failures (fail-stop, non-Byzantine), and use RPCs for messages

1. Client request comes in
2. The leader applies locally and sends the command out to all servers
Raft: Basic Commit Operation

(2f +1) config for f failures (fail-stop, non-Byzantine), and use RPCs for messages

1. Client request comes in
2. The leader applies locally and sends the command out to all servers
3. All servers append the entry in their logs
Raft: Basic Commit Operation

$(2f + 1)$ config for $f$ failures (fail-stop, non-Byzantine), and use RPCs for messages

1. Client request comes in
2. The leader applies locally and sends the command out to all servers
3. All servers append the entry in their logs
4. As soon as majority replies the command is committed
Raft: Basic Commit Operation

(2f + 1) config for f failures (fail-stop, non-Byzantine), and use RPCs for messages

Issues to solve:

1. What happens when the leader crashes? [Leader Election]
2. How to ensure all servers have the same log entries in sequence? [Log replication algorithm]
Raft Basics

Time is split in terms with at most one leader in each term
Raft Basics

Time is split in *terms* with at most one leader in each term.
Raft Basics

Time is split in *terms* with at most one leader in each term.

- Election
- Normal mode
Raft Basics

Time is split in *terms* with at most one leader in each term.

Election

Normal mode

$k$
Raft Basics

Time is split in *terms* with at most one leader in each term.
Raft Basics

Time is split in *terms* with at most one leader in each term.
Raft Basics

Time is split in *terms* with at most one leader in each term.

Election

Normal mode

election without a leader
Raft Basics

Time is split in *terms* with at most one leader in each term

A server can be: the Leader, a Follower, a Candidate

- **Follower**: only accepts commands from the leader
- **Candidate**: tries to become the leader
- **Leader**: manages client requests and followers
Raft Basics

Time is split in *terms* with at most one leader in each term.

A server can be: the Leader, a Follower, a Candidate.
Raft Leader Election

S1
term: 0
vote to: _
vote from: _

S2
term: 0
vote to: _
vote from: _

S3
term: 0
vote to: _
vote from: _
Raft Leader Election

Who calls the election? Use randomized timeouts in a range

S1
- term: 0
- vote to: _
- vote from: _

S2
- term: 0
- vote to: _
- vote from: _

S3
- term: 0
- vote to: _
- vote from: _
Raft Leader Election

S1
---
term: 0
vote to: _
vote from: _

S2
---
term: 0
vote to: _
vote from: _

timeout

S3
---
term: 0
vote to: _
vote from: _
Raft Leader Election

Step 1: declare yourself candidate, and increase the term

S1
- term: 0
- vote to: _
- vote from: _

S2
- term: 1
- vote to: _
- vote from: _

S3
- term: 0
- vote to: _
- vote from: _
Raft Leader Election

Step 2: vote for yourself

- **S1**
  - term: 0
  - vote to: _
  - vote from: _

- **S2**
  - term: 1
  - vote to: S2
  - vote from: S2

- **S3**
  - term: 0
  - vote to: _
  - vote from: _
Raft Leader Election

Step 3: ask for votes from others, prepare RequestVoteRPC [term, ID]
Raft Leader Election

Step 4: followers increase their term, and grant vote if they haven’t so far
Raft Leader Election

Step 5: check if the majority has voted for you?

S1
- term: 1
- vote to: S2
- vote from: _

S2
- term: 1
- vote to: S2
- vote from: S2, S1

S3
- term: 1
- vote to: S2
- vote from: _
Raft Leader Election

Step 6: If yes, declare yourself the leader and send out heart-beats

<table>
<thead>
<tr>
<th>S1</th>
<th></th>
<th>S2</th>
<th></th>
<th>S3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>term: 1</td>
<td>vote to: S2</td>
<td>term: 1</td>
<td>vote to: S2</td>
<td>term: 1</td>
<td>vote to: S2</td>
</tr>
<tr>
<td>vote from: _</td>
<td></td>
<td>vote from: S2, S1</td>
<td></td>
<td>vote from: _</td>
<td></td>
</tr>
</tbody>
</table>
Raft Leader Election

What could go wrong here?

Multiple machines timeout at the same time?
Use randomized timeouts

None of the machines gets the majority?
Another round of elections

The candidate crashes?
If no-one hears from the new leader, another server will timeout and initiate an election
Issues to solve:

1. What happens when the leader crashes? [Leader Election]
2. How to ensure all servers have the same log entries in sequence? [Log replication algorithm]
Log Replication : Basics

(Recap): “Raft is an algorithm for managing replicated log”

The leader services client commands

- Append them in its log and send it out to followers (AppendEntryRPC)
- Difference between “replicated” and “committed”
- Difference between “committed” and “known to be committed”

The goal of Raft is to

- Ensure an identical log in the presence of failures
- Ensure that no committed entry is ever lost
Log Replication: the AppendEntryRPC Summary

**Request:**
- The current leaderID
- The current term
- prevIndex and prevTerm
- New entries[] to append
- Last committed entry

**Response:**
- If the log matches, then accept and ACK
- Otherwise, reject and NACK
- The current term that the follower knows

\[
\text{leaderLog}(\text{prevIndex}, \text{prevTerm}) = \text{followerLog}(\text{prevIndex}, \text{prevTerm})
\]

If there was an election

Identify holes/inconsistencies
Log Replication: An Example Problem

A log entry is identified by its index and its term

Index: 1  2  3  4  5  6  7  8  9  10  11  12
F0 F1 F2 F3 F4 F5 F6
Log Replication : An Example Problem

Let’s say F0 is the leader and it got a client command to commit
Log Replication : An Example Problem

As soon as the majority has it, the entry is committed (but is not known)
Log Replication: An Example Problem

With the next entry, the commitment of the last index can be announced.
Log Replication : An Example Problem

All good so far, then someone calls an election
Log Replication: An Example Problem

F6 become the leader and accepts 3 entries for the term 2

Index:  1      2      3      4      5      6     7      8      9     10    11    12

F0   1 | 1 | 1
F1   1 | 1 | 1
F2   1 | 1 | 1
F3   1 | 1 | 1
F4   1 | 1 | 1
F5   1 | 1 | 1
L     1 | 1 | 1

2 2 2
Log Replication: An Example Problem

... becomes slow, someone calls an election, but F6 wins again!
Log Replication: An Example Problem

This time F5 wins and accepts 4 new entries for term 4
Log Replication: An Example Problem

It manages to replicate 4th and 5th entry but failed after that. F2 is slow. Here, 4th index is committed and known, and 5th is implicitly committed.
Log Replication: An Example Problem

Now, F1 wins the election for the term 5 and replicates 6th and 7th entries. When it tries to commit the 6th entry, it will announce that index 5 is committed.

<table>
<thead>
<tr>
<th>Index: 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>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Log Replication : An Example Problem

This time F3 wins the election for the term 6 and accepts 4 entries
Log Replication : An Example Problem

While replicating the index 8, F3 confirmed that index 7 is committed.
Log Replication: An Example Problem

This time F4 becomes the leader and accepts 2 entries for the term 7 and but dies.

<table>
<thead>
<tr>
<th>Index</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>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>F5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>F6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Log Replication: An Example Problem

Now L0 becomes the leader again and we have ...

Index:  1      2      3      4      5      6     7      8      9     10    11    12

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>
Log Replication: An Example Problem

Now L0 becomes the leader again and we have ...

<table>
<thead>
<tr>
<th>Index</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>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>F4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>F6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

committed and known committed but now known missing committed entries interesting ones? wrong entries
Log Replication: Raft’s Solution

The leader’s log is THE log

No provisions for finding and reconciling holes

Log information flows in one direction, from the leader to followers

The leader can overwrite follower’s log with its entries

The leader also tells which entries are committed (to apply the state machine)

How does the Leader do it: find the last point where its log and a follower’s log disagree and from there on re-write follower’s log
Log Replication: An Example Problem

Try to match index by index

Index: 1  2  3  4  5  6  7  8  9  10  11  12
L    1  1  1  4  4  5  5  6  6  6  8
F1   1  1  1  4  4  5  5  6  6  8  8
F2   1  1  1
F3   1  1  1  4  4  5  5  6  6  6  6
F4   1  1  1  4  4  5  5  6  6  6  7  7
F5   1  1  1  4  4  4  4
F6   1  1  1  2  2  2  3  3  3  3

Commit 8 with (prevIndex:10, prevTerm:6)? No
Log Replication: An Example Problem

Try to match index by index

<table>
<thead>
<tr>
<th>Index</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>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>F5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Commit 6 with (prevIndex:9, prevTerm:6)? YES!
Log Replication: An Example Problem

Try to match index by index (see index 10 is now implicitly committed)

<table>
<thead>
<tr>
<th>Index:</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>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>F1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>F2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>F3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>F4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>F5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>F6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Commit 8 with (prevIndex:10, prevTerm:6)?
YES!
Log Replication: An Example Problem

And implicitly discover which entries are now committed (see index 9 and 10)

You overwrite entries in the follower’s log
Log Replication: An Example Problem

If no failures then all log entries converges and all committed entries and discovered and announced.
But wait, there is more!

Index 10 survived and got committed. What if F3 became a leader? Or F4? Or worse F6?

Index: 1  2  3  4  5  6  7  8  9  10  11  12

interesting ones?
Raft Leader Election: Revisited

RequestVoteRPC [term, ID]
ResponseVoteRPC : the first one (with a higher term) gets the vote

Index: 1  2  3  4  5  6  7  8  9  10  11  12
L    1  1  1  4  4  5  5  6  6  6  
F1   1  1  1  4  4  5  5  6  6
F2   1  1  1
F3   1  1  1  4  4  5  5  6  6  6  6
F4   1  1  1  4  4  5  5  6  6  6  7  7
F5   1  1  1  4  4  4  4
F6   1  1  1  2  2  2  3  3  3  3
Raft Leader Election: Revisited

RequestVoteRPC [term, ID] + (last index and term)
ResponseVoteRPC: the first one (with a higher term) gets the vote + (given that candidate’s log is as up-to-date as the voter’s)

Index: 1 2 3 4 5 6 7 8 9 10 11 12

L 1 1 1 4 4 4 5 5 5 6 6 6
F1 1 1 1 4 4 4 5 5 6 6 6 6
F2 1 1 1 4 4 4 5 5 6 6 6 6
F3 1 1 1 4 4 4 5 5 6 6 6 6
F4 1 1 1 4 4 4 5 5 6 6 6 7
F5 1 1 1 4 4 4 4 3 3 3 3 3
F6 1 1 1 2 2 2 3 3 3 3 3 3

Up-to-date?
1. Compare terms
2. Compare length
Raft Leader Election: Revisited

RequestVoteRPC \([\text{term, ID}] + (\text{last index and term})\)
ResponseVoteRPC: the first one (with a higher term) gets the vote + (given that candidate’s log is as up-to-date as the voter’s)

| Index: 1 2 3 4 5 6 7 8 9 10 11 12 |
|------|------|------|------|------|------|------|------|------|------|------|------|
| L    | 1    | 1    | 1    | 4    | 4    | 5    | 5    | 6    | 6    | 6    | 6    |
| F1   | 1    | 1    | 1    | 4    | 4    | 5    | 5    | 6    | 6    | 6    | 6    |
| F2   | 1    | 1    | 1    | 4    | 4    | 5    | 5    | 6    | 6    | 6    | 6    |
| F3   | 1    | 1    | 1    | 4    | 4    | 5    | 5    | 6    | 6    | 6    | 6    |
| F4   | 1    | 1    | 1    | 4    | 4    | 5    | 5    | 6    | 6    | 6    | 6    |
| F5   | 1    | 1    | 1    | 4    | 4    | 4    | 4    | 6    | 6    | 7    | 7    |
| F6   | 1    | 1    | 1    | 2    | 2    | 2    | 3    | 3    | 3    | 3    | 3    |

Vote?
F0: higher term in the log
F1: higher term in the log
F2: OK
F3: higher term in the log
F4: higher term in the log
F5: higher term in the log
F6: OK
Raft Leader Election: Revisited

RequestVoteRPC \([\text{term, ID}] + (\text{last index and term})\)
ResponseVoteRPC: the first one (with a higher term) gets the vote \( + (\text{given that candidate’s log is as up-to-date as the voter’s})\)

<table>
<thead>
<tr>
<th>Index</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>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
<td>L</td>
</tr>
<tr>
<td>F1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td>L</td>
</tr>
<tr>
<td>F2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td>L</td>
</tr>
<tr>
<td>F3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
<td>L</td>
</tr>
<tr>
<td>F4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>L</td>
</tr>
<tr>
<td>F5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>L</td>
</tr>
<tr>
<td>F6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td>L</td>
</tr>
</tbody>
</table>
That is it!

Obviously there are more details about

- Client interaction
- Membership management
- Log Compaction
- State-machine management
- Checkpointing and restore
- Practical guidelines
  - How to deploy
  - Timeout recommendations
  - ...

Figure 3.1: A condensed summary of the Raft consensus algorithm (excluding membership changes, log compaction, and client interaction). The server behavior in the lower-right box is described as a set of rules that trigger independently and repeatedly. Section numbers such as §3.4 indicate where particular features are discussed. The formal specification in Appendix B describes the algorithm more precisely.
Understandability
...in Comparison to What?

Paxos.

A set of protocols (not islands), proposed by Leslie Lamport
Writing about a lost civilization allowed me to eliminate uninteresting details and indicate generalizations by saying that some details of the parliamentary protocol had been lost. To carry the image further, I gave a few lectures in the persona of an Indiana-Jones-style archaeologist, replete with Stetson hat and hip flask.

“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. People reading the paper apparently got so distracted by the Greek parable that they didn’t understand the algorithm.”

“I submitted the paper to TOCS in 1990. All three referees said that the paper was mildly interesting, though not very important, but that all the Paxos stuff had to be removed.”

“When Ed Lee and I (Chandu Thekkath) were working on Petal we needed some sort of commit protocol to make sure global operations in the distributed system completed correctly in the presence of server failures. [...] Leslie gave Ed a copy of the Part-Time Parliament tech report, which we both enjoyed reading. I particularly liked its humour and to this day, cannot understand why people don’t like that tech report.”

“Meanwhile, the one exception in this dismal tale was Butler Lampson, who immediately understood the algorithm’s significance. He mentioned it in lectures and in a paper, and he interested Nancy Lynch in it.”
Which Paxos: A Brief History

the idea was there

submission to TOCS

Petal/Frangipani at DEC

TOCS published

Disk Paxos

Cheap Paxos

Fast Paxos

Generalized Paxos

Stoppable Paxos

Mencius

Vertical Paxos

EPaxos

Taken from http://paxos.systems/variants.html
Paxos

Paxos has become synonym with consensus

Paxos is a broad term for a whole family of consensus protocols

Paxos’s claim to fame is its elegant and thorough treatment of the problem

The Problem: “reaching consensus on a single value in an asynchronous setting”

+ Very well understood and analyzed problem
- How often does a system choose a single value? (Multi-Paxos is not specified precisely in literature)
Paxos Challenges

Reaching a consensus on a single-value is just the first step towards highly-available systems

1. How to reach a consensus on a single value?
2. How to reach consensus on multiple values?
3. How to maintain the consensus/decision log?
4. How to maintain the state-machine?
5. How to manage cluster membership?
6. How clients interact with the system and what kind of guarantees they get?
7. Bootstrapping and recovery of the system?
8. Performance optimizations, and maintenance?
Paxos Followups

- How to build a highly available system using consensus. Lampson, 1996
- Paxos made simple, Lamport, SIGACT 2001
- The ABCD’s of Paxos, Lampson, PODC 2001
- Deconstructing Paxos, Boichat et al., SIGACT 2003
- Generalized consensus and Paxos, Lamport, 2005
- Paxos made live: an engineering perspective, Chandra et al., PODC 2007
- Paxos Made Practical. Mazieres, 2007
- Paxos for system builders: an overview, Kirsch et al., LADIS 2008
- Paxos made moderately complex, Van Renesse, 2012
- ... and many more

“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.”
- Chandra et al., PODC’07

“The dirty little secret of the NSDI community is that at most five people really, truly understand every part of Paxos ;-). —NSDI reviewer” - Ongario, Raft presentations
Paxos Challenges

Reaching a consensus on a single-value is just the first step towards highly-available systems

1. How to reach a consensus on a single value?
2. How to reach consensus on multiple values?
3. How to maintain the consensus/decision log?
4. How to maintain the state-machine?
5. How to manage cluster membership?
6. How clients interact with the system and what kind of guarantees they get?
7. Bootstrapping and recovery of the system?
8. Performance optimizations, and maintenance?
Raft Approach

Reaching a consensus on a single-value is just the first step towards highly-available systems

1. How to reach a consensus on a single value?
2. How to reach consensus on multiple values?
3. How to maintain the consensus/decision log?
4. How to maintain the state-machine?
5. How to manage cluster membership?
6. How clients interact with the system and what kind of guarantees they get?
7. Bootstrapping and recovery of the system?
8. Performance optimizations, and maintenance?
My Subjective Impression

Is Raft easier to understand and explain? Yes
My Subjective Impression

Is Raft easier to understand and explain? Yes

+ An interesting read of a complete system design
+ Fewer numbers of states to consider
+ Intuition is easy, e.g., do timeouts reset after an election? Yes. As they meant to break ties in case of a split vote, each machine sets a new timeout for a new election/term

But we are comparing a Raft system with Paxos protocol
My Subjective Impression

Is Raft easier to understand and explain? Yes

+ An interesting read of a complete system design
+ Fewer numbers of states to consider
+ Intuition is easy, e.g., do timeouts reset after an election? Yes. As they meant to break ties in case of a split vote, each machine sets a new timeout for a new election/term

But we are comparing a Raft system with Paxos protocol

Is everything sorted out in Raft?

- Flaky machine (liveness vs. safety), client interaction is fuzzy (!), Byzantine faults, slow leader, non-commutative optimizations,...
Some Amusement from the User Study

“Cool idea, I think it’s what Paxos should have been.”

“The quizzes were too long. Could not complete in the time provided. [...]”

“Both are super complex!”

“Paxos is easier to understand because it does not have any many details as Raft. [...]”

“Good job on the lecture videos.”

“I obviously noticed that Raft and Paxos are very similar - to the point that I feel like Raft is actually paxos presented differently. [...]”

“Videos were a bit dry”

“Ousterhout is a boss. Thanks for the lectures!”
Discussion
Raft: Client Interaction - 1/3

How does a client find a Raft cluster servers?
- Use network broadcast or DNS lookups (requires network help)

How does a client find the leader?
- A follower can tell the identity of the leader to a client

How does a client request is serviced?
- The leader processes a the request by replicated it across the cluster
Raft: Client Interaction - 2/3

Consistency semantics? As such it would provide “at-least once” semantics, why?

- Leader failure before acknowledgement
- Network duplicates

With some modifications, Raft can provide Linearizability: “instantaneously, exactly once”

- Identify duplicate commands
  - give clients session IDs and command IDs
  - Manage session (addition, expiration, stale, etc.)
Raft: Client Interaction - 2/3

Read-only command optimizations

Can anyone serve? No, as all servers are not in the same state

Can only-leader server? No, it might have lost its leader status
  ● A leader voluntarily steps down if it cannot maintain heart-beats

Combination of the two? Yes, but without complete log replication protocol
  ● Identify committed entries after re-election with no-op command
  ● Ensure your leader status with heart-beats and reply
  ● Alternatively, “sequential-consistency” can be provided as well
Raft Cluster Membership Changes - 1/2

Avoid complicated multi-server changes

Series of ±1 server changes

- Maintain a majority overlap between $C_{\text{old}}$ and $C_{\text{new}}$ configurations

Servers act immediately on a new configuration as soon as it is added to the log

Next round of changes are allowed as soon as the current one is committed

two distinct majorities, when moving from a 3 to 5 nodes cluster
Raft Cluster Membership Changes - 2/2

Servers

- Give votes without checking the current configuration
- Accept entries without checking the current configuration

Optimizations/Concerns/Recommendations

- A new server is caught up as a non-voting member
  - The leader can abort the change if the new server is slow or non-responsive
- Add servers before removing to maintain fault-tolerance level
- Bootstrap the first server with a configuration with itself in it
What are the Other Options?

Leader-based protocols: Viewstamped Replication, ZAB (ZooKeeper), etc.

Process of the election

- Detection of a failed leader (how?, practical recommendations?)
- Who can become leader (any or specific server(s)?)

Performance considerations

- Rotating leaders (Mencius)
- Offloading to clients (Fast-Paxos)
- Exploiting commutativity (Generalized-Paxos, EPaxos)