Distributed consensus, replicated state machines and... a Raft!?
Summary

“Laying the groundwork”: Consensus; CAP theorem; Failures semantics.

Raft: Motivation; Assumptions; Overview; Leadership election; Log safety; Fault-tolerance; (Lots of) Examples

Recent work: Byzantine fault-tolerance; Asymmetric partitions; Linearizability proof (Coq - Verdi) etc...
Distributed consensus?

Getting a set of processes to agree on a single data value.

T. V. I. A.

Example:

- A national election: “Who are we going to elect president?”

- Processes are servers; database replica on each servers (=nodes)
CAP Theorem

In the event of a network partition, which property do you want to keep without sacrificing latency?

Consistency: All clients see the same data even if requested concurrently.

Availability: All client’s requests to non-failing nodes must result in a response.
Consistency?

Many different consistency models: strict, atomic, causal, eventual, strong, weak etc...

In the case of Raft, we are using “atomic consistency” as our CM.

For more details, refer to [Tanen]
Failures semantics

How are nodes (= processes) in our cluster allowed to fail?
Failures semantics

Fail-stop: a process fails by stopping without warning.
Example: power outage, kernel panic etc...

Byzantine: a process fails by deviating from its expected behavior, and/or exhibiting different behavior for different observers.
Example: “traitorous” Byzantine general, defect on telemetric hardware etc...
Raft: In Search of an Understandable distributed consensus algorithm.

Dr Diego Ongaro, and Professor John Ousterhout
Stanford University (2014)
Distributed consensus algorithms

*The Part-Time Parliament* - Leslie Lamport *(Paxos)*

*Viewstamped replication* - B. Oki, Barbara Liskov *(Influenced Raft)*

*Unreliable failure detectors for reliable distributed systems* - T. Chandra, S. Toueg *(Chandra-Touueg)*
Motivation

“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”

- Chubby authors

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

- NSDI reviewer

See [1:RaFT]
Paxos made simple - L. Lamport

Paxos made moderately complex - R. Van Renesse, D. Altinzbuken

Paxos made practical - D. Mazieres

Paxos made transparent - H. Cui et al.

Paxos made live - T. Chandra, R. Griesmert, J. Redstone

Paxos made fun - A. Ounn (wip)
Assumptions

- The cluster works in an asynchronous fashion (no upper bounds for message delays)

- The network is unreliable: partitions, duplication, reordering can happen (will happen).

- Nodes fail by stopping (i.e. no Byzantine fault-tolerance).
Assumptions

- It is the client’s responsibility to communicate with the leader.

- Nodes have access to infinite persistent storage; no corruptions; write-ahead logging.

See [3: ARC RaFT]
- Reduction of the state space
- Detailed specifications (RPCs etc..)
- Lots of existing implementations (check out mine!)
We want to have a high-degree of replication.

We do not want to return obsolete/stale data.

This is a **coordination problem** - how to manage Rs/Ws and guarantee atomic consistency?
Raft: Overview

Leader election

Log replication

Safety
Leader Election

Randomized timers

Heartbeats to detect crashes/reset timers

Majority of nodes
The Leader Election happens using the RequestVote RPC.

To become a Leader, a node has to receive a majority of votes: \( \lceil N/2 + 1 \rceil \) where \( N \) is the number of nodes in our cluster.

Split votes are handled through nodes’ timers. If an election timeout, it restarts.
Candidate

initial state - $S_i$

lose an election

timer timeout

Follower

discover Leader with a higher-term

Leader

wins an election

election timeout
Log replication

The cluster receives a “command” from a client. Somehow (Assumption) the query reaches the Leader who:

- appends the “command” to its log
- replicates the appended entry to the rest of the cluster
Log replication: fixing inconsistencies

Using RaftScope
Safety

Using RaftScope
Safety

1: "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"

2: broadcastTime ≪ electionTimeout ≪ MTBF
Recap:
1. Elects a leader
2. Handle client queries
3. Commit log entry when the Leader has committed
4. Return response to the client
5. Rinse, and repeat!
More!

Need for Byzantine fault-tolerance?

Asymmetric partitions? Geographically distributed datacenters?
[Unanimous] Unanimous: In Pursuit of Consensus at the Internet Edge - H. Howard
[Raft-Dev] - Discussion about asymmetric partitions

**Proof of Raft’s Linearizability in Coq (using Verdi):**
[Verdi] + [VerdiRaft] - https://github.com/uwplse/verdi/pull/16 J. Wilcox - D. Woos

**Misc:**
[FLP] - Impossibility of Distributed consensus with One faulty process - M. Fischer, N. Lynch, M. Paterson
References

[1:RaFT] - “In Search of an Understandable consensus algorithm” - D.Ongaro, J.Ousterhout (Stanford University)


[5:CouchDB] - CouchDB Guide 1.0.1 (slide 37)

[6:RaFTTalk] - Raft case study - Professor J. Ousterhout

[Tanen] - “Distributed systems: Principles and Paradigms” A. Tanenbaum

[Tangaroa] - BFTRaft - C.Copeland, H.Zhong

[Unanimous] - In Pursuit of Consensus at the Internet Edge - H. Howard

[Raft-DEV] - Discussion about asymmetric partitions

[Verdi] - "Verdi: A Framework for Implementing and Formally Verifying Distributed Systems"