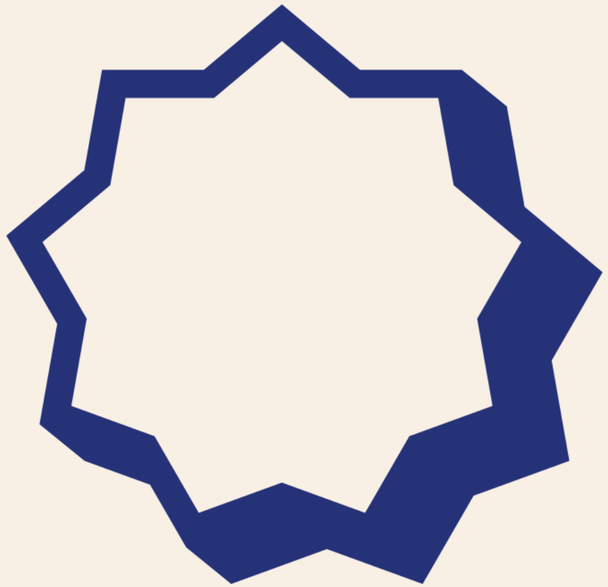
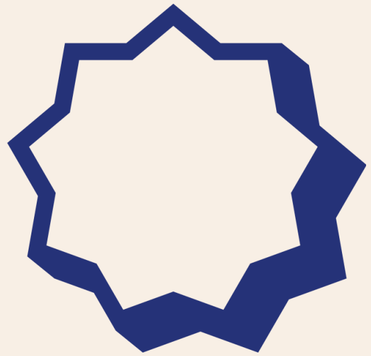


JIA KHOT

LALIT MAURYA

Reaching a consensus: Comparing RAFT and ZAB

REFERENCES



Raft

In Search of an Understandable Consensus Algorithm

ZAB

Zab: High-performance broadcast for primary-backup systems

- https://zookeeper.apache.org
- https://github.com/etcd-io/raft
- https://distributedalgorithm.wordpress.com/2015/06/20/architecture-of-zab-zookeeper-atomic-broadcast-protocol
- https://cwiki.apache.org/confluence/display/ZOOKEEPER/Zab1.0

CONSENSUS ALGORITHMS

- Consensus = agreement among distributed nodes on a single value or sequence of actions
- Ensures the system behaves like one reliable machine despite crashes or message delays
- Critical for replicated databases, config stores, and distributed coordination
- Guarantees two key properties:

    Safety: no two nodes ever decide different values

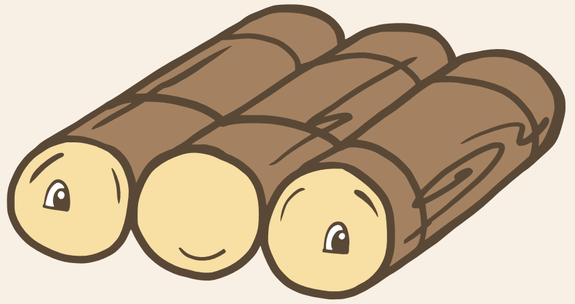
    Liveness: the system eventually reaches a decision
- Achieved through majority voting (quorums) to maintain overlap between decisions
- Used by consensus algorithms like Paxos and Raft to replicate logs consistently across servers

- Leader-based consensus designed for clarity and implementability
- **Purpose**: replicate a log consistently across servers, even with failures
- **Server states**: Leader, Follower, Candidate
- **Key components**:

  Leader Election: random timeouts → candidate → majority votes → leader

  Log Replication: leader appends entries → replicates to followers → commit on majority

  Safety: no conflicting commits; leader contains all committed entries



RAFT

RAFT: STEP BY STEP

1. System Setup

- Cluster of servers; each can be a ***Follower***, ***Candidate***, or ***Leader***
- Time is divided into ***terms***; each term begins with a new election
- Every server stores: ***current term***, ***votedFor***, and a ***replicated log***

RAFT: STEP BY STEP

2. Normal Operation (Follower)

- Followers stay passive and respond to ***RPCs*** from others
- Expect ***heartbeats*** (***AppendEntries***) from the leader
- If no heartbeat within election timeout → become ***Candidate***

RAFT: STEP BY STEP

3. Leader Election

- Candidate increments its term and votes for itself
- Sends ***RequestVote*** RPCs to all other servers
- Servers grant vote if candidate’s log is at least as up-to-date as theirs
- If candidate gets majority votes, it becomes ***Leader***
- If election times out (split vote), starts a new term with a randomized timeout

RAFT: STEP BY STEP

4. Log Replication (Leader)

- ***Leader*** receives client commands and appends them to its ***log***
- Sends ***AppendEntries*** RPCs to followers to replicate entries
- ***Followers*** append new entries to their logs if they match the leader’s log
- Once an entry is stored on a majority of servers, the leader ***commits*** it
- The leader and followers apply committed entries to their ***state*** **machines**

RAFT: STEP BY STEP

5. Maintaining Consistency

- ***Log Matching Property:*** If two logs share the same index and term, all preceding entries are identical
- ***Conflict Resolution:*** If follower’s log conflicts, it deletes mismatched entries and appends the leader’s version
- ***Leader Completeness:*** Leader’s log always contains all committed entries from previous terms

RAFT: STEP BY STEP

6. Heartbeats and Timeouts

- Leader sends periodic ***heartbeats*** (empty ***AppendEntries***) to maintain authority
- If followers stop receiving heartbeats → they trigger a new ***election***
- Randomized ***timeouts*** reduce risk of simultaneous elections

RAFT: STEP BY STEP

7. Safety and Recovery

- Only entries replicated on a ***majority*** are considered committed
- Persistent storage ensures state survives crashes
- If a leader fails, a new leader is elected, preserving all committed entries

- **ZAB (ZooKeeper Atomic Broadcast)** ensures a consistent, totally ordered broadcast of updates across all ZooKeeper servers
- Ensures all replicas apply the **same sequence of state changes**, even after crashes or restarts
- Operates in two main modes:

    Recovery Phase: elect leader and sync followers after failure

    Broadcast Phase: leader commits updates in total order

Used by **Apache ZooKeeper** for maintaining consistent metadata and coordination in distributed systems

ZAB

ZAB: STEP BY STEP

1. Purpose and Role

- ZAB ensures **total order broadcast** of updates across a ZooKeeper ensemble.
- Guarantees all servers deliver the **same sequence of transactions**, even after crashes or restarts.
- Focuses on **broadcasting state changes** reliably, not arbitrary commands.

ZAB: STEP BY STEP

2. System Model

- One **Leader**, multiple **Followers (and Observers)**
- Leader handles all **write requests**; followers serve **read requests**.
- Uses **epochs (or “Zxid” prefixes)** to order operations uniquely

ZAB: STEP BY STEP

3. Protocol Phases

- **Recovery Phase:** establish a new leader and synchronize state after a failure.
- **Broadcast Phase:** normal operation — broadcast and deliver updates in order.

ZAB: STEP BY STEP

4. Broadcast Phase (Atomic Broadcast)

- Only entries replicated on a ***majority*** are considered committed
- Persistent storage ensures state survives crashes
- If a leader fails, a new leader is elected, preserving all committed entries

ZAB: STEP BY STEP

5. Recovery Phase (Leader Election & Synchronization)

- Followers detect leader failure (via heartbeat timeout).
- Enter *election mode* and vote for a new leader (based on latest history).
- Once a leader is chosen:
    - Leader starts a new epoch and collects followers' last Zxids to find the latest state.
    - Leader broadcasts a NEWLEADER message and becomes active after majority acknowledgment.

ZAB: STEP BY STEP

6. Consistency and Guarantees

- **Total order:** All servers deliver transactions in the same sequence.
- **Prefix consistency:** No server ever commits an unacknowledged transaction.
- **Leader stability:** A leader never commits a transaction from a previous epoch.

ZAB: STEP BY STEP

7. Handling Failures

- If leader crashes → return to **Recovery Phase**.
- Uncommitted transactions are discarded to maintain global order.
- New leader re-synchronizes followers before broadcasting again.

TECH STACK

Rust

- Memory safety guarantees make it a better pick than C/C++

- Lack of a garbage collector makes it a better pick than Go

- Personal Curiosity!

RPC (gRPC)

- MPI targets tightly-coupled HPC workloads

- RPC/TCP mirrors real-world distributed systems

- Production-ready library support

Cluster Configuration

- **Base Cluster:** 5 nodes (3 for quorum, 2 spares) [Extended to 7 and 9 nodes]
- **Workloads Tested**: Both read/write heavy and mixed operation workloads
- **Fault Inject scenarios**:

    Leader Crashes

    Follower Delays

System Architecture

Each node includes:

- **Consensus Engine:** Runs Raft or ZAB logic
- **Communications Layer**: Uses RPC to communicate and handle **AppendEntries/Commit** messages
- **Client Interface**
- **Persistent Log:** Durable and persistent timestamped log
- **Timer Subsystem:** Controls elections and heartbeats