UNIVERSITÉ Concordia UNIVERSITY

EPaxos: There Is More Consensus in Egalitarian Parliaments

Yvon Ryan Wetie Mougang 40129382

Department of Electrical and Computer Engineering

February 19, 2025

Contents

Background and Motivation

- Traditional consensus protocols (e.g., Paxos [1, 2]) rely on a stable leader.
- Leader election creates bottlenecks, increases latency, and harms availability.
- In geo-replicated systems, a single master leads to extra round trips and load imbalance.
- **EPaxos** [4] removes the leader, enabling all replicas to participate equally.

Problem Statement

- How can consensus be achieved without a single point of failure or performance bottleneck?
- How can low-latency commits and robust fault tolerance be ensured under realistic, heterogeneous network conditions?

EPaxos Overview

- **Leaderless Consensus:** Any replica may act as a command leader.
- **Dynamic Ordering:** Commands are tagged with dependency lists and sequence numbers.
- **Flexible Quorums:** Fast-path commits require only a simple majority; slow-path (conflict resolution) uses a classic quorum.
- **Key Goals:**
  - Optimal commit latency (often one round-trip in the common case)
  - Uniform load balancing across replicas
  - Graceful performance degradation under failures

Key Contributions

- Eliminates the leader bottleneck by allowing concurrent proposals.
- Orders commands dynamically using local dependency information.
- Achieves optimal commit latency in wide-area networks, with only a simple majority required.
- Provides strong safety and liveness guarantees (see later slides).

References: [1, 2, 4]

Contents

Design Overview

- **Decentralized Proposals:** Clients send commands to any replica.
- **Dynamic Ordering:** Each command is assigned:
  - A dependency set (`deps`) of potentially interfering commands.
  - A sequence number (`seq`) used to break cycles.
- **Commit Mechanism:** Fast path (one round-trip) if responses are consistent; otherwise, a slow path (Paxos-Accept phase) is used.

Decentralized Command Proposals

- Clients send `Request(command)` to any replica.
- The receiving replica becomes the *command leader* for that command.
- It selects the next available instance from its log and attaches initial ordering attributes.
- A `PreAccept` message is then broadcast to a fast quorum.

Dynamic Ordering and Commit Protocol

- **Phase 1 (Fast Path):**
  - Command leader sends `PreAccept(command, seq, deps)`.
  - If a fast quorum responds with identical attributes, the command is committed.
- **Slow Path:**
  - If responses differ (due to interference), the leader invokes the Paxos-Accept phase.
  - This adds one extra communication round.
- **Optimized Fast Quorum:** In the fully optimized EPaxos, the fast-path quorum is as small as

$$\left\lceil \frac{F+1}{2} \right\rceil + F,$$

where $N = 2F + 1$ is the total number of replicas.

Commit Protocol Overview

Phase 1 (PreAccept)

- **Command Leader Actions:**
  - Increment instance number.
  - Compute initial `seq` and `deps` based on the local log.
  - Send `PreAccept` to a fast quorum.
- **Replica Actions:**
  - Update `seq` and `deps` from their log.
  - Reply with `PreAcceptOK`.

Fast vs. Slow Path

- **Fast Path:** If all fast quorum replies match, commit immediately.
- **Slow Path:** Otherwise, run the Paxos-Accept phase to resolve conflicts.

Execution Algorithm Overview

- After commit, commands are executed based on their dependency graphs.
- **Key Steps:**
  1. Build the dependency graph: Each command points to commands it depends on.
  2. Compute strongly connected components (SCCs).
  3. Topologically sort the SCCs.
  4. Execute commands in sorted order, using `seq` numbers to break ties.
- This ensures **execution consistency** (all replicas execute interfering commands in the same order).

Protocol Guarantees

EPaxos offers the following formal guarantees [4]:

- **Nontriviality:** Every committed command must have been proposed by a client.
- **Stability:** Once a command is committed at an instance, it remains so.
- **Consistency:** No two replicas can commit different commands for the same instance.
- **Execution Consistency:** Interfering commands, if committed, are executed in the same order at every replica.
- **Execution Linearizability:** If clients serialize interfering commands (i.e., propose one only after the previous is committed), every replica executes them in that order.
- **Liveness:** As long as fewer than half the replicas are faulty and messages eventually arrive, every command is eventually committed.

Minimum Working Nodes Theorem

Theorem

To tolerate F faults, the system requires at least $N = 2F + 1$ replicas.

- A majority (i.e., $F + 1$) is needed for overlapping quorums.
- This ensures that any two quorums share at least one correct replica to detect conflicts [1, 3].

EPaxos Message Flow

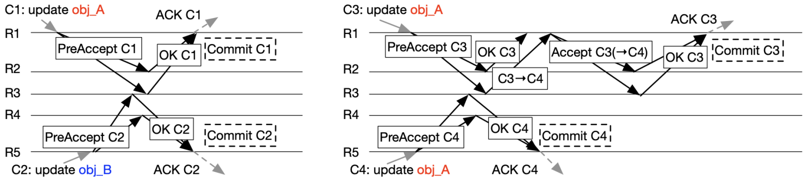

Figure 1: EPaxos message flow. R1, R2, . . . , R5 are the five replicas. Commands C1 and C2 (left) do not interfere, so both can commit on the fast path. C3 and C4 (right) interfere, so one (C3) will be committed on the slow path. C3->C4 signifies that C3 acquired a dependency on C4. For clarity, we omit the async commit messages.

Comparison with Related Work

Feature	**Paxos**	**Mencius**	**Gen. Paxos**	**EPaxos**
Leader Dependency	High	Moderate	Stable Leader	None
Latency (Rounds)	2 RTTs	2 RTTs	2 RTTs+	1 RTT (fast path)
Quorum Size	$F+1$	$F+1$	N	$\lceil \frac{F+1}{2} \rceil + F$
Load Balancing	Poor	Improved	Limited	Excellent
Dynamic Ordering	No	No	Partial	Yes

References: [1, 5, 6, 4]

Recovery and Fault Tolerance

- When a replica times out waiting for a commit, it runs an **Explicit Prepare** phase.
- Recovery gathers the most recent `PreAccept` state from a majority.
- The recovering replica then either finalizes the commit (via Paxos-Accept) or, if no command is found, commits a no-op.
- This mechanism ensures liveness even under failures [4].

Contents

Experimental Setup

- **Testbed:** Amazon EC2 with replicas in California, Virginia, Ireland, Oregon, and Japan.
- **Workloads:** Replicated key-value store with varying command sizes and conflict rates.
- **Comparisons:** EPaxos is evaluated against Multi-Paxos, Mencius, and Generalized Paxos [5, 6].

Results and Findings

- **Latency:** EPaxos often commits commands in one round-trip (fast path), even under interference.
- **Throughput:** Load is balanced across replicas; EPaxos processes fewer messages per command (especially with batching and thrifty optimizations).
- **Fault Tolerance:** The system continues to make progress as long as a majority of replicas are live.

Throughput Results

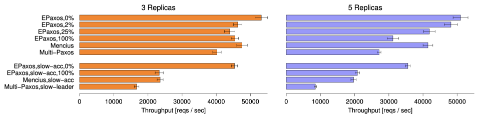

Figure 2: Throughput for small (16 B) commands (95% CI).

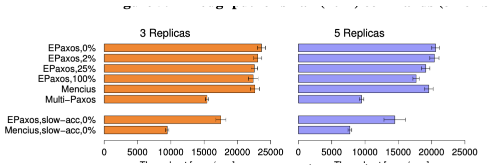

Figure 3: Throughput for large (1 KB) commands (95% CI).

Reference: [4]

Latency Results

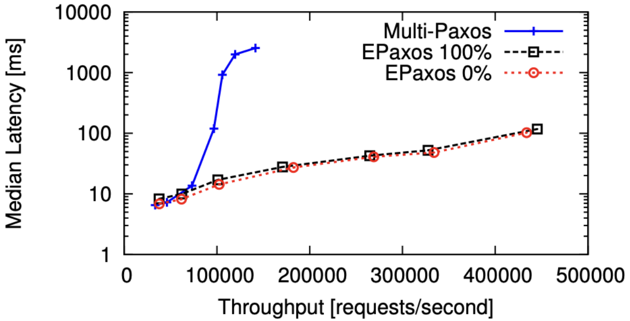

Figure 4 : Throughput vs. Latency.

Reference: [4]

Fault Tolerance Results

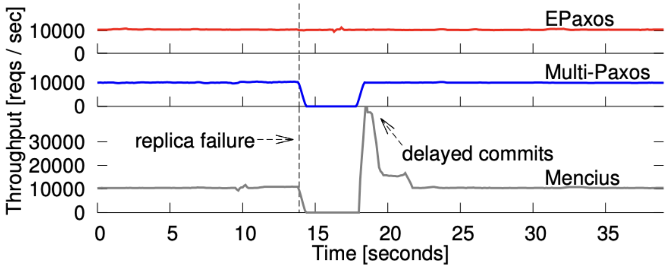

Figure 5: Commit throughput when one of three replicas fails (Multi-Paxos leader fails).

Reference: [4]

Contents

Strengths of EPaxos

- **Decentralized Operation:** No single leader; reduces bottlenecks and improves availability.
- **Dynamic Ordering:** Efficient handling of non-interfering commands allows fast-path commits.
- **Fault Tolerance:** Robust under failures; recovery protocols ensure progress with a simple majority.
- **Load Balancing:** Even distribution of consensus load across replicas.

Weaknesses and Limitations

- **Protocol Complexity:** The dynamic ordering and dependency management add algorithmic overhead.
- **Extra Round-Trip:** In the presence of conflicts, an extra round-trip is required.
- **GC and Message Overhead:** Larger messages (with dependency attributes) may incur higher garbage collection and network costs.
- **Recovery Nuances:** Optimized fast-path quorum sizes introduce subtle recovery challenges.

Critical Analysis Summary

- EPaxos represents a significant evolution of Paxos-based consensus by eliminating the leader.
- It achieves lower latency and higher throughput in wide-area settings.
- However, these gains come at the cost of increased protocol complexity and careful handling of dependencies.
- Future work could focus on further optimizations and simplifying recovery.

Contents

Anticipated Questions

- How does EPaxos reconcile commands during network partitions?
- What is the impact of dynamic ordering on system overhead?
- Under what conditions might leader-based protocols be preferable?
- How do the theoretical guarantees translate to real-world performance?

Prepared Answers

- **Reconciliation:** The explicit prepare phase ensures that a recovering replica learns the latest committed command.
- **Overhead vs. Latency:** Although managing dependencies adds overhead, it reduces communication rounds in the common case.
- **Leader-Based Use Cases:** In highly homogeneous environments with low conflict rates, the simpler leader-based approach may suffice.
- **Real-World Performance:** Experimental results on EC2 demonstrate that EPaxos achieves lower commit latency and better throughput under realistic conditions [4].

Contents

Conclusion

- EPaxos eliminates the leader bottleneck by enabling every replica to propose commands.
- Its dynamic ordering and flexible quorum selection lead to optimal commit latency in wide-area settings.
- Although the design introduces additional complexity, the performance and fault-tolerance benefits are significant.
- EPaxos sets a new standard for distributed consensus in heterogeneous and geo-replicated environments.

Questions & Discussion

Questions?

References

- L. Lamport. *The part-time parliament.* ACM Trans. Comput. Syst., 1998.
- L. Lamport. *Paxos made simple.* SIGACT News, 2001.
- M. J. Fischer, N. A. Lynch, and M. S. Paterson. *Impossibility of distributed consensus with one faulty process.* J. ACM, 1985.
- I. Moraru, D. G. Andersen, and M. Kaminsky. *A proof of correctness for Egalitarian Paxos.* CMU-PDL-13-111, 2013.
- Y. Mao, F. P. Junqueira, and K. Marzullo. *Mencius: building efficient replicated state machines for WANs.* In Proc. 8th USENIX OSDI, 2008.
- L. Lamport. *Generalized consensus and Paxos.* 2005.
- L. Lamport. *Fast Paxos.* 2006.