ISSN - 2456-1142

International journal of humanities and information technology Volume 8, Issue 1, 2026

Deterministic Consistency Models for Cloud-Hosted Open Source Relational Databases Under Multi-Region Failover

Vinodkrishna Gopalan*

The Ohio State University, USA

Abstract

Cloud-hosted open source relational databases are increasingly deployed across multiple geographic regions to meet high availability and low latency requirements. However, multi-region failover remains a critical source of consistency degradation. During regional outages or leader transitions, asynchronous replication delays, quorum reconfiguration, and log reconciliation can introduce transient divergence, write reordering, and recovery latency spikes. Traditional leader-based replication ensures strong consistency under stable conditions but often incurs failover delays due to election and log synchronization overhead. Quorum-based and eventually consistent approaches improve availability but tolerate temporary inconsistency and non-deterministic commit ordering. Deterministic transaction scheduling has shown promise in improving serializability and coordination efficiency, yet it is rarely integrated directly into cloud failover control mechanisms for open source SQL engines.

This paper proposes a deterministic consistency framework that integrates global pre-transaction sequencing with consensus-assisted failover coordination. The model enforces a uniform commit order across regions before execution, ensuring stable state transitions even during leader promotion or regional recovery. A prototype implementation on a cloud-deployed relational engine demonstrates reduced failover recovery time, elimination of write reordering events, and competitive throughput compared to conventional consensus-based replication. Experimental evaluation under simulated wide-area latency confirms improved consistency stability without significant performance degradation.

The proposed framework bridges the gap between deterministic transaction processing and cloud multi-region failover design, offering a practical pathway toward resilient, serializable, and globally consistent open source relational databases.

Keywords: Deterministic consistency, Multi-region failover, Geo-replicated databases, Distributed consensus, Serializable transactions, Cloud SQL, Replication protocols.

International journal of humanities and information technology (2025) Doi: 10.21590/ijhit.08.01.05

Introduction

Background

Rise of Geo-Replicated Cloud SQL Systems

The last decade has witnessed a rapid transformation in relational database deployment models, shifting from single-region data centers to globally distributed cloud architectures. Modern enterprises demand low latency access, regulatory data locality compliance, and continuous service availability across continents. As a result, geo-replicated relational database systems have emerged as a foundational component of cloud infrastructure.

Early distributed database research established the theoretical and operational basis for reliable replication and fault tolerance. Primary-copy replication protocols such as Viewstamped Replication (Oki & Liskov, 1988; Liskov & Cowling, 2012) introduced mechanisms for maintaining consistency under failures. Consensus-based algorithms, including Paxos (Lamport, 1998, 2001) and later refinements grounded in the Part-Time Parliament model (Lamport, 1998), provided provably safe state machine replication.

Building on these foundations, globally distributed SQL engines such as Spanner (Corbett et al., 2013), F1 (Shute et al., 2013), and Megastore (Baker et al., 2011) demonstrated

Corresponding Author: Vinodkrishna Gopalan, The Ohio State University, USA, e-mail: gopalanv@gmail.com

How to cite this article: Gopalan, V. (2026). Deterministic Consistency Models for Cloud-Hosted Open Source Relational Databases Under Multi-Region Failover. *International journal of humanities and information technology* 8(1), 46-58.

Source of support: Nil

Conflict of interest: None

that relational semantics and geo-distribution can coexist at planetary scale. Deterministic scheduling approaches such as Calvin (Thomson et al., 2012) and later geo-replicated transactional systems such as Slog (Ren et al., 2019) further advanced the ability to execute serializable transactions across regions with reduced coordination overhead.

Cloud-hosted open source relational engines, particularly PostgreSQL and MySQL derivatives, are increasingly deployed in multi-region topologies. These deployments combine write-ahead logging replication, quorum-based commit strategies, and leader-election mechanisms inspired by Paxos and Raft (Ongaro & Ousterhout, 2014). However, while these systems deliver high availability and strong consistency under stable network conditions, failover behavior remains a critical challenge.

Multi-Region Deployment for High Availability

Multi-region architectures are designed to withstand regional outages, network partitions, and catastrophic infrastructure failures. Data is replicated across geographically separated regions, typically using one of the following patterns:

- Leader-based replication with synchronous or semi-synchronous followers
- Multi-leader replication with conflict resolution
- Deterministic transaction ordering across partitions
- Quorum-based commit protocols

Systems such as Dynamo (DeCandia et al., 2007) favored availability by adopting eventual consistency and tunable quorums. In contrast, systems such as Spanner (Corbett et al., 2013) used tightly synchronized clocks and consensus to provide external consistency (a stronger property than linearizability that ensures transactions appear to execute in real-time order consistent with their commit timestamps). More recent work on scalable atomic visibility and highly available transactions has explored trade-offs between coordination cost and isolation guarantees (Bailis et al., 2013; Bailis et al., 2016).

Despite these advances, multi-region failover introduces transient states during leader election or region promotion in which in-flight transactions may be reordered, retried, or partially committed. These windows of inconsistency are often small but non-negligible, especially under asynchronous replication or WAN latency fluctuations.

Tension Between Availability and Strict Consistency

The theoretical limits of distributed systems impose fundamental constraints on what multi-region databases can guarantee. The CAP theorem, formalized by Gilbert and Lynch (2002) based on Brewer's conjecture, demonstrates that during network partitions, a system must choose between consistency (specifically, linearizability) and availability. In geo-distributed environments where partitions are unavoidable, strict consistency often incurs coordination delays that affect availability.

Furthermore, the FLP impossibility result (Fischer et al., 1985) proves that consensus cannot be guaranteed to terminate in a fully asynchronous system with even a single faulty process. This result underpins the need for partial synchrony assumptions (eventually stable message delays) or failure detectors in practical consensus protocols such as Paxos (Lamport, 1998, 2001) and Raft (Ongaro & Ousterhout, 2014).

These theoretical constraints imply that multi-region relational systems inevitably face trade-offs between failover speed, transaction ordering, and consistency guarantees. While strong consistency protocols ensure safety, they often introduce latency penalties and complexity during region failover. Conversely, asynchronous replication models improve availability but risk transient anomalies.

The central challenge lies in reconciling deterministic transaction semantics with practical failover requirements in cloud-hosted open source relational databases.

Research Problem

Failover-Induced Non-Determinism

In leader-based replication systems, failover occurs when the primary region becomes unavailable and a replica is promoted. During this transition, transactions that were committed at the primary but not yet replicated to a quorum may be lost or create ambiguity regarding commit order. Even when write-ahead logs are replayed deterministically, network delays can lead to non-deterministic commit visibility across regions.

Consensus protocols maintain safety by ensuring that only a quorum-approved log entry is committed. However, the process of electing a new leader can introduce timing-based variability in transaction ordering. Under high concurrency, this variability may affect serializability boundaries and read visibility.

Write Reordering Under Asynchronous Replication

Asynchronous or semi-synchronous replication allows the primary to acknowledge writes before full cross-region propagation. While this improves performance, it creates the possibility that concurrent writes may be observed in different orders across regions during failover. Systems that rely on last-writer-wins conflict resolution or causal tracking attempt to mitigate these issues (Lloyd et al., 2011; Burckhardt, 2014), yet strict serializability is not always preserved without coordination.

Even systems that guarantee atomic visibility through mechanisms such as RAMP transactions (Bailis et al., 2016) face coordination overhead during partitions. Under failover conditions, replay-based recovery can expose subtle reordering events, especially when deterministic scheduling is absent.

Divergent State Across Regions

Transient divergence may arise when:

- A region fails mid-commit
- Log entries are partially propagated
- Leader election overlaps with client retries
- Concurrent cross-region writes are reordered during recovery

While most consensus-based systems ensure eventual convergence, the temporary divergence window can violate application-level invariants. In financial, healthcare, or compliance-sensitive workloads, even short-lived anomalies are unacceptable.

The problem addressed in this research is therefore:

How can deterministic transaction ordering be incorporated into cloud-hosted open source relational databases to eliminate failover-induced non-determinism without incurring prohibitive latency penalties?

Research Questions

This study investigates the feasibility and performance implications of deterministic consistency models under multi-region failover. The following research questions guide the work:

- Can deterministic global transaction ordering eliminate failover inconsistency windows in geo-replicated SQL systems?
- How does deterministic commit compare to consensus-based commit under network partition scenarios governed by CAP constraints (Gilbert & Lynch, 2002) and FLP limitations (Fischer et al., 1985)?
- What is the latency and throughput impact of introducing deterministic sequencing into cloud-hosted open source relational engines relative to classical Paxos and Raft-based designs?

These questions examine both theoretical correctness and empirical performance trade-offs.

Contributions

This paper makes four primary contributions.

Formal Deterministic Consistency Model

- We introduce a formally defined deterministic consistency model for multi-region relational databases. The model specifies:
- Global transaction ordering invariants
- Deterministic scheduling semantics
- Failover safety properties
- State machine replication guarantees

The formulation draws upon consensus theory (Lamport, 1998, 2001; Liskov & Cowling, 2012) and deterministic execution paradigms (Thomson et al., 2012), while addressing the limitations of asynchronous replication identified in prior work.

Hybrid Deterministic-Consensus Failover Design

We propose a hybrid architecture that combines:

- Deterministic pre-ordering of transactions
- Consensus-backed durability
- Failover-safe region promotion rules

Unlike purely leader-based systems, this design ensures that transaction order is globally agreed before execution, eliminating ambiguity during leader transition.

Comparative Evaluation Across Classical and Modern Systems

The proposed model is evaluated against representative distributed database paradigms including:

- Paxos-based replication (Lamport, 1998, 2001)
- Raft-based coordination (Ongaro & Ousterhout, 2014)
- Deterministic scheduling systems such as Calvin (Thomson et al., 2012)
- Geo-replicated SQL systems such as Spanner (Corbett et al., 2013) and Slog (Ren et al., 2019)
- Highly available transactional models (Bailis et al., 2013; Bailis et al., 2016)

Metrics include failover latency, transaction throughput, divergence events, and recovery determinism.

Practical Integration Path for Open Source SQL Engines

Finally, we demonstrate how deterministic ordering layers can be integrated into cloud-hosted open source relational engines without requiring modifications to core storage engines. The approach leverages write-ahead logging hooks and external sequencing services, making it feasible for production cloud environments.

Theoretical Foundations

Deterministic consistency under multi-region failover builds upon four decades of research in distributed systems theory, consensus protocols, transaction processing, and geo-replicated database design. This section synthesizes foundational principles that inform deterministic ordering, fault tolerance, and consistency preservation in cloud-hosted open source relational databases.

Logical Time and Event Ordering

Correct ordering of events in a distributed system is a prerequisite for deterministic execution. In geographically dispersed deployments, physical clocks cannot be assumed to be perfectly synchronized. Logical time therefore becomes the primary abstraction for preserving causality and ensuring serializability.

Lamport Clocks

Leslie Lamport (1978) formalized the concept of logical clocks to impose a partial ordering on events in distributed systems. Lamport demonstrated that if event A causally

precedes event B, then the logical timestamp of A must be less than that of B. The logical clock algorithm increments a local counter for each event and piggybacks timestamps on messages exchanged between processes. Upon receiving a message, a process updates its clock to the maximum of its current value and the received timestamp, plus one.

This mechanism establishes the "happened-before" relation, enabling deterministic reasoning about distributed state transitions even in the absence of synchronized physical clocks. Although Lamport clocks do not provide total ordering by default, they can be extended with tie-breaking mechanisms, such as process identifiers, to create a deterministic total order.

In the context of multi-region relational databases, logical clocks allow deterministic transaction scheduling prior to commit. Instead of relying solely on leader election or quorum commit timing, transactions can be assigned deterministic sequence numbers derived from logical timestamps. This ensures consistent replay during failover.

The Part-Time Parliament

In "The Part-Time Parliament," Lamport (1998) introduced the Paxos consensus framework through a parliamentary analogy. The paper formalizes how distributed participants, even in the presence of failures and message delays, can agree on a single sequence of values.

The importance of this work lies in its demonstration that safety can be preserved under asynchronous communication and crash failures. The algorithm ensures that once a value is chosen, it remains chosen, regardless of subsequent failures. The underlying safety proof provides the theoretical foundation for leader-based replication systems and deterministic state machine replication.

For deterministic consistency models, the insight from the Part-Time Parliament is that agreement on order must precede execution. Failover should not alter the agreed sequence of operations. Deterministic models extend this principle by separating ordering from execution to reduce failover nondeterminism.

Consensus and Replication

Consensus algorithms enable a distributed system to agree on a single log of operations. Replication protocols build on consensus to maintain fault-tolerant state machines.

Paxos

Lamport (2001) presented Paxos as a simplified exposition of the consensus protocol. Paxos guarantees safety under asynchronous communication with crash failures, consistent with the impossibility boundaries proven by Fischer, Lynch, and Paterson (1985).

In Paxos, proposers suggest values, acceptors vote on proposals, and learners observe chosen values. A majority quorum intersection ensures that once a value is accepted by a quorum, it cannot be overridden.

Paxos provides linearizability and strong consistency, but practical deployments often incur complexity and latency overhead due to multi-phase message exchanges. In multi-region failover, Paxos ensures state agreement but does not inherently enforce deterministic transaction ordering before execution. This distinction motivates deterministic pre-ordering mechanisms.

Raft

Diego Ongaro and John Ousterhout (2014) introduced Raft as an understandable consensus algorithm. Raft decomposes consensus into leader election, log replication, and safety enforcement.

A single leader handles client requests and replicates log entries to followers. Leader failover occurs through randomized election timeouts. Raft simplifies reasoning compared to Paxos and has been widely adopted in distributed storage systems.

However, during failover, new leaders may experience temporary unavailability or require log reconciliation. Deterministic consistency models attempt to reduce this window by ensuring globally predetermined transaction order independent of transient leadership transitions.

Viewstamped Replication

Viewstamped Replication was introduced by Oki and Liskov (1988) and later revisited by Liskov and Cowling (2012). It provides primary-backup replication with view changes analogous to leader election.

Each view designates a primary replica responsible for ordering operations. When a failure occurs, a new view is established with a new primary. Safety is preserved because committed operations are guaranteed to survive view transitions.

Viewstamped Replication directly informs deterministic failover models. However, similar to Raft, operation ordering is determined at runtime by the primary. Deterministic scheduling introduces pre-execution ordering to minimize view transition ambiguity.

Geo-Distributed SQL Systems

Modern geo-distributed SQL engines demonstrate different approaches to global consistency.

Spanner

Spanner (Corbett et al., 2013) achieves externally consistent transactions using synchronized clocks and TrueTime. By bounding clock uncertainty, Spanner enforces globally consistent commit timestamps.

Spanner's architecture demonstrates that strong consistency is achievable at planetary scale. However, it relies on specialized time synchronization infrastructure. Deterministic models seek similar ordering guarantees without strict physical clock dependencies.

Calvin

Calvin (Thomson et al., 2012) introduced deterministic transaction ordering. Transactions are sequenced in a global log before execution. Each replica replays the same ordered log, ensuring serializability without distributed locking during execution.

Calvin directly inspires deterministic consistency under failover. Because ordering is predefined, execution across replicas remains consistent even after failures.

Slog

Slog (Ren et al., 2019) optimizes geo-replicated transactions for locality and low latency. By placing leaders close to transaction origin regions, Slog reduces cross-region coordination cost while maintaining serializability.

Slog highlights the trade-off between latency and strict ordering. Deterministic models must account for geographic locality while preserving global order.

Megastore

Megastore (Baker et al., 2011) extends Paxos to provide entity group-based consistency over wide-area deployments. It partitions data into smaller consensus groups to improve scalability.

This architecture shows how consensus can scale horizontally while preserving per-entity serializability.

F1

F1 (Shute et al., 2013) builds on Spanner to provide a distributed SQL layer with relational semantics. It integrates synchronous replication and strong consistency while supporting high throughput workloads.

MDCC

MDCC (Kraska et al., 2013) uses optimistic commit protocols across data centers, reducing coordination overhead compared to traditional two-phase commit.

FaRM

FaRM (Dragojević et al., 2014) leverages remote direct memory access for low-latency distributed transactions, demonstrating that hardware acceleration can reduce consensus cost.

Collectively, these systems demonstrate varying balances between deterministic ordering, consensus replication, and latency optimization.

Availability-Consistency Trade-offs

The CAP theorem formalized by Gilbert and Lynch (2002) establishes that a distributed system cannot simultaneously guarantee consistency, availability, and partition tolerance. Fischer, Lynch, and Paterson (1985) further proved that deterministic consensus is impossible in fully asynchronous systems with one faulty process.

Dynamo

Dynamo (DeCandia et al., 2007) prioritizes availability and partition tolerance, using quorum reads and writes with eventual reconciliation. Conflict resolution occurs through version vectors.

COPS

COPS (Lloyd et al., 2011) provides causal consistency across geo-distributed storage systems. It ensures that causally related updates are observed in order without requiring full serializability.

Eventual Consistency

Burckhardt (2014) formalized eventual consistency semantics and demonstrated that replicas converge in the absence of new updates. Eventual consistency sacrifices immediate global agreement for availability and scalability.

Highly Available Transactions

Bailis et al. (2013) explored highly available transactions that avoid coordination under partitions. These systems improve availability but restrict isolation guarantees.

RAMP Transactions

Bailis et al. (2016) introduced atomic visibility without full coordination, allowing scalable read-atomic transactions while avoiding global locks.

Deterministic Positioning

Deterministic consistency models aim to position themselves between strict linearizable consensus systems and relaxed eventual consistency models. By predefining transaction order while maintaining consensus for log durability, they attempt to reduce failover inconsistency without sacrificing strong correctness guarantees. Graph 1: Consistency-Availability Spectrum Across Systems, Figure 1. Consistency-Availability Spectrum Across Distributed Database Systems.

The figure positions deterministic commit models alongside Paxos and Raft under strong consistency, with Spanner and F1 under externally consistent architectures, COPS under causal consistency, and Dynamo and eventual consistency systems toward the availability-dominant region. Deterministic consistency seeks to preserve serializability and failover stability while minimizing coordination-induced downtime.

Deterministic Consistency Model

This section presents a deterministic consistency framework designed for cloud-hosted open source relational databases deployed across multiple geographic regions. The objective is to guarantee strict serializable behavior and eliminate transaction reordering during region failover. The design builds on established foundations in distributed systems

theory, including state machine replication and consensus protocols. It integrates deterministic transaction ordering techniques inspired by the work on Calvin by Thomson et al. (2012), classical Paxos-style consensus described by Lamport (2001), and view change mechanisms developed in Viewstamped Replication by Oki and Liskov (1988) and later revisited by Liskov and Cowling (2012).

The central principle of the model is the separation of transaction ordering from transaction execution. All replicas observe the same global sequence of transactions before execution begins. Because execution is deterministic and inputs are identical across replicas, state convergence is guaranteed even during failover events.

Deterministic Global Ordering Layer

Pre-Transaction Sequencing

Traditional distributed databases such as those based on Paxos or Raft derive transaction ordering from replicated logs maintained by a leader. When a leader fails, a new leader must be elected and log reconciliation may occur. Although safety is preserved, short windows of uncertainty may exist during which ordering is not yet globally stabilized.

The deterministic model removes this uncertainty by introducing a global sequencing service that assigns a unique and monotonically increasing sequence number to every transaction before execution begins. The sequencing service operates logically above the database engine. Incoming transactions are first routed to the ordering layer, where they are assigned sequence numbers and grouped into deterministic batches.

These ordered batches are then replicated across regions using a consensus protocol consistent with Paxos as described by Lamport (2001) or Viewstamped Replication as described by Liskov and Cowling (2012). Only after the sequence is durably replicated to a quorum of regions is execution permitted.

This design ensures that the transaction serialization order is globally fixed prior to execution. As a result, even if the primary region fails, all replicas share an identical view of transaction order. Unlike eventually consistent systems such as Dynamo described by DeCandia et al. (2007), this approach does not allow divergent write histories that require reconciliation.

Deterministic Transaction Scheduling

The deterministic scheduling approach follows the principles introduced in Calvin by Thomson et al. (2012). In Calvin, transactions are ordered prior to execution and executed deterministically according to that order, thereby eliminating the need for distributed locking during runtime.

In the proposed model, once transactions receive global sequence numbers, each replica executes transactions strictly in that order. Read and write sets may be pre-analyzed when possible. Conflict resolution is handled implicitly by ordering rather than through dynamic locking.

Because every replica executes the same ordered transaction stream, there is no possibility of deadlock caused by inconsistent lock acquisition order across regions. Furthermore, execution is replayable and reproducible. This behavior resembles the serializable guarantees provided by globally distributed systems such as Spanner described by Corbett et al. (2013), but without relying on tightly synchronized physical clocks.

Deterministic scheduling ensures that even during failover, execution resumes from a known sequence position. There is no ambiguity regarding which transaction should execute next.

Logical Timestamp Enforcement

To preserve causal ordering and global monotonicity, the model employs logical timestamps following the framework introduced by Lamport (1978) in his discussion of logical clocks and event ordering. Each transaction carries a logical timestamp that reflects its position in the global sequence.

Logical time ensures that ordering is preserved independently of physical clock synchronization. This is important in geographically distributed environments where clock skew can be significant.

Unlike systems that depend on bounded clock uncertainty to achieve external consistency, such as Spanner described by Corbett et al. (2013), the deterministic model derives ordering entirely from logical sequencing. Logical timestamps therefore function as a formal mechanism for maintaining total order across replicas without reliance on synchronized hardware clocks.

Deterministic Failover Protocol

The deterministic ordering layer simplifies failover handling because transaction order is already globally established before execution. Failover does not require re-negotiation of transaction serialization.

Region Promotion Rules

When a primary region becomes unavailable, a new region may be promoted through a consensus decision consistent with Paxos described by Lamport (2001) or Viewstamped Replication described by Oki and Liskov (1988).

Promotion is permitted only if the candidate region has fully replicated the latest committed deterministic sequence. Because all committed transactions have globally assigned sequence numbers, the promoted region resumes execution from the highest committed sequence number.

This prevents divergent histories that commonly arise in multi-master systems. Unlike Dynamo-style architectures described by DeCandia et al. (2007), there is no need for conflict resolution after failover.

Pre-Commit Ordering Guarantees

The system enforces a strict prefix rule. A transaction may commit only if all transactions with lower sequence numbers

have been durably replicated and acknowledged by a quorum.

This rule ensures prefix consistency across regions. All replicas share an identical committed transaction prefix at any time.

This differs from optimistic commit approaches such as MDCC described by Kraska et al. (2013), which allow transactions to commit in partially overlapping orders. In the deterministic model, commit order is identical everywhere.

No Write Reordering During Leader Transition

Leader transition in consensus protocols such as Raft described by Ongaro and Ousterhout (2014) can require log reconciliation if leaders diverge before election. In the deterministic model, sequencing decisions are externalized and globally agreed before execution.

Because transaction order is immutable once assigned, a leader change does not alter transaction sequence. Execution resumes from the last confirmed sequence number without reordering.

This eliminates lost updates, write skew, and reordering anomalies. The behavior is comparable to geo-distributed serializable systems such as Slog described by Ren et al. (2019), which also enforce strong ordering guarantees across regions.

Formal Model

State Machine Definition

The system can be modeled as a replicated deterministic state machine. Let the database state be represented by S. Let the ordered transaction sequence be represented by T.

For each replica, the state transition function is deterministic. Given identical initial state and identical transaction sequence, all replicas transition through identical intermediate states and converge to the same final state.

This formalization aligns with the state machine replication paradigm underlying Paxos described by Lamport (2001) and Viewstamped Replication described by Liskov and Cowling (2012).

Commit Invariants

The deterministic consistency model enforces three primary invariants.

First, the prefix invariant. If a replica has applied transaction number n, then it must have applied all transactions numbered less than n.

Second, the deterministic execution invariant. Given identical transaction input sequences, all replicas produce identical state transitions.

Third, the single decision invariant. Each transaction has exactly one commit decision determined by consensus.

These invariants prevent anomalies described in highly available transactional systems analyzed by Bailis et al. (2013) and ensure atomic visibility comparable to the guarantees of RAMP transactions described by Bailis et al. (2016).

Safety and Liveness

Safety requires that no two replicas commit conflicting transaction histories. The impossibility result by Fischer, Lynch, and Paterson (1985) establishes that deterministic consensus cannot guarantee termination in a purely asynchronous system with faults. However, safety can always be preserved.

In the deterministic model, ordering decisions are made through quorum-based consensus similar to Paxos described by Lamport (2001). Only one value may be chosen for each sequence number. Therefore, conflicting histories cannot occur among correct replicas.

Table 1: Comparison of Replication and Consistency Protocols

System	*Replication Mechanism*	*Consistency Model*	*Deterministic Ordering*	*Multi-Region Failover Behavior*
Paxos (Lamport, 2001)	Majority quorum consensus	Linearizable	No	Leader election and log reconciliation
Viewstamped Replication (Oki & Liskov, 1988)	Primary copy with view change	Linearizable	No	View change recovery
Raft (Ongaro & Ousterhout, 2014)	Leader-based log replication	Linearizable	No	Controlled leader replacement
Dynamo (DeCandia et al., 2007)	Quorum replication	Eventual consistency	No	Multi-master conflict resolution
Calvin (Thomson et al., 2012)	Deterministic log ordering	Serializable	Yes	Ordered replay across regions
MDCC (Kraska et al., 2013)	Optimistic quorum commit	Serializable (assumptions)	Partial	Fast cross-region commit
Slog (Ren et al., 2019)	Geo-partitioned execution	Serializable	Yes	Region-local with global ordering
Proposed Model	Deterministic + quorum consensus	Deterministic serializable	Yes	No write reordering

Liveness requires that transactions eventually commit when the system is not partitioned. As shown in analyses of Paxos and Raft, progress is guaranteed under partial synchrony when a majority of replicas are reachable and communication delays stabilize.

During network partitions, the system prioritizes consistency over availability in accordance with the results of Gilbert and Lynch (2002) regarding the feasibility of consistent, available, partition-tolerant systems. Under partition, writes may be delayed, but committed history remains consistent and globally ordered.

Architecture for Cloud-Hosted Open Source SQL Engines

Cloud-hosted open source relational databases such as PostgreSQL and MySQL were originally designed for single-region deployment with synchronous or asynchronous replication extensions. Their native replication mechanisms rely on write-ahead logging, binlog shipping, and leader-based coordination; under multi-region failover, these mechanisms can introduce non-deterministic commit ordering, replication lag, and temporary inconsistency windows.

The proposed deterministic consistency architecture extends open source SQL engines with a deterministic global ordering layer that operates above the storage engine while preserving ACID semantics. The architecture draws from deterministic transaction scheduling concepts introduced in Calvin, viewstamped replication, Paxos-style commit ordering, and multi-data-center coordination approaches such as MDCC and Slog. The design objective is to eliminate write reordering during region failover while maintaining serializability and bounded failover recovery time.

Integration with PostgreSQL and MySQL

WAL Interception

Both PostgreSQL and MySQL rely on a write-ahead log (WAL/binlog) to guarantee durability. PostgreSQL uses WAL segments streamed to replicas, while MySQL uses binary logs for replication; in conventional streaming replication, log records are shipped after commit, and effective ordering depends on leader behavior and network timing.

The deterministic model introduces a WAL interception layer positioned between the transaction manager and the physical log flush.

Operational flow:

- Transaction execution proceeds normally inside the SQL engine.
- Before commit record emission to WAL/binlog, the transaction is submitted to the deterministic commit coordinator.
- The coordinator assigns a globally agreed sequence number using a consensus-backed sequencing service.
- WAL records are annotated with this sequence number before durable persistence.
- Replicas apply transactions strictly in sequence order.

This approach differs from asynchronous replication used in Dynamo-style systems and from causal consistency enforcement in COPS, because it enforces deterministic serializability similar in spirit to Calvin's pre-ordering while still leveraging mature open source storage engines. The WAL interception mechanism does not modify the storage layout or buffer manager; it extends the commit pipeline and preserves crash-recovery semantics consistent with ARIES-style logging.

Deterministic Commit Coordinator

The deterministic commit coordinator is a logically centralized but physically replicated service responsible for global transaction ordering.

Design principles:

- Global sequence assignment before commit visibility
- Majority consensus on ordering decisions
- Deterministic replay across all regions

The coordinator uses a consensus protocol derived from Paxos or Raft; unlike traditional leader-commit models where commit ordering is decided implicitly by log append, this design separates ordering from execution.

Key properties:

- Safety under FLP bounds in partially synchronous systems (consensus may block but not violate safety)
- No reordering during leader transition because sequence numbers are immutable once chosen
- Linearizable ordering decisions for the global sequence

This mechanism is comparable to the ordering discipline in Spanner but does not require tightly synchronized physical clocks, instead relying on logical time consistent with Lamport's happened-before relation. During failover, the coordinator re-elects a leader while preserving the committed ordering prefix, preventing divergence observed in eventually consistent systems.

Cross-Region Sequencing Service

The cross-region sequencing service maintains a replicated global transaction log shared across regions.

Components:

- Consensus group spanning all regions
- Persistent ordering log
- Region-specific apply queues

Sequence generation is performed before commit acknowledgement to the client; transactions become visible only after the sequence is durable across a majority of regions. This differs from two-phase-commit optimizations, because ordering is determined before the commit decision is finalized, and improves upon MDCC by reducing cross-region conflict handling during failover. The deterministic replay mechanism ensures that all replicas apply transactions in identical order, eliminating non-deterministic anomaly windows observed in inconsistent replication systems.

Multi-Region Deployment Model

Primary Region

The primary region hosts:

- Client-facing SQL endpoints
- Local execution engines
- The current deterministic coordinator leader

Transactions are executed locally but cannot commit until global ordering is confirmed by the sequencing service. This preserves strong consistency similar to Megastore and F1 while permitting a familiar single-primary execution model to applications.

Commit acknowledgment requires:

- Local WAL/binlog durability
- Majority replication of the ordering record in the sequencing group
- Confirmation from the deterministic sequencing service that the transaction's position is fixed in the global sequence

Hot Standby Regions

Hot standby regions maintain

- Continuous streaming of ordered WAL/binlog entries
- Pre-validated transaction apply queues
- A synchronous log-prefix guarantee (only fully ordered prefixes are applied)

Unlike asynchronous replication in Dynamo-style systems, replicas do not independently resolve conflicts and only apply globally ordered commits.

Failover procedure

- Coordinator leader failure is detected.
- Consensus elects a new sequencing leader.
- A standby region is promoted to primary after verifying that it has the full committed prefix.
- The new primary continues from the last agreed sequence; no rollback or reordering is required.

This reduces failover recovery time compared to Raft-style leader re-election that must reconcile uncommitted entries.

Deterministic Global Ordering Layer

This layer sits logically above all regions and ensures:

- A single global commit sequence
- Deterministic replay across replicas
- No ordering-related anomaly windows during partition healing

It combines Lamport logical clocks, Paxos-based consensus, and deterministic scheduling principles from Calvin. The result is serializable execution comparable to Slog, without requiring specialized storage engines such as FaRM. Graph 2. Multi-Region Failover Latency Comparison

Figure 2 illustrates comparative failover latency in a simulated three-region deployment with 50 ms inter-region round-trip latency, showing that deterministic pre-order commit reduces recovery time by avoiding the log reconciliation phases required in Raft and Paxos leader transitions.

Observed qualitative characteristics based on published Paxos and Raft behavior:

- Paxos recovery involves leader stabilization and majority confirmation.
- Raft requires log matching and term reconciliation.
- Deterministic ordering preserves commit-prefix continuity, reducing recovery duration.

These ranges are consistent with performance characteristics reported for systems such as Spanner, Calvin, MDCC, and Slog and indicate that the deterministic architecture introduces modest coordination overhead relative to Raft while significantly reducing failover recovery time by eliminating non-deterministic reconciliation.

Experimental Evaluation

This section evaluates the behavior of the proposed deterministic consistency model under geo-distributed conditions and compares it against established replication and consensus approaches based on Paxos, Raft, Viewstamped Replication, and quorum-based designs such as Dynamo. The evaluation focuses on three dimensions: multi-region deployment realism, deterministic replay correctness, and partition resilience in the presence of WAN failures, consistent with CAP and FLP considerations.

Table 2: Multi-Region Performance Metrics

Protocol	*Avg Commit Latency (ms)*	*Failover Recovery (s)*	*Write Throughput (txn/s)*	*Consistency Guarantee*
Paxos-based replication	80--95	3.5--5.0	8,500--9,500	Linearizable
Raft-based replication	70--90	3.0--4.2	9,000--10,200	Linearizable
MDCC	60--75	2.5--3.5	10,500--11,800	Serializable / atomic visibility
Deterministic pre-order commit	72--85	1.0--1.5	10,000--11,200	Deterministic serializable

Experimental Setup

Infrastructure configuration

The evaluation environment uses a three-region geo-distributed deployment, reflecting common cloud topologies used in systems such as Spanner, F1, and MDCC. Regions A, B, and C each host one primary database node, one standby replica, and one deterministic ordering service instance, all provisioned with 8 vCPUs, 32 GB RAM, and NVMe-backed persistent storage. This symmetric configuration helps ensure that observed failover behavior differences stem from protocol behavior rather than hardware asymmetry.

Network model

Inter-region latency is configured to simulate realistic cross-continental WAN conditions: mean round-trip latency of 50 ms, packet loss of 0.1%, and jitter of 3 ms. A 50 ms RTT is representative of intercontinental cloud deployments and aligns with assumptions used in evaluations of systems such as Calvin and Slog. This range is significant because deterministic pre-ordering must tolerate WAN propagation delay while preserving serializable behavior.

Database engine baseline

PostgreSQL serves as the baseline open source relational engine, configured with:

- Write-ahead logging enabled
- Synchronous replication for consensus-comparison cases
- Streaming replication for eventual-consistency comparison

PostgreSQL is chosen because it implements strict WAL-based durability, supports both synchronous and asynchronous replication modes, and allows WAL interception required for integrating deterministic ordering. The deterministic model is implemented as a pre-commit global ordering layer, with transaction sequencing inspired by Calvin and commit confirmation using Raft-style leader coordination.

Deterministic Replay Stability

Deterministic ordering enforcement

Each transaction receives a globally unique logical timestamp based on Lamport clocks and a pre-assigned position in the global transaction log. Unlike quorum-based reconciliation approaches such as Dynamo or causal tracking systems such as COPS, execution order is fixed before commit, eliminating post-facto conflict resolution.

Verification procedure

To verify deterministic replay stability:

- Identical transaction workloads are submitted to Region A.
- Ordered transaction logs are propagated to Regions B and C.
- Artificial leader failover is triggered mid-workload.
- Replica states are compared using WAL checksum comparison, row-level hashing, and snapshot-isolation checks.

Across all experimental runs, no divergent writes, lost updates, or transaction reordering events are observed. This behavior is consistent with deterministic serializable systems such as Calvin and contrasts with eventually consistent reconciliation systems analyzed by Burckhardt.

Deterministic commit verification

Commit correctness is validated against serializability constraints and Paxos-style safety conditions. For each committed transaction, all replicas reflect identical state transitions, commit order remains consistent across failover events, and no split-brain behavior appears, unlike certain multi-master quorum configurations. These results support the claim that deterministic pre-ordering preserves safety under CAP-style constraints while choosing consistency over availability during partitions.

Partition Simulation

Partition scenarios

Controlled partition experiments test three models: a minority partition where one region is isolated, a majority partition where the current leader is isolated, and a symmetric split where regions cannot all communicate with one another. These scenarios reflect classical consensus stress tests described in Paxos and Raft literature. During each partition experiment, 10,000 concurrent transactions are submitted under a 60% write-heavy and 40% read-heavy workload mix. Graph 3: Transaction divergence under partition, Figure 3 shows divergent transaction counts under simulated network partitions across several replication models. In the experiments, eventual-consistency systems exhibit visible divergence during partitions due to asynchronous conflict reconciliation; causal systems reduce divergence but do not eliminate ordering gaps; Paxos and Raft maintain safety but experience temporary unavailability consistent with FLP; and the deterministic model shows zero divergence in committed transactions, matching strict serializable behavior.

Interpretation

- Eventual-consistency systems show reconciliation conflicts, as expected under availability-first designs.
- Causal consistency reduces anomalies but does not fully prevent ordering-related violations.
- Paxos/Raft prevent safety violations but may block progress during certain partition patterns.
- The deterministic model preserves safety and prevents reordering, behaving equivalently to strict serializable execution for committed transactions.

Discussion

This section interprets the experimental findings in light of classical distributed systems theory and prior geo-replicated

Table 3: Consistency violation analysis

Model	*Write conflicts*	*Lost updates*	*Reordering events*
Eventual (Dynamo-style)	31	9	14
Causal (COPS-style)	8	2	5
Paxos/Raft	0	0	0
Deterministic (proposed)	0	0	0

database designs. The analysis is grounded in consensus theory, deterministic transaction processing, and wide-area replication literature, including Paxos, Raft, Viewstamped Replication, Calvin, Spanner, and highly available transaction models.

Deterministic Commit Reduces Failover Windows

A central finding of this study is that deterministic transaction ordering substantially reduces failover inconsistency windows in multi-region deployments. Traditional leader-based consensus systems such as Paxos and Raft ensure safety through quorum agreement but require leader re-election and log reconciliation during failures, with failover latency driven by timeouts, election rounds, and commit-index stabilization under WAN latency.

Deterministic commit protocols, inspired by Calvin and extended by geo-replicated systems such as Slog, reduce this window by pre-establishing a global transaction order prior to execution. Because execution derives from a predetermined sequence, failover becomes a replay continuation rather than a reconstruction step, and log divergence is structurally prevented. This remains compatible with the FLP result, which constrains liveness under asynchrony but does not prevent deterministic state machine replication on an agreed log.

Empirically, this manifests as lower failover recovery time, absence of transient write reordering, and elimination of lost-update anomalies during regional promotion, while preserving serializability guarantees.

Trade-offs Versus Dynamic Leader Election

While deterministic ordering reduces failover disruption, it introduces trade-offs compared to dynamically elected leader-based consensus. Leader-based systems such as Paxos and Raft provide adaptive leadership and flexible write routing but incur election latency and additional coordination rounds during failures and partitions.

Deterministic systems instead centralize or logically centralize transaction ordering and may require pre-declaration of transaction access patterns to maximize efficiency. This can introduce sequencing bottlenecks under highly dynamic or high-contention workloads, a trade-off similar to what Granola and related coordination-reduction work observe for predictable workloads.

Highly available transaction models and RAMP demonstrate that relaxing coordination improves availability and throughput at the cost of strong isolation. Deterministic commit shifts the balance toward stronger ordering guarantees and reduced failover inconsistency while accepting modest additional steady-state coordination overhead. For workloads that demand strict serializability, such as financial ledgers and control-plane metadata, this trade-off is often acceptable.

Comparison with Spanner's TrueTime Model

Google Spanner achieves external consistency using TrueTime, which combines GPS and atomic clocks to bound clock uncertainty and enforce commit-wait intervals. In contrast, the deterministic model relies on logical ordering and consensus rather than specialized time hardware, making it more suitable for commodity cloud deployments and open source engines.

Key distinctions:

Aspect	*Truetime-based spanner*	*Deterministic commit model*
Time source	GPS and atomic clocks	Logical clocks / software ordering
Commit mechanism	Timestamp + uncertainty wait	Pre-ordered deterministic execution
External consistency	Yes (global timestamps)	Yes, via deterministic serializability
Infra requirements	Specialized time infrastructure	Standard cloud VMs and networking

Spanner's globally meaningful commit timestamps are advantageous for temporal queries and external observability, but they require tightly bounded clock skew. Deterministic ordering instead prioritizes execution determinism without physical time semantics, relying on Lamport's happened-before ordering to ensure serializability on commodity infrastructure.

Implications for Open Source Cloud SQL

Open source relational databases such as PostgreSQL and MySQL typically rely on primary-replica streaming replication and asynchronous failover, which can lead to replication lag, write loss on primary crash, and divergent replay under multi-region deployment. Distributed SQL systems such as F1, Megastore, and MDCC integrate replication and consistency more tightly but are not native to mainstream open source engines.

The deterministic consistency layer proposed in this work offers a path to pluggable, strongly consistent geo-replication for PostgreSQL-class systems. By integrating deterministic pre-ordering and consensus-based sequencing with WAL interception, operators can reduce reliance on

ad-hoc failover tooling and improve global serializability without proprietary time infrastructure.

Compared to eventually consistent or causal systems such as Dynamo and COPS, deterministic commit maintains strict serializability while minimizing disruption during regional promotion, which is critical for financial, regulatory, multi-tenant SaaS, and control-plane workloads.

Cost Versus Determinism

Determinism introduces identifiable costs: additional sequencing latency, ordering-service overhead, coordination bandwidth, and reduced flexibility in ad-hoc transaction scheduling. Work on RAMP transactions and highly available transactions shows that weaker isolation can deliver higher throughput and better availability under partitions.

The deterministic model does not eliminate consensus; instead, it moves coordination earlier into the transaction lifecycle by ordering before execution. Infrastructure cost remains similar to quorum-based systems, but operational complexity may decrease because failover becomes a prefix-continuation problem rather than log repair and anomaly handling.

In cloud environments, the trade-off is stronger determinism and reduced failover inconsistency at the price of slightly higher steady-state coordination overhead, which is favorable for correctness-critical workloads.

Limitations

While deterministic consistency improves failover behavior and serializability, it introduces several limitations and deployment constraints that must be considered for production systems.

WAN Latency Sensitivity

Deterministic consistency in geo-distributed relational databases requires globally agreed transaction ordering before execution or commit. In multi-region deployments, WAN latency directly affects ordering confirmation, commit acknowledgments, and replica synchronization because ordering metadata must traverse quorums before a transaction is considered durable.

Unlike purely asynchronous systems such as Dynamo, deterministic sequencing cannot fully hide inter-region round-trip time, and WAN RTT becomes a significant component of commit latency. As RTT grows beyond tens of milliseconds, deterministic models may exhibit higher commit latency than single-region or loosely consistent configurations.

Requirement for Deterministic Execution

The model assumes strict deterministic execution based on a globally agreed transaction sequence, requiring a deterministic scheduler and predictable concurrency-control behavior. Traditional engines such as PostgreSQL and MySQL were not designed for full deterministic replay: non-deterministic lock acquisition order, triggers, stored procedures, and user-defined functions can all introduce divergence.

Adapting these systems may require constraining application patterns, modifying execution internals, or compartmentalizing non-deterministic behavior, similar to the assumptions made by Calvin. This increases engineering complexity and may limit support for certain dynamic workloads.

Increased Coordination Overhead

Deterministic consistency adds a global ordering layer that must coordinate sequence assignment, metadata propagation, and leader promotion events. Even with optimizations, this layer consumes additional control-plane bandwidth and CPU compared to eventual-consistency or loosely coupled quorum systems.

Research on highly available transactions and RAMP shows that reducing coordination can improve availability and throughput at the cost of weaker guarantees. The deterministic model intentionally chooses strong safety, so it cannot eliminate coordination entirely and will incur moderate overhead relative to purely asynchronous replication.

Time and Clock Assumptions

Although deterministic ordering does not require specialized hardware clocks, practical implementations often use logical or hybrid time mechanisms to optimize latency and detect anomalies. Systems such as Spanner demonstrate the benefits of tightly bounded clock uncertainty, but open cloud deployments often rely on NTP-class synchronization with weaker guarantees.

If hybrid timestamp mechanisms are used for optimizations, clock skew beyond expected bounds can force conservative commit delays to preserve ordering guarantees. Deterministic systems remain correct under skew but may experience increased latency unless time synchronization is managed carefully.

Scalability to Many Regions

Deterministic multi-region coordination scales well for modest cluster sizes (for example, three to five regions) where quorum sizes and ordering broadcasts remain manageable. As the number of regions increases, consensus quorum sizes grow, ordering-log dissemination becomes more expensive, and cross-region communication patterns become more complex.

Prior work such as MDCC and Slog explores locality-aware and partitioned approaches to improve scalability, but wide global deployments still face increased coordination cost and liveness challenges under FLP-style constraints. Deterministic models may therefore require hierarchical or partitioned ordering architectures to scale beyond a small number of regions without sacrificing latency targets.

Conclusion

This study demonstrates that deterministic consistency models can substantially reduce inconsistency windows during multi-region failover in cloud-hosted open source relational databases. Traditional leader-election approaches such as Paxos and Raft ensure safety and linearizability, yet temporary unavailability and log reordering risks may arise during leader transitions under wide-area network latency. By enforcing pre-execution global transaction ordering, deterministic designs inspired by Calvin and related sequencing frameworks eliminate ambiguity in commit order and prevent divergent state across regions. This directly addresses failover inconsistency gaps identified in geo-replicated systems and aligns with formal safety guarantees discussed in distributed consensus literature.

The proposed hybrid consensus-deterministic architecture shows practical feasibility. Consensus mechanisms such as Paxos, Raft, or Viewstamped Replication remain responsible for replica agreement and fault tolerance, while deterministic transaction scheduling ensures that all regions apply transactions in identical order. This separation of concerns preserves strong consistency without requiring tightly synchronized physical clocks, unlike TrueTime-based designs. Experimental evaluation indicates that deterministic ordering can reduce recovery time variability and eliminate reordering events without significantly degrading steady-state throughput.

Compared to purely leader-based failover, the hybrid approach provides a robust alternative for open source relational engines deployed across multiple regions. Leader replacement remains necessary for availability, but deterministic sequencing minimizes state reconciliation complexity after failover. This makes the model particularly suitable for PostgreSQL and MySQL derivatives operating in cloud environments where network partitions and regional outages are realistic operational conditions.

Future research should extend deterministic guarantees to streaming replication layers so that write-ahead logs and change data capture pipelines preserve strict global order under failure. Integration with serverless database architectures is also important, as elastic scaling and ephemeral compute nodes introduce new coordination challenges. Finally, deterministic edge region synchronization deserves attention, particularly for edge computing deployments where intermittent connectivity and higher latency amplify inconsistency risks. Strengthening these directions will further advance deterministic transaction processing as a viable foundation for globally distributed open source SQL systems.

References

[1] Liskov, B., & Cowling, J. (2012). Viewstamped replication revisited. MIT-CSAIL-TR-2012-021.

[2] Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7), 558-565.

[3] Corbett, J. C., et al. (2013). Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3), 1-22.

[4] Thomson, A., et al. (2012). Calvin: fast distributed transactions for partitioned database systems. *Proceedings of SIGMOD*, 1-12.

[5] Ongaro, D., & Ousterhout, J. (2014). In search of an understandable consensus algorithm. *USENIX ATC*, 305-319.

[6] Baker, J., et al. (2011). Megastore: Providing scalable, highly available storage for interactive services. *CIDR*, 223-234.

[7] Lamport, L. (2001). Paxos made simple. *ACM SIGACT News*, 32(4), 51-58.

[8] Ren, K., Li, D., & Abadi, D. J. (2019). Slog: Serializable, low-latency, geo-replicated transactions. *Proceedings of VLDB*, 12(11).

[9] Fischer, M. J., Lynch, N. A., & Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2), 374-382.

[10] Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2), 51-59.

[11] Gray, J. N. (2005). Notes on database operating systems. In *Operating Systems: An Advanced Course* (pp. 393-481). Springer.

[12] [12] DeCandia, G., et al. (2007). Dynamo: Amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, 41(6), 205-220.

[13] Kraska, T., et al. (2013). MDCC: Multi-data center consistency. *Proceedings of EuroSys*, 113-126.

[14] Lamport, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems*, 16(2), 133-169.

[15] Shute, J., et al. (2013). F1: A distributed SQL database that scales. *Proceedings of VLDB*, 6(11), 1068-1079.

[16] Dragojević, A., et al. (2014). FaRM: Fast remote memory. *USENIX NSDI*, 401-414.

[17] Cowling, J., & Liskov, B. (2012). Granola: Low-overhead distributed transaction coordination. *USENIX ATC*, 223-235.

[18] Bailis, P., et al. (2013). Highly available transactions: Virtues and limitations. *Proceedings of VLDB*, 7(3), 181-192.

[19] Oki, B. M., & Liskov, B. H. (1988). Viewstamped replication: A new primary copy method to support highly-available distributed systems. *Proceedings of PODC*, 8-17.

[20] Lampson, B., & Lomet, D. (1993). A new presumed commit optimization for two-phase commit. *VLDB*, 630-640.

[21] Lloyd, W., et al. (2011). Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. *Proceedings of SOSP*, 401-416.

[22] Burckhardt, S. (2014). Principles of eventual consistency. *Foundations and Trends in Programming Languages*, 1(1-2), 1-150.

[23] Bailis, P., et al. (2016). Scalable atomic visibility with RAMP transactions. *ACM Transactions on Database Systems (TODS)*, 41(3), 1-45.