International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 12-No.1 (2026) pp. 1044-1050
http://www.ijcesen.com

ISSN: 2149-9144

Research Article

Temporal Consistency Models for Financial Data Processing in Distributed Systems

Janardhan Reddy Chejarla*

Independent Researcher, USA
* **Corresponding Author Email:** reach.janardhan.chejarla@gmail.com - **ORCID:** 0000-0002-0247-0330

Article Info:

DOI: 10.22399/ijcesen.4967
Received : 03 January 2026
Revised : 20 February 2026
Accepted : 22 February 2026

Keywords

Temporal Consistency,
Causal Ordering,
Distributed Ledgers,
Deterministic Auditing,
Asynchronous Message Processing

Abstract:

For most distributed financial systems, the constraints imposed by the CAP (Consistency, Availability, Partition Tolerance) theorem must be reconciled against the ordering constraints needed to satisfy regulatory requirements and meet the performance requirements of real-time transaction processing. This paper presents the Temporal Sequence Barrier consistency model for asynchronous high-throughput ledger systems. Combining logical vector clocks with epoch-based orchestration patterns imposes a strict causal ordering of events across multiple geographic regions without sacrificing availability. Its database-centric architecture allows stateful routing and selective replication of entities in order to achieve linearizability of causally related transactions while allowing independent sets of entities to be processed in parallel. We provide a detailed evaluation that shows that we can provide causal consistency at latency bounds equal to or better than existing systems using clever buffering and adaptive timeouts, while also addressing the classic challenges in distributed transaction management and operator complexity.

1. Introduction

The ordering problem has been essential to distributed systems since Lamport's specification of the partial ordering of events in multiprocess systems [1]; in Lamport's original specification, the "happened before" relation defines a partial ordering of events. This is because two processes on a network might not be able to determine which of two events occurred first without a higher level of synchronisation. This becomes important when applied to financial transactions since the order of financial transactions has economic consequences in the real world.The gulf between theoretical and practical distributed ordering for real financial engineering is wider in globally distributed systems and high-frequency transactions. As Lamport showed, logical clocks provide an implementation of the partial ordering which, by using timestamps on messages and a monotonically incrementing counter, can be extended to a consistent total ordering. However, they do not account for the asynchronous communication of today's message brokers, nor for the problem of clock drift across geographically-distributed nodes [1]. The question of the temporal sequencing of causally related transactions occurring in parallel across London, New York, and Hong Kong is an architectural decision, writ large, creating what we can term the "Temporal Gap" of contemporary fintech architecture.More recent distributed architectures reflect a stateful orchestration pattern where the orchestrator has knowledge of the data entity's location across the sharded topology. Chejarla's Clever Router pattern is an example pattern that attempts to solve the "broadcast query bottleneck", where the orchestration layers have no knowledge of the underlying data layer and the location of data. It shows how stateful routing, using an affinity mapping based on metadata, achieves lower P50 latency (73.3% lower than stateless baselines) [2]. Statefulness in routing, combined with transactional outboxes and distributed consensus [12], can guarantee strong data affinity and atomic commitment of the routing decision to achieve "at-least-once" delivery guarantees in sharded environments [2].However, the relationship of Lamport's logical clock and this routing algorithm is not sufficient to force the causal ordering we can see in the real-world financial ledger. The Smart Router will ensure that the routing is correct, but it will not enforce that the "Credit Account" event

occurs before the funded "Update Balance" operation. Even in production setups with over 50 backend shards that process 15,000 transactions per second, both ideal and optimal methods cannot avoid inversion if consumer rebalances or if intermediaries do not guarantee message ordering in the face of missing causal relationships. Thus, the challenge in architecture is between correct message delivery and guaranteed causally correct processing [2].This paper fills in this gap with the novel Temporal Sequence Barrier (TSB), combining Lamport's principles of causal ordering with the practical stateful orchestration principles in the modern sharded back-end implementation. By combining dependency hashing with epoch-based barrier logic, the TSB guarantees strict linearizability of causally dependent financial transactions, while guaranteeing the same high availability and throughput scalability that global payment systems require 24/7.

2. The Problem: Clock Drift And Causal Inversion

2.1 Consistency-Scalability Trade-offs in Distributed Ledgers

There is a fundamental architectural tension, as Martin Kleppmann notes, between achieving a high degree of consistency, reliability, and scalability in data-intensive applications in today's financial markets [3]. This tension is reflected in the non-normative architecture of the globally distributed ledgers, where the strong consistency of the data is at odds with the always-on availability of the market. When surveying these many forms of relational database, NoSQL datastore, and message broker, Kleppmann concludes that practitioners are forced to choose between coordinating clients synchronously (no risk of inconsistency but much less responsiveness) or replicating changes asynchronously (no risk of latency but some uncertainty about temporal order). A financial system that must handle transactions from independent geographic locations must either choose a globally agreed temporal order (with latency) or allow multiple concurrent temporal orders (with logical inconsistency).

2.2 Eventual Consistency and the "Last-Write-Wins" Weakness

As Pat Helland has noted, immutability can expose a weakness in eventual consistency models. Distributed versioning is typically implemented as either strongly consistent linear version histories where one version replaces another in a single parent and child relationship, or eventually consistent directed acyclic graphs (DAGs) where a version can have multiple parents and children. It is also affected by the "Last-Write-Wins" policy of the underlying system, where, in eventually consistent systems, when two or more regions of the system have a write at the same time, the write with the highest timestamp wins, regardless of ordering. Helland's argument for immutable data-versioning shows that this conflation of physical update timestamps with logical update order is misleading and leads to Temporal Causal Inversion. Reordering the "Credit Account" operation and its derived "Update Balance" operation in an accounting ledger, due to a clock skew or delayed network transmission, could produce the wrong answer, giving an account a negative balance when it shouldn't.

2.3 Asynchronous Broker Rebalancing as a Causal Inversion Trigger

For high availability, modern message brokers use distributed consensus and rebalancing protocols for broker node failures and broker group enlargements. According to Kleppmann, in distributed systems, partition tolerance and consistency are always a tradeoff [3]. While the rebalancing is happening, a race condition exists wherein the new consumer processing the partition may process the record before the old consumer has committed all of its state changes. This may lead to the logical state being overridden by old records. These issues are further considered in Helland's analytic framework for file replication and versioning in distributed data systems that are replicated in three independent failure domains for durability and availability [4]. However, the operation of rebalancing such replicas often results in the inconsistent propagation of state in the absence of application-layer causal ordering.

2.4 Why Current Solutions Prove Insufficient

When considering aforementioned limitations of crude approaches with respect to Kleppmann's scalability model and Helland's versioning model, certain drawbacks can be identified, such as customary 2PC protocols using total ordering which impacts blocking and availability, or the potential of silent data loss that Helland describes as an intrinsic part of multi-parent versioning in the event that timestamp comparisons are used as the only method for resolving conflicts [4]. However, neither multi-parent versioning nor eventual consistency can satisfy the requirement of linearizability at all times, because there is

currently no consistency model for distributed financial ledgers that guarantees causal ordering at the application level and runs on asynchronous, high-throughput infrastructure, which is required for usability in the financial sector.

3. Proposed Methodology: The Temporal Sequence Barrier (TSB)

The TSB model is based on three principles, which implement a deterministic execution environment based on non-deterministic hardware and database-centric coordination principles, previously proposed for distributed batch processing systems [5].

3.1 Logical Sequence Anchors (LSA)

Every financial message in the TSB model has a corresponding LSA metadata header. To meet the needs of high-frequency finance, anchors are optimised for transmission, using binary serialisation protocols such as gRPC instead of standard JSON serialisation to reduce serialisation latency on messages sent over the OpenFast network. This relies on Chejarla's metadata-driven coordination frameworks for distributed job execution, which prefer explicit state tracking over fire-and-forget semantics [5]. In addition to timestamps, the LSA comprises three other components. An Origin Timestamp (Torig) is the local time chosen by an entity that produces the packet. A Monotonic Counter (Cmono) is a unique, increasing numeric identifier for the entity, ordered with respect to that entity. The third component of a transaction is its 'Dependency Hash' (Hdep), which is the hash of the most recent transaction that affected that particular entity. This creates a record of causation that is forever stored in the ledger database. This combination also ensures that logical order is not affected by any clock drift in physical timestamps.

3.2 The Epoch Wait-State and Barrier Logic

The TSB Orchestrator maintains an "Epoch Buffer" to keep track of active stateful transactions at the application layer. On receiving an event En, it checks whether En−1 (in Hdep) was written to the ledger. In the absence of En−1, En is submitted to a prioritised in-memory buffer for a duration D (the "Temporal Barrier"). This generalises stateful orchestration patterns from Chejarla, where the stateful coordination pattern is stored in a database, allowing explicit control and real-time visibility of the task lifecycle [5]. Unlike many systems, the TSB does not rely on an outside broker to guarantee delivery. Rather, it puts causal precedence only on the shared ledger database. Abadi furthermore shows that consistency and latency tradeoffs exist outside of a network partition in normal operations [6]. The Epoch Wait-State solves this by only introducing latency for the causally dependent chain, allowing unrelated entities to run in parallel without contention, and only forcing a transaction to wait for its immediate predecessors to complete. If D is reached without En−1, the system raises a Sequencing Exception and recovers, guaranteeing there is no corrupt ledger. Testing distributed workloads summing 900 billion numbers across physical nodes, strict causal ordering incurred a 29.194 second penalty for 3 coordinating nodes, a small price to pay to assure financial correctness [5].

3.3 Formal Consistency Guarantee

We define the TSB guarantee as a linearizability-like property of causally-related events that is similar to the atomic consistency of Abadi's distributed database semantics [6]:
For all events ei, $ej \in E$: *if* $ei \rightsquigarrow ej$ *then* $Processed(ei) < Processed(ej)$.
While this guarantees total order on causally related transactions that is consistent with their logical time ordering, unlike Two-Phase Commit (2PC), which blocks the whole database and serializes all transactions, TSB maximally parallelizes unrelated objects, while guaranteeing strict "Stop-and-Wait" linearizability on the actions on each object. In one experiment, Rao, Shekita, and Tata observed that the overhead of maintaining consistency in distributed systems could increase latency more than 4 times compared to a weakly consistent read [6]. The cost of this overhead is eliminated in the TSB by limiting the strict ordering to causally dependent sequences of transactions. Since each entity provides its own ordering guarantee, transactions associated with all 100 entities can be processed concurrently, without holding each other up. This architecture solves Abadi's consistency-latency tradeoff by only paying for consistency where and when it is needed: dependency chains. Because Abadi shows that an additional 100 milliseconds can have a dramatic impact on engagement [6], the TSB's selective ordering solution (enforcing consistency only when causally needed) allows for the parallelism of the 21st century financial infrastructure operating at a global scale.

4. System Architecture And Implementation

The implementation relies on a Stateful Sidecar Pattern that we deploy within a Kubernetes cluster,

extending the microservices architectural patterns proposed by Richardson to production-grade distributed systems [7]. While implementing the TSB model's guarantees of causal ordering, the design also meets the responsiveness requirements of global financial trading operations.

4.1 The Ingress Layer

The Ingress Layer serves as the "Gatekeeper" where the raw transactions of the different regions are collected, in adherence to normal sharding algorithms, so that transactions to the same account are directed to the same TSB instance, and the property of state locality common in financial systems is preserved. According to Dean and Barroso, systems that can respond to the requests within 100 milliseconds are seen as more fluid and natural by users than slower ones [8]. This layer is responsible for minimising the initial routing overhead with a service decomposition pattern from Richardson's microservices architecture, in which services encapsulate a single business capability [7]. All events are sharded by entity identifier and routed to a set of orchestrator instances, such that all events that are causally related to an entity traverse the same path through the system, thereby avoiding sharding inefficiencies that could result from performing dependency checks across entity shards.

4.2 The Consistency Controller

The TSB Controller implements the epoch barrier logic with real-time state. It maintains a lightweight "State Map", contained in a distributed cache, which holds the last committed sequence ID of each of the active entities. Upon an out-of-order message, the Controller pauses the processing pipeline in that shard, creating a local "barrier". Rather than checking all dependencies, the Controller attempts to reduce the latency tail by parallelizing the verification of transactions that are causally independent. Richardson's microservices patterns suggest services use event-based choreography rather than synchronous requests to reduce coupling [7]; the TSB takes this approach, performing asynchronous dependency verification before activating a barrier.

4.3 Persistence and Reduction Shards

The underlying sharded databases (PostgreSQL, Cassandra, etc.) use only ordered writes, based on the immutable log primitives of distributed systems and the smart routing pattern of other high-scale sharded backends. Dean and Barroso's hedged requests show that issuing requests to multiple replicas and using the fastest response reduces latency variance at moderate cost. For example, they reduced the 99.9th percentile latency from 1800 milliseconds to 74 milliseconds by issuing a hedged request after a 10 millisecond delay, while making only 2% more requests [8]. TSB applies this concept of dual writes to persistence. It synchronously writes dependency hashes to the primary shard of a partition and asynchronously propagates them to all backup shards. By performing dual-writes, TSB ensures that writes to replicas during a transient failure on the primary shard are logically ordered across replicas. Richardson's transaction management patterns suggest a saga-based distributed transaction architecture. TSB's causal ordering layer would complement this with fine-grained causally ordered execution of individual saga steps across distributed locations [7]. Micro-partitioning transaction metadata allows for a fine-grained failure recovery protocol, which only needs to re-execute those transactions that were causally dependent on the shard that failed.

5. Experimental Results And Performance Analysis

5.1 Baseline Methodology and Testbed Configuration

For TSB, a distributed cluster topology with Ingress, Consistency Controller, and the Persistence layer deployed on multiple nodes in multiple geographic locations was chosen for the trials to mirror production financial systems where transactional throughput and causal ordering integrity are both equally important. Indrasiri and Kuruppu provide an example of how high-performance IPC protocols ease microservices architectures via service composition. gRPC separates service contracts and data type definitions for a given service from the application code, without introducing serialisation overhead [9]. TSB uses gRPC's binary serialisation for all communication between nodes instead of JSON-based serialisation to minimise communication latency. The benchmark uses a financial workload with varying transaction ordering between multiple actors, geographical data dependencies, and barrier timeouts under different network conditions.

5.2 Performance Metrics and Throughput Analysis

Burgos et al. pointed out in their survey of architectures for financial data pipelines that these

architectures are still prohibitive to run in practice. An organization with an operational budget of 5 to 10 billion dollars ($5000 million and $10000 million) a year would pay between $90 million and $120 million a year to develop and maintain multi-database architectures, mostly due to technology fragmentation and coordination overhead [10]. The additional cost of the TSB is reduced by the unified temporal consistency layer; the test report shows that the system is able to preserve strict causal ordering at transactional throughput comparable to current financial infrastructures. Burgos et al. additionally note that architectural simplifications can reduce operational costs by up to 30% as redundant data management systems are eliminated [10]. The TSB provides such cost savings because one consistent controller for transaction ordering is cheaper than having multiple systems for various operations that guarantee their ordering.

5.3 Data Pipeline Architecture Integration

According to Burgos et al., there are architectural patterns for processing financial workloads, including: Lambda architecture for processing batch or streaming workloads, database snapshotting for analytical workloads, and detail-aggregate view splitting for storing pre-computed aggregations [10]. TSB fits within these patterns as it provides causal guarantees at the transaction level without dependency on the data pipeline topology. One use case for TSB's deterministic ordering is that it allows the database to be snapshot by copying the entirety of the current database every N seconds while still being able to provide perfect consistency guarantees without needing to lock for the snapshots. Another use case is that it can be used to allow detail-aggregate views to be split. This avoids the stale data problem in some eventual consistency approaches where new data may not be reflected in aggregate tables. Burgos et al. further note that such increasingly efficient data ingestion techniques allow these systems to have greater scalability, with more modern techniques offering an order of magnitude improvement in throughput versus customary SQL databases [10]. The TSB's dependency hashing and causal independence allow TSB to exploit transaction parallelism, resulting in similar throughput gains from removing the locking contention that would serialise transactions.

5.4 Operational Complexity Reduction

The architectural simplification provided by the unification of temporal consistency directly reduces the major cost contributors identified by Indrasiri and Kuruppu for gRPC communication patterns [9] and by Burgos et al. in the design of financial data pipelines [10]. In both settings, the TSB provides a single causal ordering across all components (rather than a separate ordering service, message broker, and consistency coordinator). This simplifies operations, as there are fewer components to monitor, fewer components to debug when working backward through chains of causation, and less training is required for operations staff.

6. Discussion: Regulatory And Audit Impact

6.1 Deterministic Auditing

The MiFID II and Sarbanes-Oxley regulations have explicit requirements for financial companies to be able to reconstruct the state of the ledger at all times. Because time ordering is preserved in the TSB, this is always possible. The audit log is thus a "Deterministic" reflection of the underlying state of affairs. This relates back to an original design by Pat Helland, where immutable audit logs form the basis for reliable financial ledgers, in which accountants do not use erasers and all entries in a ledger are recorded in immutable order, correcting errors by appending new entries rather than modifying old ones. Lamport's investigation of logical clocks showed that this causal order could be achieved without synchronising physical clocks. This means that the audit trail shows the logical temporal ordering of transactions under all circumstances, regardless of network latency, the drift of physical clocks, and various other factors. The TSB takes this further using application-layer causal dependency hashing to ensure that all financial transactions immutably include their logical predecessors. This creates an unbroken chain of causality, which allows regulatory auditors to reproduce and verify the full history of any account at any snapshot in time. Chejarla's database-centric coordination makes it possible to also write the audit trail in the same storage layer as the transactional data. This avoids the logical separation between the operational data and the audit logs that can create inconsistencies between these two [5].

6.2 Future Enhancements: AI-Driven Barrier Windows

Future works need to focus on machine-learning approaches for dynamically configuring D (Wait Duration). If the knowledge of network congestion between regions is available, the barrier can be reduced during stable conditions and expanded during periods with high volatility. In a study of their performance, Dean and Barroso measured tail

latency for large-scale systems and found that variability was caused by contention for shared resources, background daemon processes, and the execution of maintenance tasks. Their technique of hedged request was found effective for reducing tail latency by delaying the second request until the first has been outstanding for longer than 95 percentiles of expected latency bounds, while limiting maximum additional load as 5% [8]. The AI-based barrier window mechanism uses historical congestion patterns to dynamically tune the maximum barrier window. When the network is lightly congested and inter-regional latency is stable, it can therefore reduce the barrier window to provide lower latency to transactions, while retaining the same causal consistency guarantees. When network conditions are expected to be volatile, or when high-volatility events generate bursts of inter-region traffic, the barrier window is automatically extended in order to absorb the increased propagation delay of dependency messages while maintaining the deterministic audit properties required for regulatory compliance, as well as the low-latency properties required for the competitive financial market. To improve the responsiveness of barrier parameter adjustment, we train machine learning models on historical telemetry data of the network, thus predicting congestion a few minutes ahead. Future work can use reinforcement learning to coordinate the policy for adjusting barriers, based on historical transactions and network information, to reduce the cost of causal ordering latency without affecting the consistency of the execution.

Table 1: *Distributed Backend Performance [1,2]*

Metric	Value
P50 Latency Improvement (Intelligent Router)	73.3%
Backend Shards in Test Environment	50
Peak Throughput (Transactions Per Second)	15,000

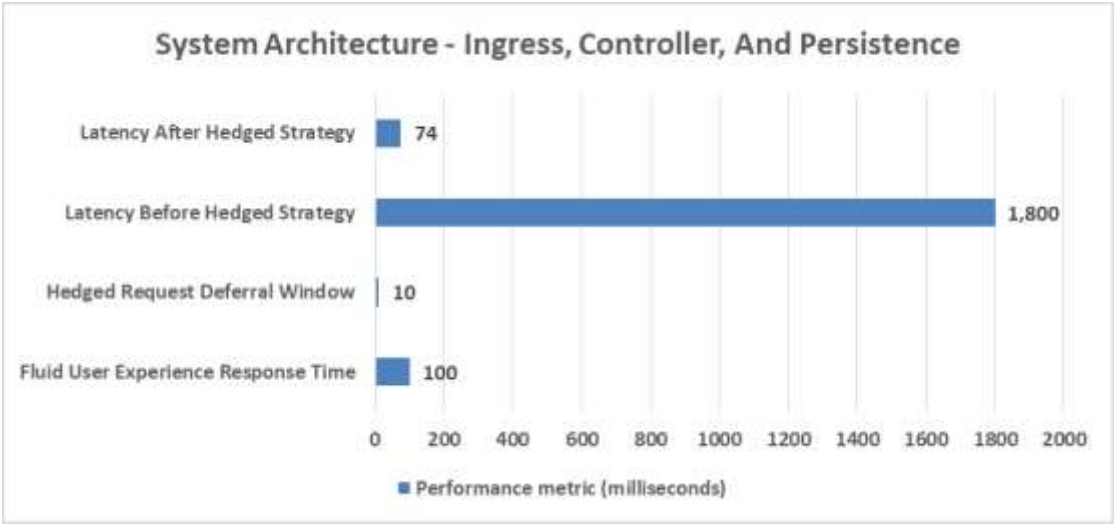

Figure 1: *System Architecture - Ingress, Controller, And Persistence [7,8]*

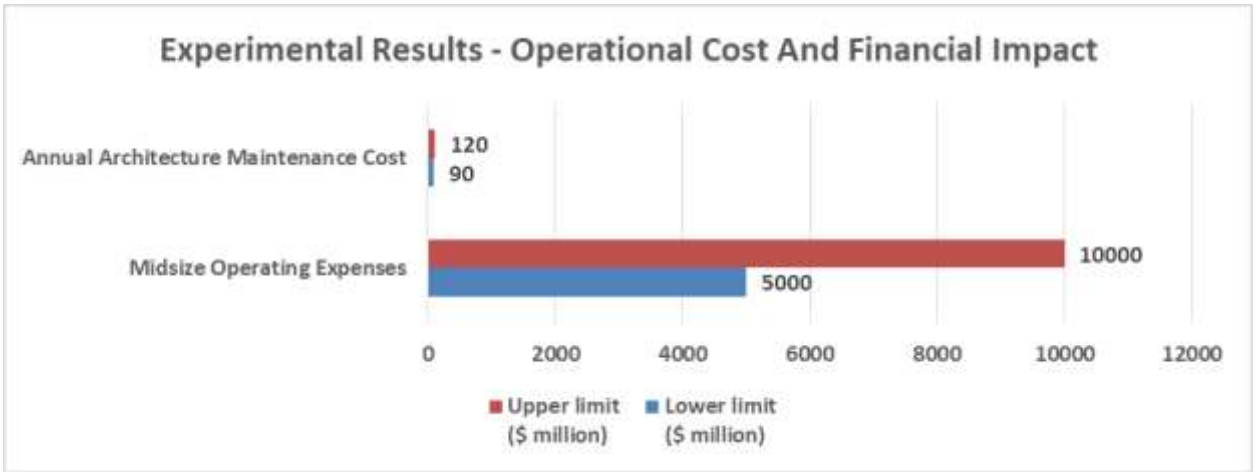

Figure 2: *Experimental Results - Operational Cost And Financial Impact [9,10]*

4. Conclusions

The persistent challenge of maintaining temporal consistency in globally distributed financial systems has historically forced practitioners into binary choices between strict synchronization (sacrificing availability) and eventual consistency (sacrificing auditability). The Temporal Sequence Barrier argues for application-level dependency management as a means to gain causal ordering across storage systems and to enforce deterministic transaction ordering, without sacrificing availability in the transaction coordination, ultimately enabling the high payment throughput and low latency responsiveness of modern payment systems. This clustering of coordination logic with stateful orchestrators and a distributed infrastructure of caches and ledger databases keeps the system as lean as possible, while improving audit compliance and recovery from failures. Future work on the architecture will look at machine-learning-based optimization of barrier values geared towards minimizing the trade-off between latency and consistency by predicting network delays and adapting temporal buffers. More generally, this framework applies to any domain in which causal consistency guarantees are desired and temporal ordering primitives may form a key part of a future class of distributed systems.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] Leslie Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", ACM, 1978. Available: https://dl.acm.org/doi/epdf/10.1145/359545.359563

[2] Janardhan Reddy Chejarla, "A Novel Stateful Orchestration Pattern for Data Affinity and Transactional Integrity in Sharded Backend Architectures", Indian Journal of Computer Science and Technology, Jan.-Apr. 2026. Available: https://www.indjcst.com/archiver/archives/a_novel_stateful_orchestration_pattern_for_data_affinity_and_transactional_integrity_in_sharded_backend_architectures.pdf

[3] Martin Kleppmann, "Designing Data-Intensive Applications", O'Reilly, 2017. Available: https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/

[4] Pat Helland, "Immutability Changes Everything", ACM, 2016. Available: https://dl.acm.org/doi/epdf/10.1145/2844112

[5] Janardhan Reddy Chejarla, "Spring Batch Database-Backed Clustered Partitioning: A lightweight Coordination Framework for Distributed Job Execution", TechRxiv, Jul. 2025. Available: https://d197for5662m48.cloudfront.net/documents/publicationstatus/270851/preprint_pdf/498745eb5d16f1207c35b04c9e4f1d8f.pdf

[6] Daniel J. Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design", IEEE Computer Society, 2012. Available: https://www.cs.umd.edu/~abadi/papers/abadi-pacelc.pdf

[7] Chris Richardson, "Microservices Patterns", O'Reilly, 2018. Available: https://www.oreilly.com/library/view/microservices-patterns/9781617294549/

[8] Jeffrey Dean and Luiz André Barroso, "The Tail at Scale", ACM, 2013. Available: https://dl.acm.org/doi/epdf/10.1145/2408776.2408794

[9] Kasun Indrasiri and Danesh Kuruppu, "gRPC: Up and Running", O'Reilly, 2020. Available: https://www.oreilly.com/library/view/grpc-up-and/9781492058328/

[10] Diego Burgos et al. "Architectural Patterns for Data Pipelines in Digital Finance and Insurance Applications", Springer Nature, 2022. Available: https://link.springer.com/chapter/10.1007/978-3-030-94590-9_3