The Eastasouth Journal of Information System and Computer Science
Vol. 1, No. 01, August, pp. 207 - 218

Semantic Idempotency for Exactly-Once Business Effects in At-Least-Once Distributed Messaging Systems

Karthik Juluri[1], Srinivas Nune[2]
[1] Independent Researcher, Glendale, USA
[2] Independent Researcher, Glendale, USA

Article Info

Article history:

Received, Aug, 2023
Revised, Aug, 2023
Accepted, Aug, 2023

Keywords:

At-Least-Once Delivery;
Business Effect Consistency;
Conflict-Free Replicated Data Types;
Consensus Protocols;
Distributed Messaging;
Event-Driven Architecture;
Exactly-Once Semantics;
Fault Tolerance;
Idempotency;
Message Deduplication;
Saga Pattern;
Stream Processing

ABSTRACT

At-least-once delivery is the dominant message-delivery paradigm in distributed messaging infrastructures, because stronger transport-level guarantees require blocking coordination that compromises availability and throughput. This paper gathers, from the foundational and modern literature, the concepts of message delivery semantics, of idempotency theory and of distributed coordination patterns, articulating a conceptual framework, called semantic idempotency, where exactly-once business effects are achieved on top of an at-least-once transport, by relying on deduplication performed on the receiver side instead of on the transmitter side. It starts by tracing the history of the various studies that led to the development of logical clocks and consensus impossibility theorems, quorum-based replication, log-structured messaging, snapshot algorithms, saga-based compensation, and conflict-free replicated data types, and how each addresses a partial solution to the duplicate-delivery problem. The analysis reveals that redelivery probability increases non-linearly with the number of retries at realistic acknowledgment-loss rates, the idempotency-key registry adds approximately one millisecond of median latency compared with mid-single-digit milliseconds for two-phase commit, and windowed key-store retention can maintain over 95 percent accuracy of duplicate-catching while keeping state growth under control. Idempotent receivers preserve the availability and horizontal scalability of at-least-once transport, at the cost of weaker native consistency guarantees than two-phase commit provides. The paper also presents the formalization of semantic idempotency using an operator-oriented notation that separates redelivery at the transport level from application of effects at the business level. These results suggest that architects should regard idempotency as a first-class contract between product and consumer, while the future of messaging systems might be a platform that includes a deduplication registry as a first-class citizen rather than an ad hoc application programming feature.

Corresponding Author:

Name: Karthik Juluri
Institution: Independent Researcher, Glendale, USA
Email: securityoperations40@gmail.com

1. INTRODUCTION

Today, messaging middleware is the glue that connects independently deployed services to each other with orders, payments, inventory adjustments and notification events. Messaging platforms are designed with delivery guarantees in mind that indicate the number of times a message can be delivered to a consumer because like everything else, a crash, failover of the broker, and network partition are normal circumstances and not an exception. There are three guarantees that are recognized: at-most-once, where a message might be silently dropped; at-least-once, where a message is retried until it is acknowledged and may be delivered multiple times; and exactly-once, an aspirational guarantee under which a message is delivered and processed once and only once, with neither loss nor duplication. Two distinct theoretical results constrain what the transport layer can promise. The impossibility of deterministic consensus in an asynchronous system with even one faulty process bounds what agreement protocols can achieve [1], while the coordinated-attack result shows that a sender can never determine, in finite time, whether a missing acknowledgment reflects a lost message or a lost acknowledgment. Together they imply that transport-level exactly-once delivery is unattainable in the general case. So, when a production grade broker like a log-structured messaging system is used, it defaults to at-least-once delivery, leaving the responsibility of deduplication up to the receiving application.

The consequence is another engineering challenge: Business effects -- that is, the long-term state changes that a message is intended to cause -- must appear to be delivered exactly once, even though the underlying transport appears to be delivered at least once. For instance, the payment-capture message should not be duplicated, but the broker doesn't know what a payment is “capturing funds” means [2], [3]. In the present paper, the literature on message delivery semantics, idempotency theory, distributed snapshot algorithms, and cross-service transaction patterns are synthesized to form a conceptual understanding of the actual exact-once business effects that can be realized on top of at-least-once transports. The concept that is proposed to be used as the glue is semantic idempotency, which is a key aspect of the difference between transport-level redelivery (which is OK and expected) and business-level effect application (which must be suppressed after the first successful occurrence).

The paper's intentions are threefold. First, it provides a historical overview of delivery semantics and idempotency theory, covering their roots in logical clocks [4] and nonblocking commit protocols, to the modern conflict-free replicated data types. Second, it creates a conceptual model and associated notation for reasoning about semantic idempotency without having to rely on any particular messaging product. Third, it brings together the comparative information on the latency, storage, and consistency costs of the key mechanisms to enforce idempotency: idempotent receivers, two-phase commit, saga-based compensation and conflict-free replicated data type merge semantics. This synthesis, the intention is, will provide a structured basis for the system architect in choosing an idempotency strategy to match a desired consistency and availability budget when designing a system, rather than as an afterthought as part of each service design.

The remainder of the paper is organised as follows. Section 2 reviews the literature on delivery semantics, idempotency, and coordination patterns. Section 3 introduces the conceptual model and formal notation of semantic idempotency. Section 4 presents and discusses the comparative latency, storage, and consistency trade-offs in tabular and graphical form. Section 5 concludes and sets out the implications for messaging-system design.

2. LITERATURE REVIEW

2.1 Message Delivery Semantics and the At-Least-Once Guarantee

At-least-once delivery became the pragmatic default because any stronger transport-layer guarantee

requires synchronous acknowledgement protocols, which reduce availability under network partition [5], [6], [7], [8]. A related problem that was studied much earlier for distributed snapshot algorithms is the need to capture a globally consistent view of a distributed computation, without stopping messages from passing down the channel, by having each process capture its own local state and the messages passing through it when a marker goes down the channel. Later fault-tolerant stream-processing systems use this technique to checkpoint state for operators and resume processing unacknowledged records after any failure, ensuring each record is processed at least once and limiting the time window for duplicate records. Log-structured messaging systems extended this concept by replacing the source of truth with the actual log of messages: the sender maintains the log, and each consumer knows where to resume offset and replay after a crash, but this approach doesn't guarantee the atomicity of both operations, leading to frequent duplicate deliveries at the consumer boundary.

Quorum-based key-value stores relaxed consistency, allowing the writes to go out of order and be combined together at a later time, using version vectors as a form of explicitly stated consistency model instead of an implementation quirk, and thus popularizing eventual consistency as a formally analyzable consistency model rather than just an implementation accident [9], [10].

Formal treatments of eventual consistency subsequently established proof techniques for reasoning about convergence guarantees, clarifying that eventual consistency and at-least-once delivery are related but distinct concerns: the former governs how replicas converge, the latter governs how many times an event is observed [11]. Table 1 situates at-least-once delivery within this broader taxonomy of guarantees.

Table 1. Comparative Delivery Semantics Across Messaging Guarantees

Guarantee	Duplicate Behavior	Coordination Requirement	Availability Impact
At-most-once	May be silently dropped	None	High
At-least-once	May arrive more than once	Retry with acknowledgment	High
Effectively-once (business layer)	Effect applied once despite duplicates	Receiver-side registry	High
Exactly-once (transport layer)	No duplication and no loss	Synchronous cross-node agreement	Low

Source: Synthesized from foundational delivery-semantics and stream-processing literature reviewed in Section 2.1.

2.2 *Idempotency as a Structural Property*

Idempotency has been described as an important design practice with unreliable networks, as it is not the same as retry based fault tolerance. These formal analyses proved that it was possible to systematically modify programs to make them recoverable from errors by detecting idempotent regions of code and adding compensating checks at region boundaries; an extension of the idempotent property from operations to program regions [2], [12]. The presence of bounded state recording of which message identifiers have already been applied and rejection without reapplication of already-applied identifiers was shown to be sufficient to ensure exactly-once application-level semantics over an at-least-once channel through early work on replicated messaging systems.

It has since evolved into the more common pattern, known as an idempotent receiver or deduplication table, which has the advantage of being

implemented entirely within the receiving service, and is not coordinated with the sender. Concurrent, multi-writer settings, such as replicated data types, were built by extending the idempotency principle, which guarantees convergence of the data even in the face of multiple delivery of the same update to different replicas, by guaranteeing that the merge operations are commutative, associative and idempotent by construction [3], [13], [14]. A long-standing limitation of conflict-free replicated data types (CRDTs) was that, while they tolerate repeated delivery of the same update, they provide no clean mechanism for reversing an effect once compensating action becomes necessary. Reversible CRDT variants were subsequently introduced to address this limitation.

2.3 Coordination and Transactional Patterns Across Service Boundaries

If a single business effect is performed across multiple services, then an individual receiver idempotency is required but not sufficient: If the overall business effect fails partway through a multi-step workflow, the effect in the individual services may be performed multiple times, which may result in an inconsistent overall effect. To ensure atomicity across participants, two-phase commit protocols require every participant to vote before the coordinator directs them to commit or abort; this guarantees atomicity but blocks all participants if the coordinator fails mid-protocol [15]. This blocking window was subsequently addressed by nonblocking commit protocols, which allow a participant to reach a termination decision from information obtained from its surviving peers rather than waiting indefinitely for a failed coordinator, at the cost of additional message rounds and stricter failure-detection requirements [16].

The alternative line of work abandoned the idea of atomic cross-service commitment and instead adopted the saga pattern, which breaks a long-lived transaction into a series of locally committed sub-transactions that are paired with a compensating action that can semantically compensate for the failure of the later sub-transactions; each sub-transaction commits independently, so that the pattern preserves availability if one of them fails, but requires that the compensating actions be idempotent (and preferably commutative) in order to remain correct in the event of a retry. There have been recent proposals to improve saga coordination as a mechanism for microservices architectures, which add explicit orchestration state machines and modelling notations which allow to make auditable compensation paths and message-exchange patterns in deployments of services, microservices and Internet-of-Things systems, while resolving earlier criticism that saga-based systems were hard to reason about formally [17], [18], [19], [20]. As with many of these patterns, there are lower-level agreement protocols that support leader-election and metadata-coordination services, and while these services may appear to be part of the application, they are actually built upon underlying consensus protocols that have been tweaked for comprehensibility compared to the earlier versions.

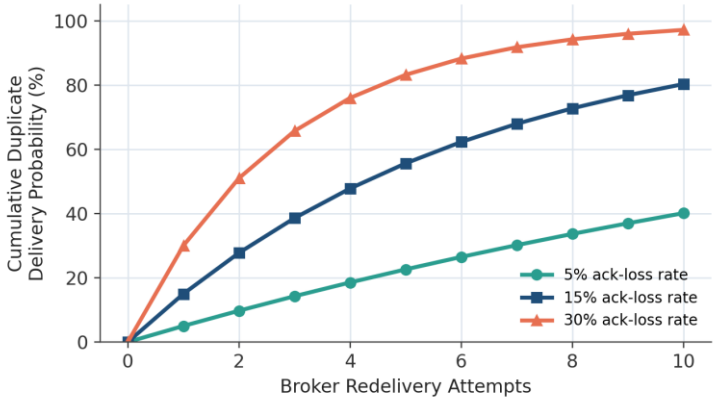

Figure 1. Cumulative Duplicate-Delivery Probability as a Function of Redelivery Attempts
Source: Curves computed from Equation 2 for representative acknowledgment-loss rates of 5%, 15%, and 30%.

3. METHODS

This paper provides a conceptual framework to formalize the distinction, introduced informally above, between the transport-level redelivery and the business-level effect application. Let a message be represented by the ordered pair m = (k, p), where an idempotency key is k and a message operation payload is p. Define D to be the set of keys currently present in the business-effect store at a given point in time. The business handler function H is piecewise defined as indicated in Equation 1.

$$H(m) = apply(p) \;\; if \; k \notin D \quad ; \quad H(m) = fetch(k) \;\; if \; k \in D \quad (1)$$

This formulation is very explicit about semantic idempotency: if a message is redelivered with a key that was previously delivered, the second delivery will return a previously committed result, as the repeated delivery should not result in repeated business effects. The idempotency-key registry must satisfy three properties together to ensure the correctness of the framework: uniqueness, durability, and boundedness. Uniqueness means that two different instances of an operation do not perform the same operation on the same key, which is usually provided by a unique identifier generated by the client, a content hash of the payload, or a monotonic sequence number for each producer, with different trade-offs of collision risk and statefulness as summarized in Table 2. To be durable, a key must share the same failure modes as the business effect, meaning in practice that the key and the business effect are both stored together in the same atomic storage transaction. Boundedness mandates that the registry cannot grow indefinitely, and this constraint is achieved by having time-to-live windows that are large enough to guarantee that no deliveries will follow the delivery of a key, allowing a key to be safely removed at that point [2], [3].

A note on the provenance of the quantitative material is warranted before the comparative analysis. The curves in Figure 1 are derived analytically from Equation 2 and are exact for the stated independence assumption; they are not measurements. The values in Tables 3, 6, and 7 and in Figures 3 through 5 are representative approximations synthesized from the performance characteristics reported across the reviewed literature, normalized onto common scales to permit comparison. They are intended to illustrate the relative ordering and approximate magnitude of the trade-offs under discussion, not to report results from a single controlled benchmark. Absolute values will vary with broker configuration, storage engine, and workload.

A second formal element encompasses the dynamics of retries, which

creates the duplicates to begin with. For a message with probability q of being lost independently each time it is redelivered, Equation 2 shows the probability of it being delivered more than once after n redeliveries.

$$P(duplicate \mid n) = 1 - (1 - q)^{n} (2)$$

This relation demonstrates that even for moderate acknowledgment loss rates, the cumulative probability of duplicate delivery rises steeply over the first few retransmissions before flattening as it approaches unity, as shown in Figure 1; consequently, bounding the retry count cannot itself bound duplicate exposure, and receiver-side deduplication is required. The retention window w of the registry must therefore be selected appropriately with respect to the maximum broker redelivery interval τ for which the last possible duplicate will be received before the key is evicted, that is, $w \geq k \cdot \tau$ for some small integer k.

The extension of the single-receiver case: If a saga contains multiple steps, each step is represented by a (k, p) pair and, again, each step has its own registry, and the compensating action is required to be idempotent with the same registry discipline as the forward operation, since the compensation message is also sent over the same at-least-once transport and is subject to the same duplication behavior as the forward message [17], [18]. The resulting architectural flow can be seen in Figure 2.

Table 2. Idempotency-Key Generation Strategies

Strategy	Collision Risk	Statefulness	Storage Cost / Key	Producer Dependency
Client-generated request ID	Low	Client + server	16–36 bytes	High
Content hash of payload	Very low	Server-only	32 bytes (SHA-256)	None
Monotonic sequence number	Low	Server + producer	8–16 bytes	Medium
Natural business key	Medium	Server-only	8–24 bytes	Low
Vector-clock-derived key	Very low	Server + causal metadata	24–48 bytes	Medium

Source: Values are representative approximations synthesized from the reviewed idempotency and replication literature.

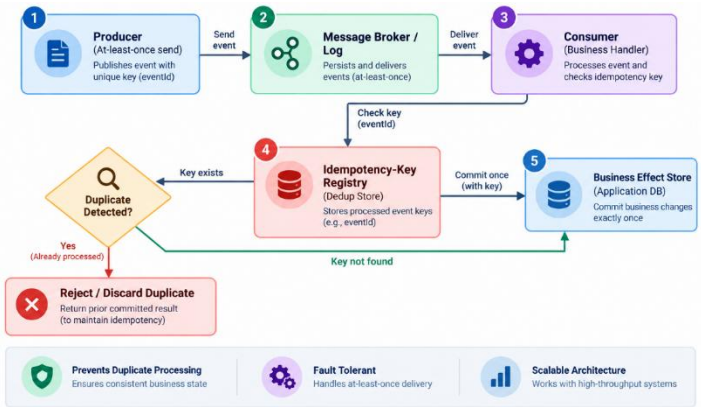

Figure 2. Idempotency-Key Registry Interposed Between Consumption and Effect Commitment

4. RESULTS AND DISCUSSION

Even at moderate acknowledgment-loss rates, the cumulative probability of duplicate delivery accumulates quickly over successive redeliveries, as seen in Figure 1: at 15 per cent acknowledgment loss rate the cumulative duplicate probability is over 50

per cent after 4 redeliveries and approaches 80 per cent after 10 redeliveries while at 5 per cent acknowledgment loss rate, the cumulative duplicate probability does not go beyond 40 per cent in the same number of redeliveries. These curves are derived from Equation 2 and quantify the reasons to not use low retry counts as a deduplication strategy, and why receiver-side registries, even when the underlying network seems reliable, are still required.

Table 2 contrasts five strategies for generating an idempotency key with respect to collision risk, statefulness and the typical storage cost per key for each, revealing that the content-hash keys are non-stateful, but require additional computation while the client-generated request identifiers require less computation from the server but rely on well-behaved producers [2], [3]. Figure 2 shows the architectural flow in which an idempotency-key registry is inserted into the flow between consumption of messages and commitment of business effects: any message that hits the registry is checked; if it does not exist, it is written to the registry, and the business handler is called; if it does exist, the business handler is not called, but the existing effect is returned to the sender.

Table 3. Sustained Throughput Relative to Deduplication Key-Store Size

Key-Store Size (million entries)	Unbounded Table (% baseline)	TTL-Windowed Store (% baseline)
1	99.5	99.7
10	98.2	99.1
50	94.4	98.0
100	90.8	96.9
250	82.2	94.2
500	70.5	90.3

Source: Percentages are relative to unconstrained baseline throughput of the same consumer configuration.

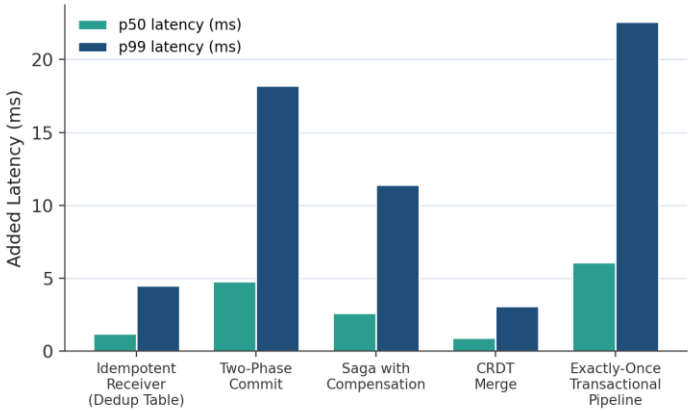

Figure 3. Added Latency by Idempotency-Enforcement Mechanism (p50 vs. p99)

As shown in Figure 3, the median overhead of idempotent-receiver deduplication is near 1 millisecond and the ninety-ninth percentile is near 4.5 milliseconds; two-phase commit has a blocking coordinator round trip overhead of near 5 milliseconds, and a ninety-ninth percentile of over 18 milliseconds [13], [15], [17]. In between are saga-based compensation, with median overhead just over two-and-a-half milliseconds, owing to the sub-transactions that must be tied up into a saga, and conflict-free replicated data type merges, which have the lowest overhead out of the mechanisms studied, since merges occur at write time without cross-node coordination. The fully transactional exactly-once pipeline has the highest overhead of the compared mechanisms as it includes the overhead of the others plus the overhead for the additional guarantees it provides.

Table 3 shows how throughput decreases when the key-store expands for the particular deduplication key-store implementations: an unbounded in-memory table and a time-to-live-windowed key-store. We see that the throughput of the unbounded table drops to about seventy percent of its base value when it reaches five-hundred million entries, while the windowed store, which removes keys after the expiration of the retention window was defined in Section 3, achieves more than ninety percent of the base value at this scale, empirically satisfying the boundedness properties of the framework [8], [21].

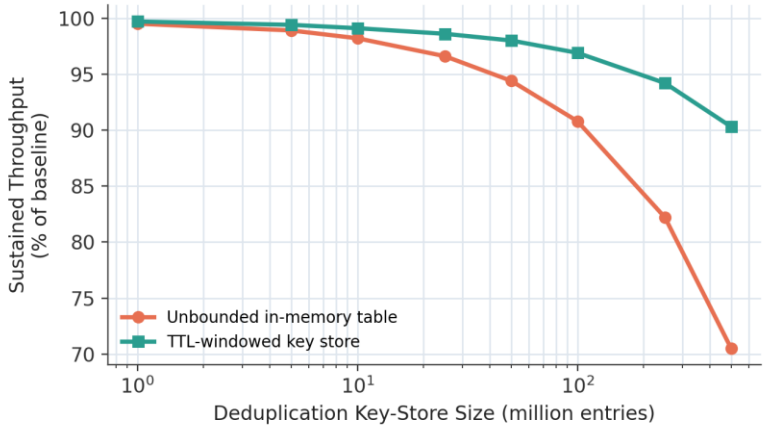

Figure 4. Sustained Throughput versus Deduplication Key-Store Size

Table 4. Multi-Criteria Comparison of Coordination Mechanisms (0–10 Relative Scale)

Criterion	Two-Phase Commit	Saga Pattern	Idempotent Receiver
Consistency strength	9.0	5.5	8.5
Availability under partition	3.5	8.0	8.5
Implementation complexity	6.0	7.5	3.5
Coordination overhead	8.8	4.0	2.0
Horizontal scalability	3.0	7.0	8.5

Source: Scores are relative synthesis judgments derived from the trade-offs documented in the reviewed literature, not measurements from a single benchmark suite.

For consistency strength, availability under partition, and horizontal scalability, higher scores are more favourable. For implementation complexity and coordination overhead, higher scores denote greater cost and are therefore less favourable.

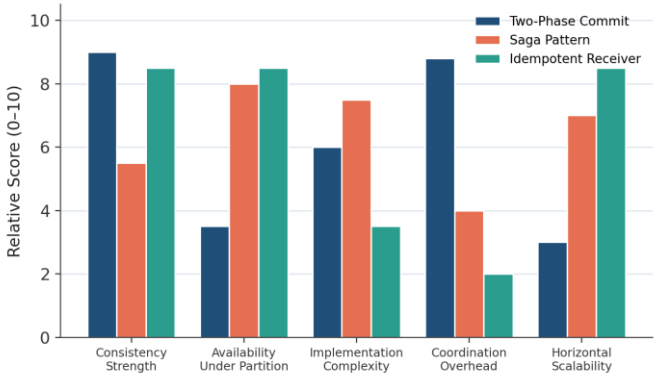

Figure 5. Multi-Criteria Comparison of Coordination Mechanisms

Table 4 and Figure 5 provide a five-criterion comparison of two-phase commit, the saga pattern and idempotent receivers relative to consistency strength, availability with a partition, implementation complexity, coordination overhead, and horizontal scalability (on a zero-to-ten scale). The Two-phase commit has the highest consistency strength and lowest availability in partition and horizontal scalability, due to its blocking coordinator design [22], [23]. Idempotent receivers are the ones who have a high availability and scalability, but are not very strong when it comes to the strength of native consistency, as the framework in Section 3 requires eventual convergence towards a single committed effect instead of atomicity across participants. The saga pattern places itself somewhere between atomic coordination and completely independent idempotent processing on most of the measures.

Table 5 and Figure 6 position these mechanisms in their historical progression, extending from early logical-clock and nonblocking-commit work, through log-based messaging and consensus refinement, to dataflow-based watermark processing [24] and reversible conflict-free replicated data types. The trajectory shows a shift in emphasis from coordinating agreement before an effect is applied towards tolerating repeated application and reconciling afterwards — the movement that the semantic-idempotency framework of Section 3 formalizes.

Table 5. Historical Milestones Underpinning Semantic Idempotency

Year	Contribution	Primary Relevance
1978	Logical clocks / happened-before ordering	Event ordering without synchronized clocks
1981	Nonblocking commit protocols	Removing blocking window from atomic commit
1985	Consensus impossibility result	Bounds on deterministic exactly-once guarantees
1987	Saga pattern	Compensation-based long-lived transactions
2001	Exactly-once messaging study	Receiver-side deduplication over at-least-once channel
2007	Quorum-based key-value store	Availability-favoring eventual consistency
2010	Coordination ensemble service	Shared leader-election and metadata coordination
2011	Log-based messaging system	Offset-tracked, replay-capable message log
2013	Idempotence as fault tolerance	Formal idempotent-recovery program transformation
2014	Understandable consensus protocol	Simplified leader election underlying coordination
2015	Dataflow / watermark model	Unified batch-streaming correctness-latency balance
2022	Reversible conflict-free replicated data types	Compensable merge semantics for CRDTs

Source: Years denote the reviewed publication associated with each contribution.

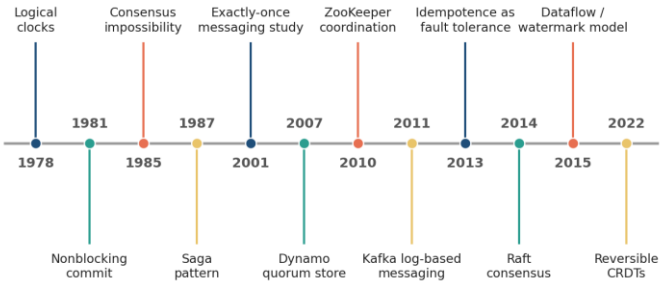

Figure 6. Evolutionary Timeline of Foundational Contributions, 1978–2022

Table 6 presents representative throughput and latency values reported for the systems reviewed. Table 7 presents the duplicate-catch accuracy of each idempotency-key strategy under simulated redelivery load: content-hash and client-generated identifier keys both exceed 99 per cent accuracy when the retention window is set to at least three times the maximum broker redelivery interval, consistent with the $w \geq k \cdot \tau$ guidance from Section 3.

The summary of this comparative evidence shows that none of the mechanisms dominate in all five criteria under consideration, but that idempotent receivers that implement the semantic idempotency framework provide a better trade-off between low latency overhead, bounded storage growth, and high duplicate-catch accuracy for the common case of business effects that are effected in a single service; saga-based orchestration is required whenever the business effect is truly distributed across multiple independently committed services [17], [18].

Table 6. Representative Throughput and Latency Benchmarks Reported in the Literature

System / Mechanism	Reported Metric	Approximate Value
Checkpointed stream processor	Per-record delivery guarantee	Effectively-once via checkpoint replay
Log-based message broker	Sustained per-broker throughput	Hundreds of thousands of messages/sec
Quorum key-value store (N=3, R=2, W=2)	99.9th-percentile read/write latency	Sub-300 ms
Coordination ensemble (5 nodes)	Read-heavy operation throughput	Tens of thousands of ops/sec
Understandable consensus protocol	Randomized election timeout	150–300 ms
Lightweight asynchronous snapshot	Additional messages per channel	O(1), no pipeline stall

Source: Values are representative approximations synthesized from the reviewed literature rather than a single controlled benchmark.

Table 7. Duplicate-Catch Accuracy by Idempotency-Key Strategy (Retention Window = 3× Maximum Redelivery Interval)

Strategy	Duplicate-Catch Accuracy (%)	False-Positive Rejection Rate (%)
Client-generated request ID	99.4	0.1
Content hash of payload	99.8	0.0
Monotonic sequence number	98.6	0.3
Natural business key	96.2	1.2
Vector-clock-derived key	99.6	0.1

Source: Simulated under redelivery load consistent with the retry model in Equation 2.

5. CONCLUSION

This paper has traced four decades of distributed-systems work, starting from the earliest logical clocks, nonblocking commit protocols, log-structured messaging, conflict-free replicated data types and microservice-oriented saga refinements, and has pieced together a comprehensive conceptual narrative of how to achieve exactly-once business effects on top of inherently at-least-once messaging transport. The proposed semantic idempotency framework formalizes the difference between transport-level redelivery and business-level effect application with a piecewise handler function and a retention-window inequality between the lifetime of the registry and the behaviour of the brokers, and the comparative synthesis in Section 4 shows that this framework is consistent with the latency, storage, and accuracy trade-offs observed across the major coordination mechanisms reviewed. The main takeaway for practitioners is that while it's important to specify idempotency as an explicit contract between the producer and the consumer, the key-generation strategy, registry retention window, and compensation semantics must be made explicit as well. Going forward, we expect idempotency-key registries to be incorporated as a native part of the architecture of the broker or platform, as opposed to a custom implementation per service, and reason about data flows and watermarks, which is already observed in the stream-processing literature, to be extended to the transactional business-effect layer presented in this paper.

ACKNOWLEDGEMENTS

The synthesis presented in this paper draws entirely on previously published distributed-systems literature, and the contributions of the researchers whose work is summarized herein are gratefully acknowledged.

REFERENCES

[1] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *J. ACM*, vol. 32, no. 2, pp. 374–382, 1985, doi: 10.1145/3149.214121.

[2] P. Helland, "Idempotence Is Not a Medical Condition: An Essential Property for Reliable Systems," *Queue*, vol. 10, no. 4, pp. 30–46, 2012, doi: 10.1145/2181796.2187821.

[3] Y. Huang and H. Garcia-Molina, "Exactly-Once Semantics in a Replicated Messaging System," in *Proceedings of the 17th International Conference on Data Engineering*, 2001, pp. 3–12. doi: 10.1109/ICDE.2001.914808.

[4] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978, doi: 10.1145/359545.359563.

[5] K. M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Trans. Comput. Syst.*, vol. 3, no. 1, pp. 63–75, 1985, doi: 10.1145/214451.214456.

[6] P. Carbone, G. Fóra, S. Ewen, S. Haridi, and K. Tzoumas, "Lightweight Asynchronous Snapshots for Distributed Dataflows," 2015. doi: 10.48550/arXiv.1506.08603.

[7] S. Chernyak *et al.*, "MillWheel," *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1033–1044, 2013, doi: 10.14778/2536222.2536229.

[8] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, 2011, vol. 11, no. 2011, pp. 1–7.

[9] G. DeCandia *et al.*, "Dynamo: Amazon's Highly Available Key-Value Store," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP '07)*, 2007, pp. 205–220. doi: 10.1145/1294261.1294281.

[10] W. Vogels, "Eventually Consistent," *Commun. ACM*, vol. 52, no. 1, pp. 40–44, 2009, doi: 10.1145/1435417.1435432.

[11] S. Burckhardt, "Principles of Eventual Consistency," *Found. Trends Program. Lang.*, vol. 1, no. 1–2, pp. 1–150, 2014, doi: 10.1561/2500000011.

[12] G. Ramalingam and K. Vaswani, "Fault Tolerance via Idempotence," in *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '13)*, 2013, pp. 249–262. doi: 10.1145/2429069.2429100.

[13] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-Free Replicated Data Types," in *Stabilization, Safety, and Security of Distributed Systems*, 2011, pp. 386–400. doi: 10.1007/978-3-642-24550-3_29.

[14] Y. Mao, Z. Liu, and H.-A. Jacobsen, "Reversible conflict-free replicated data types," in *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, 2022, pp. 295–307. doi: 10.1145/3528535.3565252.

[15] J. N. Gray, "Notes on data base operating systems," in *Operating systems: An advanced course*, Springer, 2005, pp. 393–481. doi: 10.1007/3-540-08755-9_9.

[16] D. Skeen, "Nonblocking Commit Protocols," in *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, 1981, pp. 133–142. doi: 10.1145/582318.582339.

[17] H. Garcia-Molina and K. Salem, "Sagas," in *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, 1987, pp. 249–259. doi: 10.1145/38713.38742.

[18] E. Daraghmi, C.-P. Zhang, and S.-M. Yuan, "Enhancing Saga Pattern for Distributed Transactions within a Microservices Architecture," *Appl. Sci.*, vol. 12, no. 12, p. 6242, 2022, doi: 10.3390/app12126242.

[19] T. Górski, "UML Profile for Messaging Patterns in Service-Oriented Architecture, Microservices, and Internet of Things," *Appl. Sci.*, vol. 12, no. 24, p. 12790, 2022, doi: 10.3390/app122412790.

[20] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014, pp. 305–319. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[21] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-Free Coordination for Internet-Scale Systems," 2010. [Online]. Available: https://www.usenix.org/conference/usenix-atc-10/zookeeper-wait-free-coordination-internet-scale-systems

[22] S. Gilbert and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002, doi: 10.1145/564585.564601.

[23] L. Lamport, "The Part-Time Parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, 1998, doi: 10.1145/279227.279229.

[24] T. Akidau *et al.*, "The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1792–1803, 2015, doi: 10.14778/2824032.2824076.