2026 Volume: 02 Issue: 09 www.ijrpa.com ISSN 2456-9995 Research Article

International Journal Research Publication Analysis

Page: 01-11

AN EMPIRICAL COMPARISON OF LAMPORT, VECTOR, AND HYBRID LOGICAL CLOCKS UNDER MULTI-PARTITION FAN-OUT IN APACHE KAFKA

***Kushagra Srivastava, Abhay Nimmagadda, Harsh Vardhan Singh**

India.

Article Received: 23 July 2026
Article Revised: 11 August 2026
Published on: 01 September 2026

***Corresponding Author: Kushagra Srivastava**
India.

DOI: https://doi-doi.org/101555/ijrpa.5908

ABSTRACT

Event-streaming platforms such as Apache Kafka guarantee ordering only within a single partition, yet production pipelines routinely fan events out across many partitions, where causal relationships between events are silently lost. Logical clocks restore causal reasoning, but their relative cost and correctness under realistic streaming conditions have not been quantified. This paper presents causality-bench, a benchmark harness that empirically compares Lamport clocks, vector clocks, and hybrid logical clocks (HLC), together with a Kafka-offset baseline, across five research questions: header overhead, CPU cost, skew-induced mis-ordering, stream-join correctness, and causal-graph replay fidelity. Every experiment runs on two transports, an in-process simulator and a real single-node Kafka (KRaft) broker, allowing transport effects to be isolated. Vector clocks are the only mechanism that maintains 0% mis-ordering at every skew level and achieves recall = precision = 1.000 in causal replay at every partition count, at a header cost that grows linearly with node count. The central empirical finding is that HLC's safe operating window relative to Lamport clocks is transport-dependent: HLC mis-orders fewer event pairs than Lamport only below approximately 375 ms of injected clock skew in-process, but only below approximately 60 ms over a real broker, a roughly six-fold narrowing caused by real delivery jitter. These results provide practitioners with measured decision thresholds for selecting a causality mechanism in partitioned streaming systems.

KEYWORDS: logical clocks; hybrid logical clock; vector clock; Apache Kafka; causal ordering; distributed systems benchmarking; stream processing.

I. INTRODUCTION

Apache Kafka [5] has become the de facto substrate for event-driven architectures, but its ordering guarantee is deliberately narrow: events are totally ordered within a partition and unordered across partitions. When a producer fans a causally related sequence of events out across multiple partitions, downstream consumers observe an interleaving in which the happens-before relation [1] is no longer recoverable from delivery order alone. Applications that join streams, reconstruct audit trails, or replay event histories must therefore attach an explicit causality mechanism to every message, and the three standard candidates are Lamport clocks [1], vector clocks [2], [3], and hybrid logical clocks (HLC) [4].

The qualitative trade-offs among these mechanisms are textbook material: Lamport clocks are compact but cannot detect concurrency, vector clocks capture causality exactly but grow with the number of nodes, and HLC combines physical timestamps with a bounded logical counter under a clock-synchronization assumption. What practitioners lack are measured thresholds: at what skew does HLC's physical-time dependence begin to cost correctness, how much header and CPU overhead does a vector clock actually impose in a Kafka pipeline, and how often do compact clocks produce wrong answers in concrete tasks such as stream joins and causal replay. Prior evaluations are either analytical or tied to database internals rather than partitioned log transports.

This paper answers those questions empirically. We present *causality-bench*, an open benchmark harness that instruments four timestamping mechanisms (Lamport, Vector, HLC, and a Kafka-offset baseline, OffsetClock) with an identical event workload and evaluates them under multi-partition fan-out. The contributions are: (1) a five-experiment benchmark design mapping one experiment to each of five research questions covering cost (header bytes, CPU) and correctness (skew-induced mis-ordering, false stream joins, causal replay fidelity); (2) a dual-transport methodology that runs every experiment both in-process and against a real Kafka broker, so that transport-induced effects are measured rather than assumed; (3) measured decision thresholds, most notably that HLC's safe operating window relative to Lamport is transport-dependent, approximately 375 ms of clock skew in-process versus approximately 60 ms over a real broker; and

(4) exact correctness invariants for vector clocks in this setting, 0% mis-ordering at all skew levels, 0% false joins on concurrent pairs, and recall = precision = 1.000 in causal replay at every partition count.

The remainder of the paper is organized as follows. Section II reviews related work. Section

III defines the system model and the four clock mechanisms. Section IV describes the benchmark design and Section V the experimental setup. Section VI reports results per research question, Section VII discusses the transport-dependence finding, Section VIII addresses threats to validity, and Section IX concludes.

II. Related Work

Lamport introduced logical clocks and the happens-before relation as the foundation of event ordering in asynchronous

systems [1]. Fidge [2] and Mattern [3] independently developed vector clocks, which characterize the causal partial order exactly at the cost of one counter per process. Schwarz and Mattern surveyed the fundamental limits of causality tracking and showed that no mechanism smaller than a vector can characterize causality precisely in the general case [7], a bound our replay-fidelity results illustrate empirically.

Hybrid logical clocks were proposed by Kulkarni et al. [4] to combine the monotonicity of logical clocks with the meaningfulness of physical timestamps while keeping the logical component bounded under bounded clock skew. HLC underpins commercial systems including distributed SQL databases, and Google's Spanner demonstrates the alternative of engineering skew away with dedicated hardware and the TrueTime API [6]. Our work does not modify HLC; it measures where its skew assumption breaks in a partitioned log setting, on commodity infrastructure without engineered time.

Kafka itself was introduced by Kreps et al. [5], and its per-partition ordering model is documented in the project literature [8]. Benchmarking work around Kafka has concentrated on throughput and latency of the broker; to our knowledge no prior study benchmarks causality mechanisms themselves across a real broker transport under controlled skew, fan-out, and out-of-order delivery, which is the gap this paper addresses.

III. System Model and Clock Mechanisms

A. *System Model*

We model a set of n producer nodes emitting events to a Kafka topic with p partitions. Each event carries an application payload plus a clock header serialized into Kafka record headers. Causal dependencies arise from two sources: program order within a producer, and explicit cause-effect edges when an event is emitted in reaction to a consumed event. Kafka delivers records in offset order within a partition but with no cross-partition guarantee, so a consumer reading multiple partitions observes an arbitrary interleaving of causally related events.

Ground-truth causality is recorded by the workload generator as the transitive closure of program-order and message edges, against which each mechanism's verdicts are scored.

B. Clock Mechanisms Compared

1) Lamport Clock: a single 64-bit counter incremented on each local event and set to max(local, received) + 1 on receive [1]. Comparison is strict integer comparison with no node-identifier tie-break; equal counters are reported as not causally related. It imposes a total order consistent with causality but cannot distinguish causal from concurrent pairs.

2) Vector Clock: one counter per node, merged component-wise on receive [2], [3]. Its comparison defines a true partial order: it decides causal precedence exactly and reports concurrency explicitly. Header size grows linearly with the number of nodes.

3) Hybrid Logical Clock (HLC): a pair (l, c) where l tracks the maximum physical timestamp observed and c is a bounded logical counter that breaks ties [4]; comparison is lexicographic on (l, c). Correctness of its ordering relative to real causality degrades as physical clock skew between nodes grows.

4) OffsetClock (Baseline): the Kafka (partition, offset) pair treated as a timestamp. It is included to answer the natural reviewer and practitioner question of why the broker's own ordering metadata does not suffice: offsets order events within a partition perfectly but are blind to all cross-partition causality.

IV. Benchmark Design

A. *Research Questions*

The benchmark evaluates five research questions. RQ1 (header overhead): how many bytes does each mechanism add per message, and how does this scale with node count? RQ2 (CPU cost): what is the per-operation cost of tick, merge, and serialize for each clock? RQ3 (skew sensitivity): how does injected physical clock skew translate into mis-ordered event pairs for each mechanism? RQ4 (join correctness): in a windowed stream join, how often does each mechanism assert a causal join between events that are in fact concurrent, and how often does it miss true causal pairs? RQ5 (replay fidelity): when a consumer reconstructs the causal graph from timestamps alone, what recall and precision does each mechanism achieve against ground truth as the partition count grows?

B. Experiments

One experiment addresses each research question. Experiment 1 measures serialized header sizes and overhead relative to a 1 KiB payload. Experiment 2 microbenchmarks clock

operations with warm-up and reports medians over repeated trials. Experiment 3 sweeps injected inter-node clock skew across a grid of levels and measures the percentage of sampled event pairs whose clock order contradicts ground-truth causal order; Vector-clock concurrency verdicts are scored against ground-truth concurrency. Experiment 4 constructs pairs of events that are provably concurrent (via persistent node clocks with shared-ancestor synchronization) and measures each mechanism's false-join rate, alongside missed causal pairs under an out-of-order delivery sweep. Experiment 5 replays the full event stream at p = 2, 4, 8, and 16 partitions, reconstructs the causal graph from each mechanism's timestamps, and scores edge recall and precision against the ground-truth transitive closure.

C. Dual-Transport Methodology

Every experiment runs on two transports: an in-process simulator, which delivers events deterministically and isolates the clocks' intrinsic behavior, and a real single-node Kafka broker in KRaft mode, which adds genuine produce latency, batching, and delivery jitter. The purpose is diagnostic: any threshold that moves between transports is attributable to transport dynamics rather than to the clock algorithms, and Section VI-C shows that the paper's central threshold does move, substantially.

V. Experimental Setup

The harness is implemented in Python using the confluent-kafka client; the broker is a single-node Apache Kafka instance in KRaft mode provisioned via Docker Compose. All results are reported as means over three random seeds unless stated otherwise, and a suite of nine gate tests validates the clock implementations (comparison semantics, merge rules, serialization round-trips) before any experiment runs; all nine pass on both transports.

For Experiment 3, the skew grid over the broker transport uses eleven levels (0, 50, 75, 100, 125, 150, 200, 500, 1000, 2000, and 5000 ms), with the 75, 125, and 150 ms points added specifically to pin the HLC-versus-Lamport crossover by direct measurement rather than interpolation. The in-process grid brackets its own crossover with a fine grid over 250 to 400 ms. Experiment 5 uses partition counts of 2, 4, 8, and 16 with a fixed event budget per run. In-process and broker runs of Experiments 3 and 5 use different event-count scales (for example, 2000 versus 500 events in Experiment 3), so their absolute magnitudes are compared qualitatively while thresholds are reported per transport (Section VIII).

VI. RESULTS

A. *RQ1: Header Overhead*

Table I reports serialized header sizes. Lamport and OffsetClock are constant at 8 and 12 bytes respectively, and HLC is constant at 22 bytes. The vector clock grows linearly with node count as expected, reaching 202 bytes at 25 nodes and 802 bytes at 100 nodes, 78.3% of a 1 KiB payload. Header sizes are deterministic and identical across both transports.

TABLE I: SERIALIZED HEADER SIZE AND OVERHEAD VS. A 1 KIB PAYLOAD.

Mechanism	Header (B)	Overhead (%)
Lamport	8	0.78
HLC	22	2.15
Offset	12	1.17
Vector (n=25)	202	19.7
Vector (n=100)	802	78.3

B. *RQ2: CPU Cost*

Table II reports in-process median per-operation costs at n
= 100 nodes. Merge is the discriminating operation: Lamport and HLC merges are constant-time at 262 ns and 763 ns respectively, while the vector merge costs 27,212 ns, reflecting its O(n) component-wise maximum. Vector serialization likewise grows from 1,286 ns at 10 nodes to 16,060 ns at 100 nodes, whereas the O(1) clocks show no systematic growth. Broker-side produce and round-trip latencies are excluded from the reported CPU metrics: a 44 ms produce-time outlier observed for the vector configuration at n = 100 was diagnosed as a broker flush/metadata stall unrelated to header size (802 bytes cannot account for tens of milliseconds), and such transport noise would otherwise be misattributed to clock cost.

TABLE II: MEDIAN PER-OPERATION COST, IN-PROCESS, N = 100 NODES.

Mechanism	Merge (ns)	Serialize (ns)
Lamport	262	O(1), noise-level
HLC	763	O(1), noise-level
Vector	27,212	16,060

C. *RQ3: Skew-Induced Mis-Ordering*

Table III is the paper's headline result. Three qualitative regimes are visible and hold on both transports. First, the vector clock mis-orders exactly 0.00% of pairs at every skew level, while additionally reporting 50.3–51.8% of sampled pairs as concurrent, a correct abstention rather than an error. Second, Lamport mis-ordering is flat and skew-independent (1.87% in-process, 1.32% on the broker): it ignores physical time entirely, so its error is a fixed property of the

workload's concurrency structure. Third, HLC mis-ordering rises monotonically with skew, from 0% at zero skew to 17.7% (in-process) and 26.4% (broker) at 5000 ms; its logical counter remained bounded ($c \leq 2$) throughout, confirming that the errors stem from the physical component, not counter overflow.

The crossover between HLC's rising curve and Lamport's flat line defines HLC's safe operating window. Over the real broker it is measured directly on the fine grid: HLC is ahead of Lamport at 50 ms (1.02% vs. 1.32%) and behind at 75 ms

(1.64% vs. 1.32%), placing the crossover at approximately 60 ms. In-process, HLC is ahead at 200 ms and behind at 500 ms, and linear interpolation places the crossover at approximately 375 ms (the 250–400 ms fine-grid confirmation run is noted in Section VIII). Real delivery jitter therefore narrows HLC's safe window by roughly a factor of six.

TABLE III: MIS-ORDERED PAIRS (%) VS. INJECTED CLOCK SKEW, BOTH TRANSPORTS.

Skew (ms)	Lam. in-proc	Lam. Kafka	HLC in-proc	HLC Kafka	Vector (both)
0	1.87	1.32	0.01	0.00	0.00
50	1.87	1.32	0.29	1.02	0.00
75	—	1.32	—	1.64	0.00
100	1.87	1.32	0.56	1.90	0.00
125	—	1.32	—	2.46	0.00
150	—	1.32	—	2.92	0.00
200	1.87	1.32	1.02	3.84	0.00
500	1.87	1.32	2.48	9.22	0.00
1000	1.87	1.32	4.95	16.21	0.00
2000	1.87	1.32	8.87	24.44	0.00
5000	1.87	1.32	17.66	26.41	0.00

D. RQ4: Stream-Join Correctness

Experiment 4 evaluates each mechanism on provably concurrent event pairs presented to a windowed join. The result is fully discriminating, as summarized in Table IV: the vector clock asserts a causal join on 0% of concurrent pairs, whereas Lamport and HLC assert false joins on approximately 100% of them, an inherent consequence of scalar and physical timestamps imposing a total order that cannot represent concurrency. OffsetClock exhibits the complementary failure, missing 100% of cross-partition causal pairs because offsets carry no cross-partition information. In-process and broker runs of this experiment agree within 1–2 percentage points across the entire out-of-order delivery sweep, making it the experiment with the tightest cross-transport agreement.

TABLE IV: JOIN CORRECTNESS ON CONCURRENT AND CROSS-PARTITION CAUSAL PAIRS.

Mechanism	False joins on concurrent pairs (%)	Missed cross-partition causal pairs (%)
Lamport	≈100	0
HLC	≈100	0
Vector	0	0
Offset	0	100

E. *RQ5: Causal Replay Fidelity*

Experiment 5 reconstructs the causal graph from timestamps alone and scores it against the ground-truth transitive closure at each partition count. Several quantities are exact invariants across all partition counts and are stated once: Lamport, HLC, and Vector recall are all 1.000; Offset precision is 1.000; and Vector precision is 1.000. Vector is therefore the only mechanism achieving recall = precision =
1.000 at every partition count. The quantities that vary with partition count appear in Table V. Offset recall falls from 0.58 at 2 partitions to 0.28–0.29 at 8–16 partitions as an increasing share of causality crosses partition boundaries that offsets cannot see. Lamport and HLC precision fall in near lock-step from approximately 0.87 to 0.22, meaning roughly 13% of their causal claims are false at 2 partitions and roughly 78% at
16. The vector clock's cost for its exactness is the per-event byte growth shown in the final row, from 10 bytes at 2 partitions to 66 bytes at 16.

In-process, where delivery is orderly, HLC tolerates roughly 375 ms of skew before mis-ordering more pairs than a skew-oblivious Lamport clock; over a real broker, produce batching and delivery jitter interact with skew such that the same crossover occurs near 60 ms. The practical consequence is direct: an HLC deployment validated in simulation or in a low-jitter environment can silently cross into its unsafe regime on production infrastructure whose NTP error would appear comfortably acceptable on paper. Skew tolerance for HLC must therefore be characterized on the deployment transport, not assumed from the algorithm's analysis.

B. The Price of Causality

Experiments 1 and 2 quantify the denominator of every trade-off in this paper. The vector clock's exactness (Tables III–V) is bought with linear header growth (802 bytes at 100 nodes) and an O(n) merge (27,212 ns versus 262 ns for Lamport at n = 100). Whether that price is acceptable is workload-dependent: at 16 partitions the vector header is 66 bytes per event, 6.4% of a 1 KiB payload, in exchange for eliminating a 78% false-claim rate in causal

replay. For join-heavy or audit-critical pipelines the exchange favors the vector clock decisively; for high-fan-out telemetry where approximate ordering suffices, Lamport's fixed 8 bytes and flat, predictable error rate are attractive.

C. Practitioner Guidance

The measured thresholds support concrete selection rules. Where concurrency detection is required (stream joins, causal replay, audit reconstruction), only the vector clock is correct: Lamport and HLC falsely join essentially all concurrent pairs, and offsets miss all cross-partition causality. Where compactness is required and clock infrastructure is well controlled, HLC is preferable to Lamport only while transport-measured skew remains below the deployment's crossover, approximately 60 ms on the commodity single-broker configuration measured here. Where skew is uncontrolled or unmeasured, Lamport's skew-independence makes it the predictable compact choice. Offsets alone are never sufficient for cross-partition causal reasoning, though their perfect precision makes them a sound intra-partition tiebreaker.

TABLE V: REPLAY FIDELITY VS. PARTITION COUNT (IN-PROCESS, MEAN OF 3 SEEDS)

Partitions	2	4	8	16
Offset recall	0.58	0.37	0.28	0.29
Lamport precision	0.87	0.68	0.44	0.22
HLC precision	0.86	0.67	0.44	0.22
Vector bytes/event	10	18	34	66

VII. DISCUSSION

A. The Transport-Dependent Safe Window of HLC

The five experiments read as one argument: compact clocks purchase their compactness with assumptions, and the benchmark measures exactly where each assumption fails. For HLC the assumption is bounded clock skew, and the central finding is that the failure threshold is not a property of the algorithm alone but of the algorithm-plus-transport system.

VIII. THREATS TO VALIDITY

A. *Internal Validity*

Broker-side latency metrics contain transport noise; a 44 ms produce stall observed in one vector-clock configuration was traced to broker flush behavior rather than header cost and excluded from CPU results, and all CPU claims rest on the in-process microbenchmark. The in-process crossover of approximately 375 ms in Experiment 3 is interpolated between

measured points at 200 ms and 500 ms; the bracketing fine-grid run (250–400 ms) mirrors the broker-side fine grid that pinned the 60 ms crossover by direct measurement. Stale bytecode caches were identified as a
hazard during development and are cleared by the harness before every run.

B. *Construct Validity*

In Experiment 5 the ground-truth denominator (transitive-closure edge count) shrinks from approximately 70,000 edges at 2 partitions to approximately 17,000 at 16, because a fixed event budget spread over more partitions yields shorter program-order chains and closure size grows super-linearly with chain length. Direct edge counts remain flat (approximately 480) across partition counts, supporting the generator itself is partition-independent; recall and precision are ratios and remain comparable across partition counts, but absolute edge counts should not be compared across columns.

C. *External Validity*

The broker transport is a single-node KRaft deployment driven by a Python client; multi-broker replication, cross-datacenter links, and other client runtimes may shift the measured crossover further, most plausibly downward as jitter increases. In-process and broker runs of Experiments 3 and 5 additionally use different event-count scales, so cross-transport comparisons in those experiments are qualitative (regime shapes and per-transport thresholds) rather than point-for-point; Experiment 4, which uses a scale-robust proportion metric, agrees across transports within 1–2 percentage points. All mechanisms share the same workload, seeds, and scoring within each transport, so within-transport comparisons are unaffected.

IX. CONCLUSIONS

This paper presented causality-bench, a dual-transport benchmark of Lamport, vector, and hybrid logical clocks under multi-partition fan-out in Apache Kafka. The vector clock is empirically exact in this setting, 0% mis-ordering at all skew levels, 0% false joins on concurrent pairs, and recall
= precision = 1.000 in causal replay at every partition count, at measured linear costs in header bytes and merge time. Lamport clocks exhibit a flat, skew-independent error floor, and HLC's error rises monotonically with skew. The central finding is that HLC's safe operating window relative to Lamport is transport-dependent, approximately 375 ms in-process versus approximately 60 ms over a real broker, so the choice of a compact causality mechanism cannot be made from algorithmic analysis alone.

Future work proceeds in three directions: adding a probabilistic mechanism (a Bloom clock)

as a fourth point on the cost-exactness frontier, which would make the boundary region between compact and exact clocks non-trivial; completing the in-process fine-grid skew run to convert the interpolated 375 ms crossover into a measured one; and extending the broker transport to multi-node, replicated, and cross-datacenter configurations to map how the HLC crossover moves with delivery jitter.

REFERENCES

1. L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.
2. C. J. Fidge, "Timestamps in message-passing systems that preserve the partial ordering," in *Proc. 11th Australian Computer Science Conf.*, 1988, pp. 56–66.
3. F. Mattern, "Virtual time and global states of distributed systems," in *Parallel and Distributed Algorithms*, M. Cosnard et al., Eds. Amsterdam, The Netherlands: North-Holland, 1989, pp. 215–226.
4. S. S. Kulkarni, M. Demirbas, D. Madappa, B. Avva, and M. Leone, "Logical physical clocks," in *Proc. 18th Int. Conf. Principles of Distributed Systems (OPODIS)*, 2014, pp. 17–32.
5. J. Kreps, N. Narkhede, and J. Rao, "Kafka: a distributed messaging system for log processing," in *Proc. 6th Int. Workshop on Networking Meets Databases (NetDB)*, 2011.
6. J. C. Corbett et al., "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, pp. 1–22, Aug. 2013.
7. R. Schwarz and F. Mattern, "Detecting causal relationships in distributed computations: In search of the holy grail," *Distrib. Comput.*, vol. 7, no. 3, pp. 149–174, 1994.
8. (2026) The Apache Kafka website. [Online]. Available: https://kafka.apache.org/