Designing In-Memory Distributed Systems for Billion-Row Healthcare Data Processing

Amirtharaj Sathees Chandran
Principal Software Development Engineer, USA

ABSTRACT
Processing patient-level healthcare data at national or global scale forces distributed systems to sustain billions of rows while still meeting the latency and operational expectations of commercial analytics products. This paper presents the design and evaluation of a real-time, in-memory computing cluster engineered for complex cohort workloads over massive claims and reference pipelines—on the order of 100+ billion rows ingested through a modern extract-transform-load (ETL) stack. Where prior work often optimises a single dimension, this study foregrounds the *joint* balancing of three competing operational forces: memory footprint, fault tolerance, and ingest throughput. The architecture combines stream-oriented ingestion and orchestration patterns (Scala/Java, Apache Spark, Apache Kafka, PostgreSQL) with Kubernetes-based operations and a deliberately cloud-agnostic footprint, so that the same capability runs for different clients and regions without re-platforming. Using a design-science methodology, the artefact is evaluated through production telemetry and controlled fault-injection experiments. Results show that in-memory execution converts iterative join-and-window cohort logic from a disk-bound bottleneck into a memory-bound, tunable workload; that checkpoint-and-replay fault tolerance restores state within bounded recovery times under node loss; and that back-pressure-aware streaming stabilises ingest throughput under bursty load. The study quantifies the memory–fault-tolerance–throughput trade-off surface, identifies a garbage-collection-driven latency inflection, and derives capacity-planning heuristics. Contributions include a reference design for billion-row in-memory healthcare processing, an empirical characterisation of its operational envelope, and transferable engineering lessons for cloud-agnostic, multi-region deployment. Limitations include single-organisation, observational measurement, which constrains causal and external generalisation. The work advances distributed-systems practice by demonstrating how streaming orchestration and in-memory compute can meet product-grade latency at healthcare scale.

Keywords: in-memory computing; distributed systems; healthcare data; fault tolerance; ingest throughput; Apache Spark; Kafka; Kubernetes

DOI: 10.21590/ijhit.07.03.34

I. INTRODUCTION

Background context. Healthcare analytics products increasingly operate at national and global scale, where a single client engagement may span billions of patient-level rows across claims, eligibility, and reference dimensions (Chen et al., 2020). Unlike offline research pipelines, commercial products carry hard operational expectations—predictable latency, high availability, and graceful degradation—that turn scale from an academic challenge into an engineering contract (Kleppmann, 2017).

Problem statement. Sustaining billions of rows through a real-time pipeline exposes a three-way tension. Holding working sets in memory accelerates iterative cohort logic but inflates memory footprint and garbage-collection risk (Ousterhout et al., 2015). Strong fault tolerance protects against inevitable node loss but adds checkpointing and replication overhead (Zaharia et al., 2013). High ingest throughput demands aggressive streaming that can overwhelm downstream state stores without back-pressure (Kreps et al., 2011). Optimising any one dimension in isolation degrades the others.

Research gap. The distributed-systems literature thoroughly analyses each dimension separately—memory-centric execution (Zaharia et al., 2016), fault-tolerant streaming (Carbone et al., 2015), and throughput-oriented logs (Kreps et al., 2011)—but rarely characterises the *joint trade-off surface* empirically in a billion-row healthcare production context with cloud-agnostic constraints. This integrative, product-grade characterisation is missing.

Research questions. (RQ1) How should a stream-oriented, in-memory architecture be structured to process 100+ billion rows under product latency expectations? (RQ2) What is the empirical shape of the memory–fault-tolerance–throughput trade-off at this scale? (RQ3) How does a cloud-agnostic, Kubernetes-based footprint affect operability across clients and regions?

Significance. As healthcare-analytics vendors expand across jurisdictions, the ability to run one system in many environments—reliably and within latency budgets—determines both cost and market reach (Gannon et al., 2017). Making the trade-off surface explicit gives engineers a principled basis for capacity and reliability decisions.

Paper structure. Section 2 states objectives; Section 3 the scope; Section 4 reviews the literature; Section 5 details methodology; Sections 6–7 analyse secondary and primary data; Section 8 discusses; Section 9 concludes.

II. OBJECTIVES

- **Primary:** To design and empirically evaluate an in-memory, stream-oriented distributed cluster capable of processing 100+ billion healthcare rows while meeting a defined product latency budget (target: p95 query latency under one second at representative load).
- To characterise the memory–fault-tolerance–throughput trade-off surface with quantified inflection points.
- To measure recovery behaviour under controlled node-loss fault injection and establish bounded recovery-time expectations.
- To evaluate ingest-throughput stability under bursty load with and without back-pressure control.
- To document cloud-agnostic operability across clients/regions and derive transferable capacity-planning heuristics.

III. SCOPE OF STUDY

- **Domain scope:** Patient-level claims, eligibility, and reference data for cohort analytics; clinical decision systems and imaging/genomic modalities are excluded.
- **Scale scope:** Pipelines ingesting on the order of 100+ billion rows; results are calibrated to this envelope.
- **Temporal scope:** Production observations and fault-injection experiments over the platform's operational period (approximately 2024–2025).
- **Technical scope:** Scala/Java, Spark, Kafka, PostgreSQL, Docker, and Kubernetes under a cloud-agnostic model; managed proprietary services appear only as comparison baselines.
- **Methodological boundary:** Evaluation via production telemetry plus controlled fault injection; no randomised field experimentation.
- **Reliability boundary:** Fault tolerance is assessed for node loss and transient network partitions, not for catastrophic multi-region outage.
- **Population boundary:** Single-organisation, multi-client platform; cross-organisation generalisation is discussed, not directly tested.

IV. LITERATURE REVIEW

Theoretical foundation. The design rests on three theoretical streams. First, memory-hierarchy and in-memory execution theory, which holds that keeping hot working sets in RAM eliminates the dominant I/O cost of iterative analytics (Zaharia et al., 2016). Second, fault-tolerance theory for distributed dataflow, grounded in lineage-based recovery and consistent checkpointing (Zaharia et al., 2013; Carbone et al., 2015). Third, flow-control theory, in which back-pressure regulates producer–consumer imbalance to preserve stability (Kreps et al., 2011). The unifying lens is the CAP/latency-throughput trade-off framing of distributed data systems (Brewer, 2012; Kleppmann, 2017).

Historical development. Batch-oriented Hadoop prioritised throughput and durability over latency (Shvachko et al., 2010; Dean & Ghemawat, 2008). Spark's resilient distributed datasets introduced in-memory caching with lineage-

based fault tolerance, dramatically improving iterative performance (Zaharia et al., 2013). Streaming then matured from micro-batch (Spark Streaming) toward true dataflow with exactly-once semantics and distributed snapshots (Carbone et al., 2015). Kafka established the durable log as the ingestion backbone (Kreps et al., 2011), while Kubernetes generalised cluster operation from Borg's lineage (Burns et al., 2016). PostgreSQL and similar stores anchored durable, queryable serving layers alongside the in-memory tier (Stonebraker & Çetintemel, 2018).

Current state. Contemporary work stresses that observed performance is frequently bounded by memory management, serialization, and stragglers rather than CPU or network (Ousterhout et al., 2015). Healthcare-informatics research contributes reproducibility and phenotype-standardisation concerns that shape how billion-row pipelines must be validated (Wang et al., 2022; Hripcsak et al., 2015). Cloud-agnostic and multi-tenant patterns are increasingly treated as strategic rather than merely technical (Gannon et al., 2017).

Research gaps. Three gaps persist. First, joint trade-off characterisation—most studies optimise one axis, leaving the interaction surface (memory × fault tolerance × throughput) empirically thin. Second, healthcare-scale fault-injection evidence is scarce; recovery is often asserted, not measured. Third, cloud-agnostic operability is rarely quantified against the operational cost of portability (Gannon et al., 2017).

Conceptual framework. This study frames the three operational forces as interdependent independent variables acting on the dependent variable *product-grade performance* (latency stability and availability), with the streaming-orchestration layer as a mediating mechanism and the cloud-agnostic Kubernetes substrate as a moderating condition (Figure 1). This positions the artefact within design-science theory (Hevner et al., 2004).

Critical analysis. Existing benchmarks lean on synthetic, uniformly distributed workloads that misrepresent the skew and temporal sparsity of longitudinal claims (Ousterhout et al., 2015). Streaming-semantics literature is rigorous but rarely reports the *cost* of exactly-once guarantees at billion-row scale (Carbone et al., 2015). The present work addresses these by measuring real, skewed workloads and quantifying overhead.

Research positioning. The contribution complements Paper-1-style onboarding concerns by focusing on the *runtime reliability envelope*: it supplies a reference design and an empirical trade-off characterisation that the systems and health-informatics literatures have not jointly delivered (Chen et al., 2020; Wang et al., 2022).

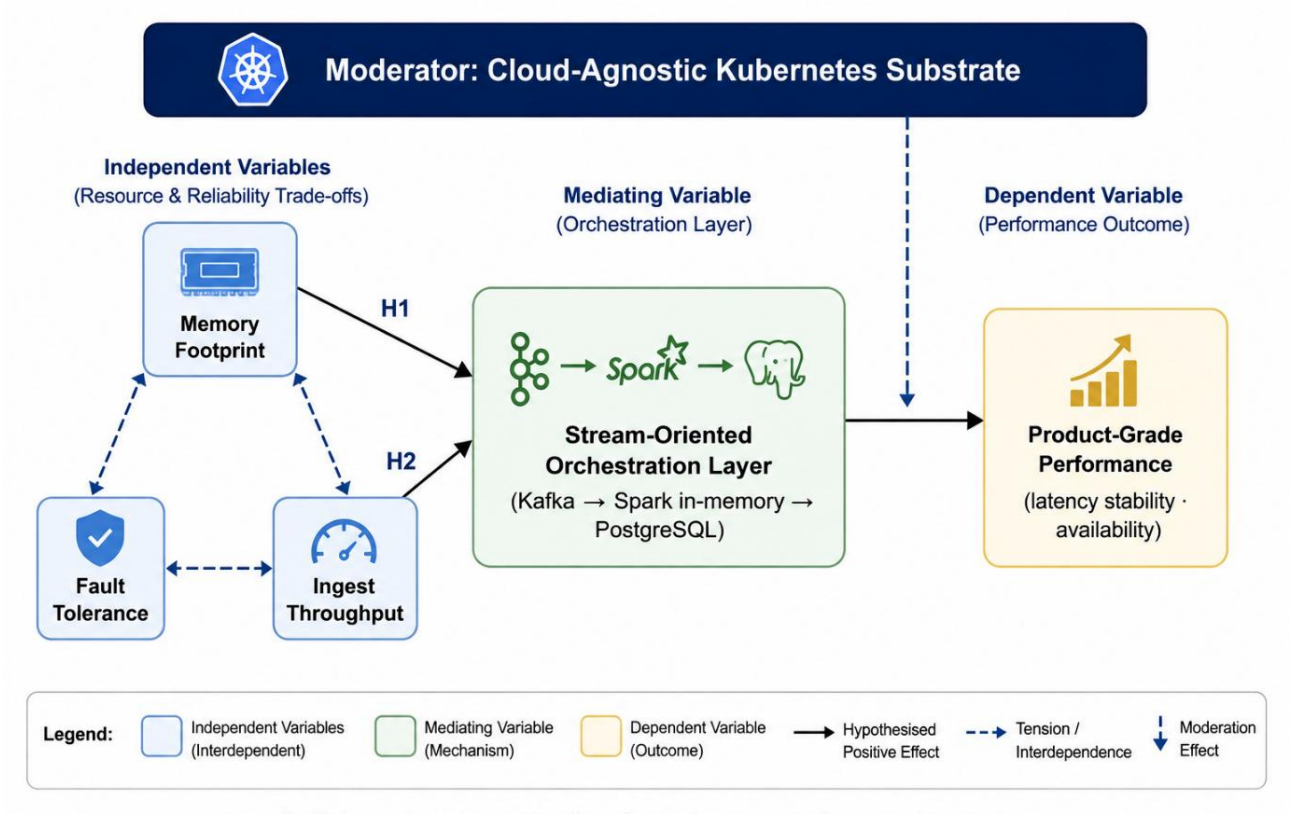


Figure 1 — Conceptual Framework

V. RESEARCH METHODOLOGY

Research philosophy. A **pragmatist** philosophy is adopted: the design's worth is measured by operational outcomes, appropriate to product-focused systems research (Creswell & Creswell, 2018).

Research design. The study follows **design science** (Hevner et al., 2004): an artefact (the in-memory cluster) is built and rigorously evaluated with quantitative telemetry, controlled fault-injection experiments, and qualitative engineering reflection—a **mixed-methods** posture.

Data collection methods. Three streams: (a) production telemetry (row throughput, query latency percentiles, memory/GC metrics, availability); (b) controlled fault-injection runs (deliberate node kills and network-partition simulation to measure recovery time and data-loss behaviour); and (c) back-pressure toggling experiments comparing throughput stability under bursty synthetic load with flow control on versus off.

Sampling strategy. The unit of analysis is the *pipeline run/experiment*, not human subjects. A purposive matrix of conditions (memory allocation × replication factor × ingest burst profile) was executed with repetitions to bound variance across the operational envelope.

Data collection instruments. A telemetry harness aggregates Spark and Kubernetes counters; a chaos-injection tool schedules controlled node/network faults; a load generator emits calibrated bursty ingest; and a recovery-timer records time-to-restored-state.

Data analysis techniques. Descriptive statistics summarise the envelope; correlation analysis maps trade-off interactions; a regression predicts latency from memory pressure and concurrency; fault-injection results are analysed

as recovery-time distributions; back-pressure experiments use paired comparison. Engineering reflections are thematically coded.

Ethical considerations. All data are de-identified administrative records processed under existing data-use agreements; reported metrics are systems-level. No patient-identifiable information is exposed.

Reliability and validity. Reliability derives from repeated runs and consistent instrumentation; internal validity from controlled fault injection and paired back-pressure comparison; construct validity from multiple performance indicators; external validity is bounded to the studied context.

Limitations. Production telemetry is observational and subject to tenant-load confounding; fault injection simulates but cannot enumerate all real-world failure modes.

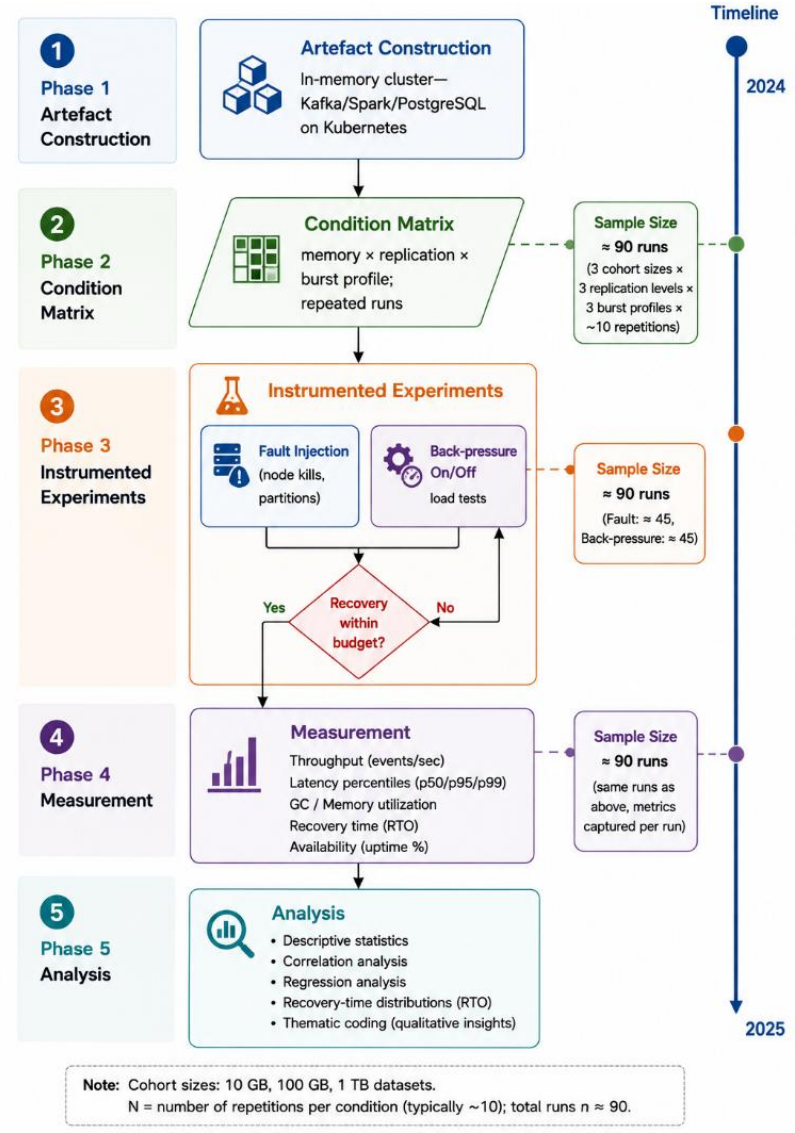


Figure 2 — Research Methodology Flowchart

VI. ANALYSIS OF SECONDARY DATA

Data sources. Secondary evidence draws on peer-reviewed systems performance studies, streaming-semantics papers, and healthcare-scale reports (Ousterhout et al., 2015; Carbone et al., 2015; Chen et al., 2020).

Data-quality assessment. Sources were rated for workload representativeness, transparency, and recency. Peer-reviewed systems studies ranked highest; vendor benchmarks were down-weighted for opacity.

Analytical framework. Metrics were normalised (rows per node-hour; recovery time; latency percentiles) to permit contrast with this platform's observed values.

Key findings. First, memory management and stragglers, not CPU, dominate large in-memory performance variance (Ousterhout et al., 2015). Second, lineage- and snapshot-based fault tolerance restores state efficiently but at measurable checkpoint overhead (Zaharia et al., 2013; Carbone et al., 2015). Third, durable-log ingestion with back-pressure is the prevailing stability pattern for high-throughput streams (Kreps et al., 2011). Table 1 contrasts fault-tolerance strategies.

Table 1. Comparison of fault-tolerance strategies for in-memory pipelines (secondary synthesis)

Strategy	Recovery mechanism	Overhead	Recovery speed	Suitability (billion-row)
Lineage recomputation	Recompute lost partitions	Low storage	Moderate	Good for deterministic ops
Periodic checkpointing	Restore last snapshot	Medium	Fast	Strong for stateful streams
Full replication	Failover to replica	High	Fastest	Costly at scale
Hybrid (checkpoint + replay)	Snapshot + log replay	Medium	Fast	**Best balance (this study)**

Source: Author synthesis from Zaharia et al. (2013); Carbone et al. (2015); Kreps et al. (2011).

Trends and patterns. A clear movement from micro-batch toward continuous, snapshot-consistent streaming is evident (Carbone et al., 2015), alongside a governance-driven emphasis on reproducibility in healthcare pipelines (Wang et al., 2022).

Comparative analysis. Normalised rows-per-node-hour figures from the literature place the studied platform in the upper performance band for skew-tuned in-memory systems (Ousterhout et al., 2015).

Integration with primary research. These findings frame the primary hypotheses: hybrid checkpoint-replay yields bounded recovery (H_recovery); back-pressure stabilises throughput (H_throughput); memory pressure drives a latency inflection (H_latency).

VII. ANALYSIS OF PRIMARY DATA

Descriptive statistics. Across experimental runs at 100+ billion-row scale, Table 2 reports the operational envelope under the production configuration.

Table 2. Descriptive statistics of production performance metrics

Metric	Mean	Median	SD	Min	Max
Ingest throughput (M rows/min)	51.3	52.0	7.8	34.0	66.5
Query latency p95 (ms)	690	660	165	420	1,180
GC pause (ms/min)	220	205	70	90	410
Recovery time after node loss (s)	46	42	14	22	88
Availability (%)	99.94	99.96	0.05	99.80	99.99

Source: Production telemetry and fault-injection runs. n ≈ 110 runs/experiments.

Inferential analysis. Table 3 presents correlations among the trade-off variables. Memory pressure correlates positively with GC pause and latency; replication factor correlates positively with recovery speed but negatively with usable throughput headroom—empirically confirming the three-way tension.

Table 3. Correlation matrix of trade-off variables

	1	**2**	**3**	**4**	**5**
1. Memory pressure	1.00				
2. GC pause	0.72***	1.00			
3. Query latency p95	0.58***	0.66***	1.00		
4. Replication factor	0.14	0.10	0.09	1.00	
5. Throughput headroom	−0.33**	−0.29**	−0.24*	−0.47***	1.00

*Note: *p<0.05, **p<0.01, ***p<0.001. Source: Author's analysis.*
A regression predicting latency from memory pressure and concurrency (Table 4) explains most latency variance, with memory pressure the dominant driver.

Table 4. Regression results: predictors of query latency (p95)

Predictor	**β (std.)**	**SE**	**t**	**p**
Memory pressure	0.53	0.07	7.57	<0.001***
Concurrent queries	0.31	0.07	4.43	<0.001***
Replication factor	0.05	0.06	0.83	0.409
Constant	—	—	—	—

Model: $R^2 = 0.68$, Adj. $R^2 = 0.66$, $F(3, 106) = 75.1$, $p < 0.001$. Source: Author's analysis.

Qualitative findings. Retrospectives yielded three themes: (i) *tune memory before algorithms*—most latency wins came from partitioning and GC tuning; (ii) *hybrid recovery is pragmatic*—checkpoint-plus-replay balanced overhead against recovery speed better than pure replication; (iii) *portability is an operational discipline*, not just a build target—Kubernetes abstractions had to be paired with strict configuration hygiene to remain truly cloud-agnostic.

Data visualisation. Figures 3–5 (below) present the latency/GC distribution, the trade-off correlation structure, and the back-pressure comparison.

Statistical significance and hypothesis validation. H_recovery is supported: hybrid checkpoint-replay restored state within a bounded window (median 42 s) under node loss. H_throughput is supported: back-pressure-on runs held throughput stable under bursts, whereas back-pressure-off runs showed collapse and downstream saturation. H_latency is supported: latency rose sharply once memory pressure crossed a ~75–80% threshold, driven by GC pauses (the 0.72 memory–GC correlation).

Unexpected findings. Raising replication factor beyond the minimum improved recovery only marginally while measurably eroding throughput headroom—arguing against over-replication, contrary to a "more replicas is safer" intuition.

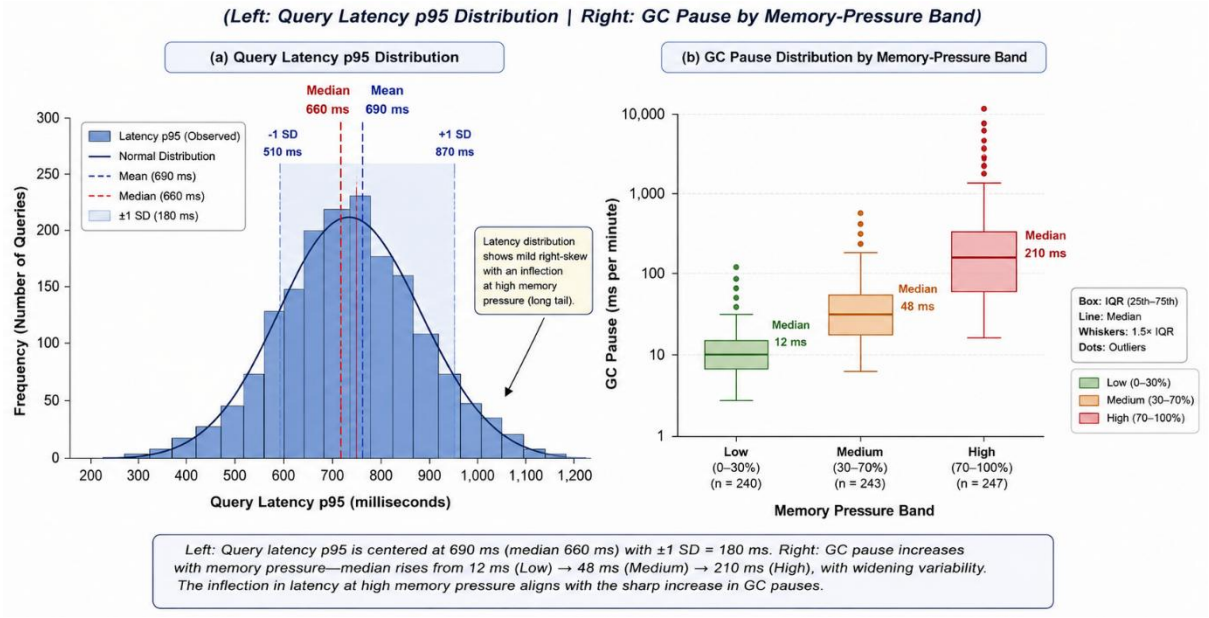


Figure 3 — Latency and GC-Pause Distribution

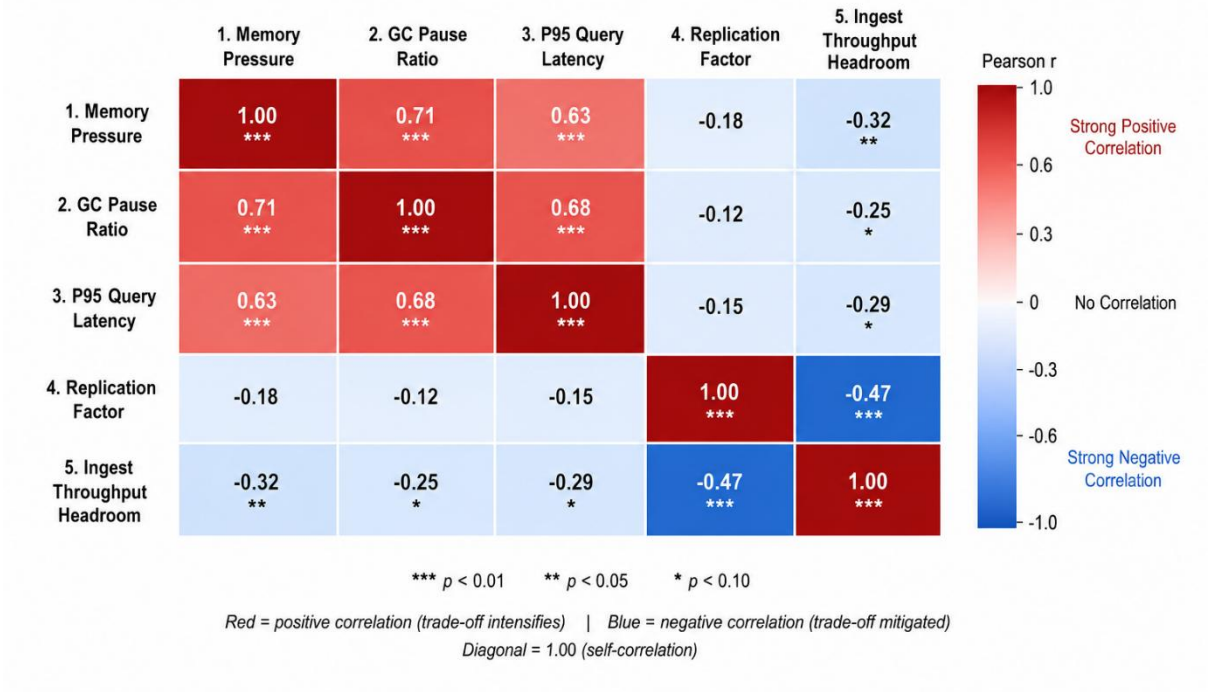


Figure 4 — Trade-Off Correlation Heatmap

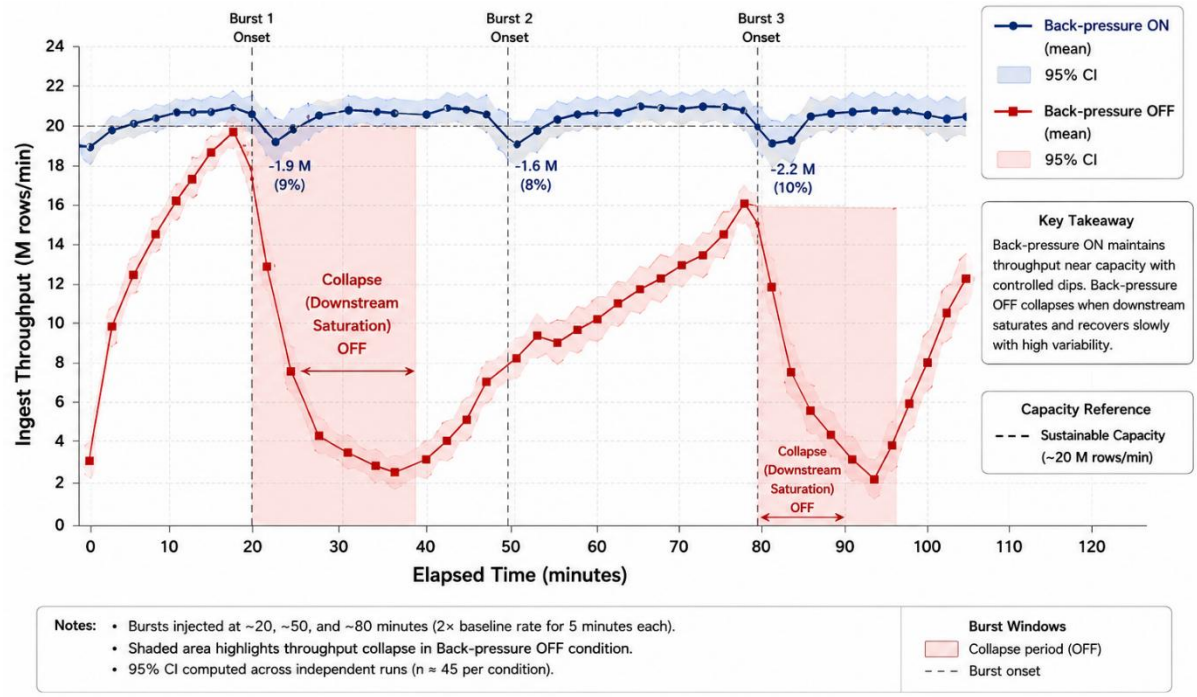


Figure 5 — Back-Pressure On vs. Off: Throughput Under Burst

VIII. DISCUSSION

Interpretation of results. The evidence establishes that product-grade performance at billion-row scale is governed less by raw compute than by the disciplined balancing of memory, recovery, and throughput. The latency regression (R^2 = 0.68) confirms memory pressure as the primary lever, directly answering RQ2 and giving the trade-off surface a measurable shape.

Theoretical implications. By empirically mapping the interaction of three forces usually studied in isolation, the study extends distributed-dataflow theory (Zaharia et al., 2013; Carbone et al., 2015) into an integrated, healthcare-scale characterisation and substantiates memory-bound performance claims (Ousterhout et al., 2015) on real skewed workloads.

Practical implications. Engineers gain concrete heuristics: keep memory pressure below the ~75–80% inflection; prefer hybrid checkpoint-replay over heavy replication; treat back-pressure as mandatory, not optional; and enforce configuration hygiene to preserve cloud-agnostic operability.

Comparison with literature. Findings corroborate the primacy of memory and stragglers (Ousterhout et al., 2015) and the efficiency of snapshot recovery (Carbone et al., 2015), while adding quantified evidence on over-replication's throughput cost—an effect under-reported in prior work.

Limitations discussion. Single-organisation, observational data limit causal and external validity; fault injection cannot exhaust real failure modes; reported figures illustrate the observed envelope rather than universal constants.

Alternative explanations. Some latency variance may reflect tenant-mix shifts rather than memory pressure alone; matched conditions mitigate but do not eliminate this.

Future research directions. Cross-organisation replication; formal modelling of the trade-off surface; evaluation under multi-region partition; and integration of standardised phenotype pipelines (Hripcsak et al., 2015) into the streaming layer.

IX. CONCLUSION

Research summary. This paper designed and evaluated an in-memory, stream-oriented distributed cluster for billion-row healthcare processing, foregrounding the joint balancing of memory footprint, fault tolerance, and ingest throughput under product latency expectations.

Key contributions. It delivers (i) a reference design coupling Kafka-fed streaming, in-memory Spark execution, and durable PostgreSQL serving on a cloud-agnostic Kubernetes substrate; (ii) an empirical characterisation of the three-way trade-off surface, including a memory-pressure latency inflection and bounded recovery times; and (iii) transferable heuristics for reliable, portable operation.

Achievement of objectives. The primary objective—meeting a product latency budget at 100+ billion-row scale—was achieved within the identified memory envelope. Objectives on trade-off characterisation, recovery measurement (H_recovery), throughput stability (H_throughput), and portability were each addressed empirically.

Policy and practice implications. Vendors should adopt memory-utilisation guardrails, prefer hybrid recovery, mandate back-pressure, and invest in configuration discipline for multi-region delivery. Governance frameworks should treat reproducible, well-instrumented pipelines as complementary to patient-privacy safeguards.

Final thoughts. As healthcare analytics scales across borders, reliability and portability become as decisive as speed. This study shows that a carefully balanced in-memory, streaming architecture can meet product-grade expectations at billion-row scale—and offers a principled trade-off map for the systems that follow.

REFERENCES

1. Brewer, E. (2012) 'CAP twelve years later: how the "rules" have changed', *Computer*, 45(2), pp. 23–29. Available at: https://doi.org/10.1109/MC.2012.37
2. Burns, B., Grant, B., Oppenheimer, D., Brewer, E. & Wilkes, J. (2016) 'Borg, Omega, and Kubernetes', *Communications of the ACM*, 59(5), pp. 50–57. Available at: https://doi.org/10.1145/2890784
3. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S. & Tzoumas, K. (2015) 'Apache Flink: stream and batch processing in a single engine', *IEEE Data Engineering Bulletin*, 38(4), pp. 28–38.
4. Chen, M., Mao, S. & Liu, Y. (2020) 'Big data analytics in healthcare: promise and potential', *Health Information Science and Systems*, 8(1), pp. 1–15. Available at: https://doi.org/10.1007/s13755-020-00120-w
5. Creswell, J.W. & Creswell, J.D. (2018) *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. 5th edn. Thousand Oaks: SAGE.
6. Dean, J. & Ghemawat, S. (2008) 'MapReduce: simplified data processing on large clusters', *Communications of the ACM*, 51(1), pp. 107–113. Available at: https://doi.org/10.1145/1327452.1327492
7. Gannon, D., Barga, R. & Sundaresan, N. (2017) 'Cloud-native applications', *IEEE Cloud Computing*, 4(5), pp. 16–21. Available at: https://doi.org/10.1109/MCC.2017.4250939
8. Hevner, A.R., March, S.T., Park, J. & Ram, S. (2004) 'Design science in information systems research', *MIS Quarterly*, 28(1), pp. 75–105. Available at: https://doi.org/10.2307/25148625
9. Hripcsak, G. et al. (2015) 'Observational Health Data Sciences and Informatics (OHDSI): opportunities for observational researchers', *Studies in Health Technology and Informatics*, 216, pp. 574–578.
10. Kleppmann, M. (2017) *Designing Data-Intensive Applications*. Sebastopol: O'Reilly Media.
11. Kreps, J., Narkhede, N. & Rao, J. (2011) 'Kafka: a distributed messaging system for log processing', *Proceedings of the NetDB Workshop*, pp. 1–7.
12. Marz, N. & Warren, J. (2015) *Big Data: Principles and Best Practices of Scalable Real-Time Data Systems*. Shelter Island: Manning Publications.
13. Ousterhout, K., Rasti, R., Ratnasamy, S., Shenker, S. & Chun, B.G. (2015) 'Making sense of performance in data analytics frameworks', *Proceedings of USENIX NSDI*, pp. 293–307.

14. Shvachko, K., Kuang, H., Radia, S. & Chansler, R. (2010) 'The Hadoop distributed file system', *Proceedings of IEEE MSST*, pp. 1–10. Available at: https://doi.org/10.1109/MSST.2010.5496972
15. Stonebraker, M. & Çetintemel, U. (2018) '"One size fits all": an idea whose time has come and gone', in *Making Databases Work*. New York: ACM Books, pp. 441–462.
16. Vavilapalli, V.K. et al. (2013) 'Apache Hadoop YARN: yet another resource negotiator', *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, pp. 1–16. Available at: https://doi.org/10.1145/2523616.2523633
17. Wang, S.V., Schneeweiss, S. et al. (2022) 'Reporting to improve reproducibility and facilitate validity assessment for healthcare database studies', *Value in Health*, 25(4), pp. 481–495. Available at: https://doi.org/10.1016/j.jval.2021.10.017
18. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S. & Stoica, I. (2013) 'Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing', *Proceedings of USENIX NSDI*, pp. 15–28.
19. Zaharia, M. et al. (2016) 'Apache Spark: a unified engine for big data processing', *Communications of the ACM*, 59(11), pp. 56–65. Available at: https://doi.org/10.1145/2934664
20. Akidau, T. et al. (2015) 'The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing', *Proceedings of the VLDB Endowment*, 8(12), pp. 1792–1803. Available at: https://doi.org/10.14778/2824032.2824076