Global Academic Journal of Economics and Business

Available online at https://www.gajrc.com
DOI: https://doi.org/10.36348/gajeb.2026.v08i04.046

OPEN ACCESS

ISSN: 2706-9001 (P)
ISSN: 2707-2584 (O)

Original Research Article

Real-Time Big Data Streaming and Event-Driven Architectures for Next-Generation Enterprise Systems

Adnan Anwar Shaikh[1]*
[1]Data Platform Senior Specialist (Information Technology), Savitribai Phule Pune University: Pune, Maharashtra, India

***Corresponding Author**
Adnan Anwar Shaikh
Data Platform Senior Specialist (Information Technology), Savitribai Phule Pune University: Pune, Maharashtra, India

Article History
Received: 27.06.2026
Accepted: 21.08.2026
Published: 24.08.2026

Abstract: Real-time enterprise computing is transitioning from periodic data movement and tightly coupled request-response integration onto continuous event streams, independently scalable services, and stateful processing that adapts as business conditions change. This review integrates research published between 2020 and 2025 on big data streaming, event-driven architectures, microservices, transactional stream processing, observability, and serverless execution. Its goal is to explain how these domains converge into a integrated architecture for next gen enterprise systems, rather than treating streaming as an isolated analytics component. A structured integrative review was carried out using DOI-verifiable, peer-reviewed sources from major computing publishers and journals. Thirty studies were retained based on their focus on architectural design, streaming semantics, scalability, reliability, consistency, deployment, or operational governance. Key factors such as event-time semantics, watermarks, durable logs, checkpointed state, idempotent consumption, schema evolution, distributed transaction models, and end-to-end observability collectively determine a system's ability to deliver timely and trustworthy outcomes. Comparative evidence demonstrates that no stream-processing framework is universally optimal; workload characteristics, state requirements, deployment topology, and cost constraints considerably influence engineering decisions. While event-driven microservices enhance autonomy and extensibility, they also bring complexity in information consistency, asynchronous failure recovery, tracing, and governance. Serverless and edge patterns can reduce operational cost and network latency, but cold starts and distributed state management remain major challenges. This review proposes a layered enterprise event fabric and a correctness-resilience loop that integrate transport, stream computation, domain choreography, data governance, and operational assurance. The resulting research agenda highlights adaptive state management, portable semantics, energy-aware scheduling, automated data contracts, and verifiable real-time service-level objectives.

Keywords: Real-Time Big Data, Event-Driven Architecture, Stream Processing, Microservices, Enterprise Systems, Distributed Data, Serverless Computing, Observability.

1. INTRODUCTION

Enterprises operate in situations where the business value of data declines swiftly after an event occurs. Functions such as payment authorization, fraud detection, inventory allocation, industrial monitoring, customer personalization, cybersecurity response, and digital operations require decisions to be made before data can be consolidated into traditional warehouse cycles. This change has elevated live data streaming from specialized analytics feature to a core architectural concern.

Wadhwa (2023) identifies a convergence between event-driven microservices and distributed data architectures for high-performance business applications, while recent research positions microservice-integrated streaming as a distinct cloud-native deployment pattern (Henning and Hasselbring, 2024).

This design transformation is significant because it replaces temporal and organizational coupling with continuous asynchronous interaction. In synchronous designs, a service typically waits for a response from another service, making the availability and latency of downstream dependencies part of the caller's critical path. In contrast, event-driven communication records state changes as events, enabling one or more consumers to process these events independently. Systematic studies on microservice deployment and communication indicate that messaging, publish-subscribe interaction, and asynchronous mechanisms have become central design choices, though they cause challenges related to deployment, message coordination, and operational complexity (Karabey Aksakalli *et al*., 2021). Research on quality attributes further demonstrates that scalability, performance, reliability, maintainability, security, and testability must be addressed collectively rather than optimized in isolation (Li *et al*., 2021).

Big data streaming brings in extra complexity, as computation occurs on unbounded and potentially disordered data while the system operates continuously. Modern stream processors manage state, time, failures, scaling, and reconfiguration, allowing applications to advance beyond stateless message handling to support continuous joins, aggregations, pattern detection, and machine learning capabilities (Fragkoulis *et al*., 2024). Watermarks offer a systematic and practical mechanism to reason about progress in event time despite out-of-order arrivals, permitting applications to determine when results are sufficiently complete and when state can be safely retired (Akidau *et al*., 2021). Transactional stream processing further expands these challenges through integrating streaming guarantees with transaction-like correctness across state updates and outputs (Zhang *et al*., 2024).

Microservices delineate business capabilities, stream processors manage continuously evolving state, and event brokers decouple producers from consumers. Failures can occur at any boundary. Records may be delivered multiple times, arrive late, be processed after a schema change, or initiate distributed business processes that only partially complete. Event sourcing enhances auditability but introduces ongoing schema-evolution obligations as long as historical events remain replayable (Overeem *et al*., 2021). Research on distributed transactions demonstrates that compensation, asynchronous protocols, and bounded consistency are often more scalable than globally coordinated commits (Xue *et al.,* 2021; González-Aparicio *et al*., 2023). These problems render architectural assurance inseparable from performance engineering.

This review fills this gap by analyzing streaming and event-driven enterprise systems as a unified socio-technical architecture. It examines the interactions among data transport, computation, domain services, state, consistency, resilience, observability, and governance. The analysis draws on recent evidence from benchmark studies, industrial investigations, system implementations, and focused surveys published between 2020 and 2025. Instead of ranking technologies, the synthesis identifies enduring design principles and trade-offs that remain relevant as individual platforms evolve.

2. Aim and Objectives

This review intends to develop an integrated architectural understanding of real-time big data streaming and event-driven systems for next-gen enterprises, with particular attention to the conditions required for extensible, correct, observable, and evolvable operation. Four objectives guide the synthesis. First, the review explains how event backbones, stream processors, microservices, and distributed data stores form an end-to-end enterprise event fabric. Second, it evaluates the technical trade-offs that shape real-time correctness, including event time, state, delivery semantics, transactional boundaries, and schema evolution. Third, it examines scalability and operational assurance through benchmarking, elasticity, tracing, resilience, and governance evidence. Fourth, it identifies emerging directions within serverless, edge, complex-event, energy-aware, and low-code streaming that are likely to influence enterprise architecture through the next generation of systems.

3. REVIEW METHODOLOGY

A structured integrative review method was used because the research question spans software architecture, distributed systems, stream processing, cloud computing, and enterprise data engineering. The review period was restricted to 2020-2025 so that the evidence reflected current cloud-native streaming, microservice, and serverless practices.

Search concepts were organised into four groups: live data streaming; event-driven or asynchronous architecture; microservices and distributed data; and operational qualities such as scalability, consistency, resilience, observability, or serverless execution. Combinations of these concepts

were used to identify peer-reviewed material in publisher databases and DOI-indexed scholarly records, including Elsevier/ScienceDirect, SpringerLink, IEEE publications, VLDB venues, and associated institutional repositories used to verify bibliographic metadata. The attached 2023 IJCET paper was used as a structural reference for an architecture-centred discussion that combines concepts, figures, tables, challenges, and future directions. Still, the present review applies a newer evidence window and a more explicit synthesis methodology.

Studies were included when they met four criteria: publication between 2020 and 2025; a direct contribution to streaming, event-driven architecture, microservices, real-time computation, or related operational systems; sufficient technical detail to support architectural interpretation; and a stable DOI or publisher record. Sources focused only on generic big data, conventional batch analytics, or pre-cloud service-oriented architecture were excluded unless their contribution was directly implemented in a modern streaming context. The final corpus contains thirty references. This fixed corpus was not selected to maximise citation count; it was selected to achieve thematic coverage across processing semantics, performance, consistency, system evolution, and emerging deployment models.

The synthesis followed a concept-matrix approach. Evidence was coded into six themes: event transport and communication; stream execution and time semantics; distributed state and consistency; scalability and performance; observability and evolvability; and emerging serverless, edge, or complex-event patterns. Benchmark studies were interpreted as comparative evidence rather than universal rankings because their results depend on workload and infrastructure. Industrial or empirical microservice studies were used primarily to detect recurring operational and evolution problems. Review papers were used to triangulate terminology and research gaps. This procedure supports a critical narrative synthesis rather than a simple catalogue of technologies.

4. Architectural Foundations of the Enterprise Event Fabric

The central architectural insight from the literature is that real-time enterprise systems are best understood as an event fabric with separable planes rather than as one streaming product. At the ingress boundary, events originate from applications, transaction systems, sensors, change-data-capture feeds, logs, external services, and user activities. A durable event backbone then provides decoupled transport, ordered partitions, retention, replay, and fan-out. Stream processors consume these records to perform filtering, enrichment, windowing, joins, pattern detection, and stateful computation. Domain services translate processed events into business actions, while data stores, analytical systems, and machine-learning components consume both raw and derived streams. Cross-cutting reliability, governance, and observability mechanisms span every layer.

Henning and Hasselbring (2021) formalised stream-processing use cases inside microservices and demonstrated that scalability should be evaluated against specific workload dimensions rather than a single throughput number. Their later comparison showed approximately linear scaling for several modern frameworks when sufficient resources were available, but also substantial differences in the rate at which additional resources were required (Henning and Hasselbring, 2024). Van Dongen and Van den Poel (2020) similarly found that framework behaviour changes across workloads, configurations, latency targets, and burst conditions. Bordin *et al*., (2020) supported the importance of workload diversity by constructing a benchmark suite spanning finance, telecommunications, sensor networks, and social data.

Its retained log becomes a coordination substrate through which consumers can advance independently, restart from known positions, or replay historical data. This characteristic is especially important when a stream processor maintains materialised state or when a new service needs to rebuild a derived view. Event-sourced systems studied by Overeem *et al*., (2021) achieved reliability, flexibility, and auditability, yet practitioners reported schema evolution as a persistent engineering challenge. Consequently, schema governance belongs in the architecture from the beginning instead than being added after producers and consumers proliferate.

Microservices fit naturally into this fabric because a service can own a business capability and subscribe only to events relevant to that bounded context. Loose coupling enables independent scaling and deployment, but coupling does not disappear; it shifts from direct API dependencies to semantic dependencies on event meaning, ordering, and timing. Apolinário and de França (2021) show that microservice coupling evolves over time and therefore needs active monitoring. Bogner *et al*., (2021) found that practitioners rely on guidelines, standardisation, and event-driven messaging to support evolvability, but still need better techniques for continuously evaluating dependencies and service granularity. Architectural governance must therefore measure both runtime flow and design-time relationships.

Figure 1 summarises the resulting enterprise event fabric. The important design principle is separation of concerns: event transport should not dictate business choreography; stream compute should not become the sole system of record; domain services should not hide cross-service consistency assumptions; and reliability, governance, and observability should be explicit planes. This separation supports technological substitution while retaining semantic contracts. It also gives a clearer basis for service-level objectives because latency, lag, recovery time, schema compatibility, and business completion might be measured at the layer where each property is controlled.

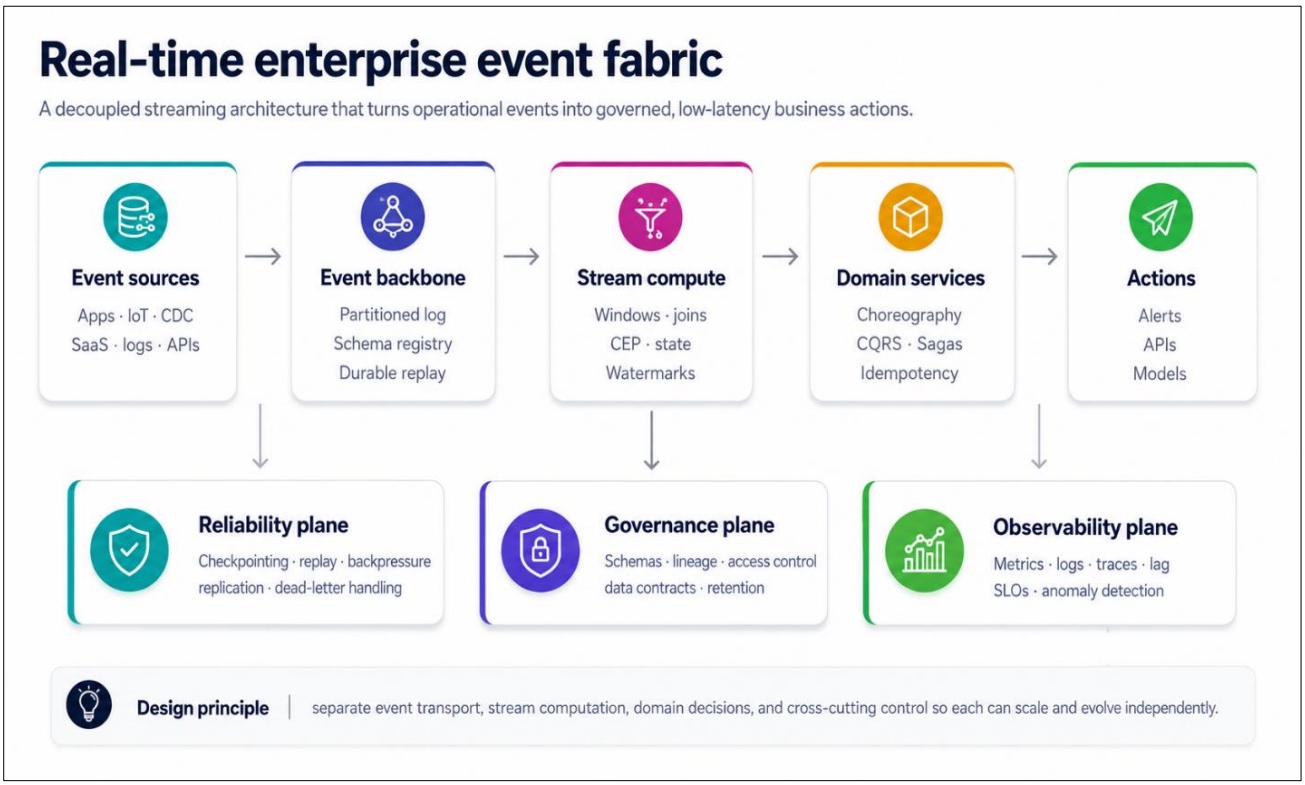

Figure 1: Layered enterprise event fabric for governed instantaneous processing

5. Stream Processing Technologies and Execution Semantics

Stream-processing engines transform a sequence of events into continuously updated results, but the engineering difficulty lies in preserving meaning under disorder, failure, and scale. Modern systems distinguish event time from processing time because networks, retries, buffering, or upstream outages may delay records. Akidau *et al*., (2021) show that watermarks provide a practical signal of temporal completeness, allowing a processor to decide when to emit results, handle late arrivals, or release obsolete state. Without an explicit time model, a low-latency pipeline may still produce incorrect windows or inconsistent alerts because arrival order is mistaken for business order.

State management is equally fundamental. Aggregations, joins, pattern detection, deduplication, and sessionization require state that may be too large for local memory and must survive process or node failure. Fragkoulis *et al*.. (2024) identify state management, fault tolerance, high availability, load management, elasticity, and reconfiguration as defining capabilities in the evolution of modern stream processors. Checkpointing creates recoverable snapshots of operator state, while replayable input allows the system to reconstruct progress after failure. These mechanisms must coordinate with output semantics; otherwise, recovery can re-emit results or lose updates.

At-most-once may lose data but avoids duplicates. At-least-once ensures retry but requires idempotent consumers or deduplication. Exactly-once processing generally means that the combined effects of state changes and outputs appear once under a specified model, not that packets physically traverse the network once. Zhang *et al*., (2024) show that transactional stream processing remains an active research area because transactional isolation, stream order, state updates, and external side effects create difficult design trade-offs. For enterprise architects, the practical lesson is to define the unit of correctness before selecting a guarantee.

Van Dongen and Van den Poel (2020) evaluated Spark Streaming, Structured Streaming, Flink, and Kafka Streams under multiple workloads and demonstrated that performance differs with optimisation goals and burst patterns. Henning and Hasselbring (2024) compared Flink, Kafka Streams, Hazelcast Jet, and Beam-based configurations deployed as microservices and found no universal winner. Flink, Kafka Streams, and Hazelcast could be efficient for different use cases, while abstraction through Beam introduced additional resource demand in their experiments. The result supports a workload-first selection strategy rather than choosing a platform by headline throughput.

Toliopoulos *et al*., (2020) implemented continuous outlier mining on Flink and demonstrated considerable speedups from parallel techniques, showing that algorithm structure and partitioning can dominate platform choice. Kontaxakis *et al*., (2023) used Flink to provide synopsis-as-a-service for extreme-scale analytics, combining parallel processing with reusable summaries to support horizontal, vertical, and federated scalability. These examples show stream processors evolving from compute engines into foundations for sector-specific real-time services.

Table 1 compares architectural roles rather than marketing features. A broker-centred library such as Kafka Streams can decrease operational layers when processing is tightly coupled to Kafka topics. In contrast, a distributed engine such as Flink is attractive for sophisticated stateful event-time computation. Micro-batch or unified engines can be useful when batch and streaming share pipelines, but latency and state semantics must be evaluated against requirements. Serverless event handlers offer elastic simplicity for short-lived event reactions, although persistent state and predictable low latency can be harder to guarantee. The correct design can also combine these options: a durable event log for transport, a stateful engine for continuous computation, and serverless functions for sporadic downstream actions.

Complex event processing adds meaning analysis over raw streams. Ortiz *et al*., (2022) demonstrated reusable microservices for instant IoT processing in a smart-port context, integrating Web of Things concepts with complex event processing. Khriji *et al*., (2022) implemented a cloud-based event-driven architecture using MQTT, Kafka, and microservices for instant wireless sensor data. Rosa-Bilbao *et al*., (2023) extended event-driven processing through integrating complex-event detection with blockchain through a low-code architecture. Together, these studies show that enterprise streaming is moving beyond movement of records toward declarative recognition of situations that require action.

Table 1: Architectural roles of representative instant processing options

Architectural option	Execution model	State / time capability	Primary enterprise fit
Broker-embedded stream library	Application-embedded continuous processing near the event log	Strong for partitioned state; event-time support depends on library semantics	Kafka-centric services, localized transformations, compact operational footprint
Distributed stateful stream engine	Clustered dataflow with operators, checkpoints, repartitioning, and recovery	Rich keyed state, event-time windows, watermarks, and recovery coordination	Complex joins, CEP, continuous analytics, large state, strict freshness targets
Unified batch-stream engine	Shared programming model for batch and continuously arriving data	Useful when pipelines share transformations; semantics vary by execution mode	Organizations consolidating batch and streaming engineering patterns
Serverless event handlers	Event-triggered functions scaled by the platform	Best for short-lived or externally persisted state; cold starts affect predictability	Bursty reactions, enrichment, notifications, sporadic downstream actions

Note: The comparison emphasises architectural role and design trade-offs rather than vendor ranking; benchmark outcomes remain workload- and infrastructure-dependent.

6. Event-Driven Integration, Consistency, and System Evolution

Event-driven architecture improves autonomy because a producer does not need to know which consumers will react to an event. This self-governance is valuable in large enterprises where teams release services at different speeds, but it also changes how correctness is managed. In an event-driven microservice system, one business outcome may span several independently stored states and may complete over seconds or minutes. The architecture therefore needs explicit patterns for

partial failure, retries, duplication, compensation, and convergence.

Xue *et al*., (2021) studied decentralised coordination of distributed microservices and used Saga-based mechanisms to help compositions reach consensus without a central coordinator. González-Aparicio *et al*., (2023) designed asynchronous transaction schemes for microservices operating over clustered NoSQL databases and evaluated consistency and performance under failure-free and failure-prone conditions. These studies illustrate a wider principle: global atomicity is often replaced using bounded local transactions plus controlled coordination. This increases scalability and availability, but business processes must define compensating actions and acceptable intermediate states.

CQRS and event sourcing are commonly associated with this model because they separate write decisions from read representations and preserve domain modifications as events. Their benefit is strongest when auditability, temporal reconstruction, or multiple derived views are important. Overeem *et al*., (2021) found several proven methods for evolving event schemas, but also showed that schema evolution is not a marginal concern in event-sourced systems. Every historic event retained for replay can become a compatibility obligation. Enterprise governance should therefore establish versioning rules, deprecation windows, consumer-driven compatibility tests, and ownership of canonical event meaning.

A stream processor may update its local state exactly once while an external service receives a duplicate side effect. Conversely, a service may complete a database transaction but fail before publishing the corresponding event. Reliable patterns deal with these gaps with idempotency keys, transactional outboxes, durable retry, deduplication, and explicit reconciliation. Transactional stream-processing research suggests that future systems will progressively integrate state and output guarantees. Still, current enterprise designs should make cross-boundary assumptions visible rather than relying on framework labels (Zhang *et al*., 2024).

Ntentos *et al*., (2021) proposed detector-based abstraction to understand microservice systems through observed structures, pointing out the need to recover architectural views from distributed deployments. Bogner *et al*., (2021) found that industry struggles with service dependencies and constant development despite the modularity promised by microservices. Apolinário and de França (2021) similarly treated coupling as a property that changes over time. Event-driven systems therefore require living architecture maps: topic ownership, producer-consumer relationships, schema versions, service dependencies, and data lineage should be discoverable rather than maintained only in static diagrams.

A retained log can increase exposure because historic records remain accessible for replay. Security controls must cover producer identity, topic authorisation, encryption, retention, consumer privileges, and downstream propagation. Governance should also differentiate operational events from analytical copies so that retention and access can be aligned with business purpose. Although security mechanisms vary by platform, the architectural requirement is stable: event mobility must not imply uncontrolled data mobility.

Figure 2 presents correctness as a continuous loop rather than a single configuration option. Event time determines when information is considered complete; checkpointed state determines what can be recovered; delivery semantics determine duplicate and loss behaviour; distributed consistency patterns determine how business processes unify; schema evolution determines whether replay remains safe; and observability determines whether these properties can be verified in production. A weakness in any one element can invalidate the perceived real-time accuracy of the whole system.

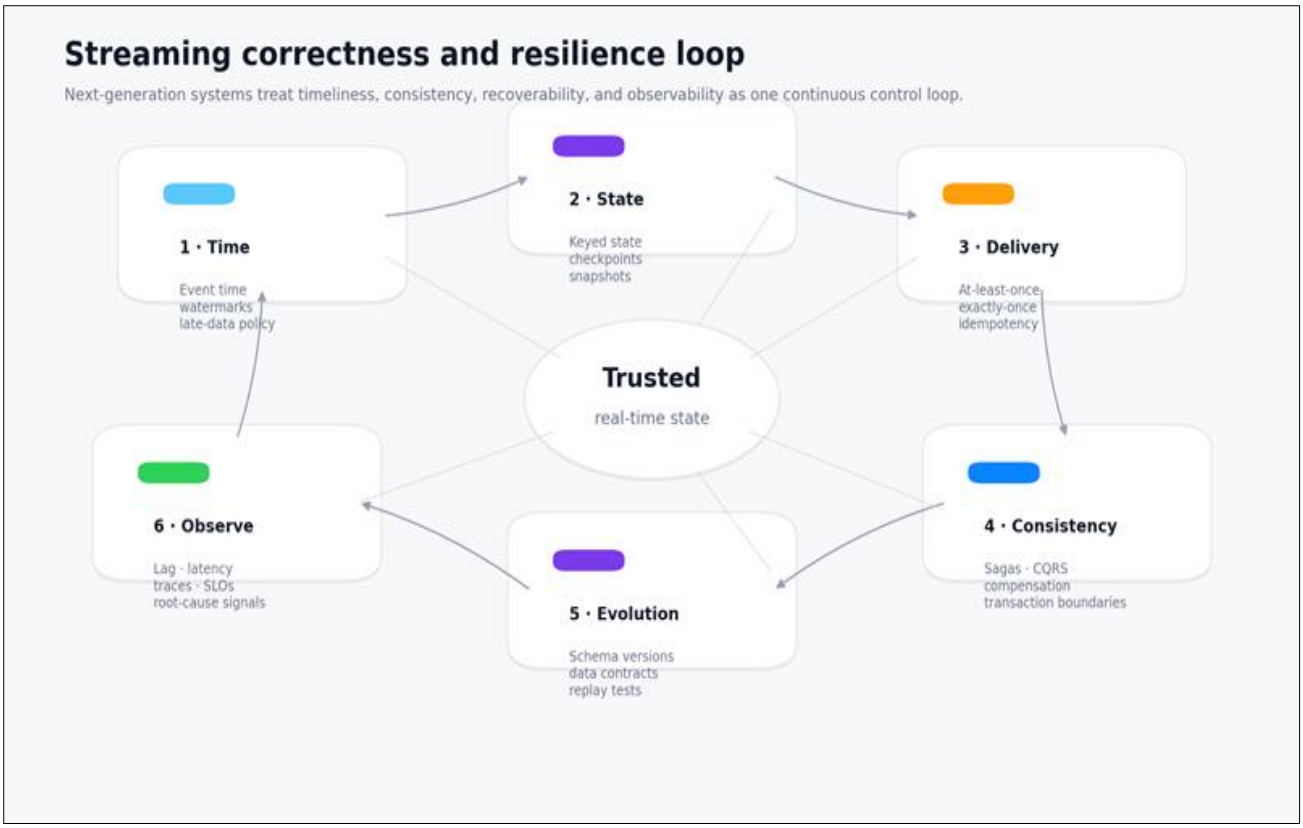

Figure 2: Correctness-resilience loop for trustworthy real-time enterprise state.
Source: Author synthesis of reviewed evidence on event time, state, consistency, evolution, and observability.

7. Scalability, Resilience, Observability, and Enterprise Operations

Scalability within a real-time enterprise system has at least three meanings: the ability to absorb higher event rates, the ability to increase computational work per event, and the ability to expand the number of independently evolving services and streams. Benchmarking that reports only maximum messages per second captures the first dimension incompletely and ignores the others. Theodolite tackles this by relating scalability regarding use cases and workload dimensions, supplying a method for determining how resource demand grows as workload changes (Henning and Hasselbring, 2021). Its later multi-framework evaluation confirms that resource efficiency can differ substantially even when systems scale approximately linearly (Henning and Hasselbring, 2024).

In streaming architectures, resilience combines broker replication, consumer retry, processor checkpoints, state restoration, load redistribution, and business-level compensation. Camilli and Russo (2022) modelled performance violations in microservice systems and showed how time-series patterns can help distinguish resilient from problematic behaviour. This perspective is important because a system may be functionally available while repeatedly violating response-time expectations. Resilience objectives should therefore include latency and lag SLOs, not only uptime.

When consumers cannot process events as quickly as producers create them, queues grow, event age increases, and "real time" silently becomes delayed time. The architecture must decide whether to scale processing, shed optional work, reduce enrichment, prioritise critical event classes, or slow producers. Since stateful operators may require repartitioning during scaling, elasticity can itself introduce latency. Capacity management should therefore observe input rate, consumer lag, operator utilisation, state size, checkpoint duration, and recovery time together.

Observability provides the evidence needed to operate this complexity. Distributed traces connect causally related work across services, metrics reveal resource and queue behaviour, and logs preserve detailed diagnostic events. Mayr-Dorn *et al*., (2021) demonstrated distributed tracing for industrial OPC UA method calls, showing how trace correlation can expose latency and endpoint problems in distributed production systems. Kohyarnejadfard *et al*., (2022) used distributed tracing data with NLP-based analysis to detect performance anomalies and accelerate root-cause investigation in microservice environments.

Observability in asynchronous systems must go beyond request traces. A single business event may fan out to multiple consumers with no shared synchronous request. Correlation identifiers, event identifiers, causation links, schema versions, and producer timestamps are therefore valuable parts of event metadata. Enterprise dashboards should distinguish transport latency, processing latency, business completion latency, and freshness. A system can show low CPU utilisation while consumers are hours behind if lag is not monitored. Conversely, high throughput can be acceptable when event freshness remains within the business SLO.

Li *et al*., (2025) proposed energy-aware scheduling and coordinated load balancing for Flink inside heterogeneous clusters, showing that scheduling policy can reduce energy use while observing performance requirements. This direction is relevant to enterprises because always-on stream processors maintain resources even during low demand. Cost optimisation should not simply minimise provisioned capacity; it should consider checkpoint overhead, state migration, broker replication, cross-zone traffic, and the financial impact of delayed decisions. Energy-aware scheduling adds sustainability as another measurable system quality alongside throughput and latency.

Ordering and late data require event-time policies and watermarks. Duplicate delivery requires idempotency or deduplication. Distributed workflows require Sagas, compensation, and reconciliation. Schema evolution entails contracts and replay testing. Backpressure requires lag-aware scaling and priority policies. Operational opacity requires metrics, logs, traces, and business-level correlation. The residual risk cannot be eliminated, but it can be made explicit and monitored. This is the difference between a streaming platform that is technically fast and an enterprise system that is operationally trustworthy.

Table 2: Recurring design risks, controls, and operational evidence in event-driven systems

Design risk	Typical failure mode	Primary architectural control	Operational evidence
Late or out-of-order events	Premature windows; inaccurate aggregates; delayed decisions	Event-time policies; watermarks; late-data handling	Watermark delay; event age; late-event rate
Duplicate delivery or replay	Repeated side effects; double counting; inconsistent state	Idempotency; deduplication; transactional output boundaries	Retry rate; duplicate detections; reconciliation exceptions
Cross-service inconsistency	Partial business completion after service failure	Sagas; compensation; bounded local transactions	Saga completion time; compensation rate; unresolved transactions
Schema evolution	Consumer breakage or unsafe historic replay	Versioned contracts; compatibility checks; replay tests	Compatibility failures; deprecated versions; replay-test results
Backpressure and skew	Growing lag; stale results; overloaded partitions	Lag-aware scaling; repartitioning; priority classes	Consumer lag; state size; checkpoint duration; latency
Operational opacity	Slow diagnosis; uncertain business completion	Correlated metrics, logs, traces; event metadata; business SLOs	Trace coverage; freshness; completion latency; alert quality

Note: Controls should be evaluated against business-level freshness, correctness, recovery, and completion objectives, not infrastructure metrics alone.

8. Emerging Patterns: Serverless, Edge, and Event Intelligence

Serverless computing extends event-driven design by allowing functions to be instantiated in response to events without fixedly allocated application servers. For bursty or irregular workloads, this can decrease operational effort and align cost with execution. Poojara *et al*., (2022) evaluated serverless data-pipeline approaches across fog and cloud environments and showed that different mechanisms fit different workload characteristics; object storage, data-flow tools, and MQTT-based designs showed distinct processing and resource trade-offs.

Moreno-Vozmediano *et al*., (2023) analysed latency and resource consumption within serverless edge analytics and highlighted cold starts as an important issue for strict latency requirements. Moina-Rivera *et al*., (2023) demonstrated event-driven serverless pipelines for video processing, illustrating that massively parallel event-triggered work can be effective when tasks can be decomposed into independent units. Bharti *et al*., (2023) further explored choreography for fork-join serverless

workflows, showing that complex event-triggered composition can be coordinated without relying only on centralised orchestration.

Edge processing is particularly relevant for industrial, retail, transport, and IoT systems where network latency, bandwidth, privacy, or intermittent connectivity limits cloud-only streaming. A sensible next-generation architecture may therefore use hierarchical event fabrics: local brokers and processors handle immediate filtering or control, regional systems aggregate operational context, and cloud platforms perform cross-enterprise analytics and long-term learning. The challenge is upholding consistent schemas, security policies, and observability across these tiers.

Complex-event processing and low-code tooling may further broaden who can define real-time behaviour. Rosa-Bilbao *et al*., (2023) showed that low-code models can integrate IoT events, complex-event analysis, and blockchain recording. Ortiz *et al*., (2022) demonstrated reusable microservices for standardised smart-port processing. These approaches suggest a situation where domain specialists define event patterns and actions at a higher level while platform teams govern execution, schemas, and reliability. The architectural risk is uncontrolled proliferation of event logic; governance must therefore make low-code deployment subject to the same testing, lineage, and SLO controls as hand-coded services.

Finally, stream intelligence is converging with machine learning. Real-time features, anomaly scores, recommendations, and predictive maintenance outputs can be generated continuously, but model inference adds computational variability and new state. The event fabric should separate model lifecycle from event transport so models can be replaced without rewriting upstream producers. Future research needs to examine adaptive placement of inference across edge and cloud, state-efficient online learning, and assurance methods that connect model quality with streaming freshness and system dependability.

9. DISCUSSION AND RESEARCH AGENDA

First, real-time architecture is a system property, not a product category. Durable transport, event-time computation, distributed state, business consistency, and observability must operate together. Benchmarks from Bordin *et al.,* (2020), van Dongen and Van den Poel (2020), and Henning and Hasselbring (2024) show why framework selection should begin with representative workloads, state size, burst behaviour, and cost constraints. Third, asynchronous autonomy creates semantic responsibility. Teams gain deployment independence but must manage data contracts, replay compatibility, idempotency, and compensation.

Fourth, the operational control plane is becoming as important as the data plane. Tracing research, coupling monitoring, and empirical evolvability studies show that architecture must be continuously observed as services and event relationships change (Apolinário and de França, 2021; Kohyarnejadfard *et al.,* 2022). Fifth, emerging serverless and edge models do not replace stateful stream processors; they expand the design space. Enterprises will increasingly combine persistent streaming cores with elastic functions and local edge processing.

Adaptive state management should relocate or compact state based on workload, latency, and energy objectives without violating processing guarantees. Portable semantics are needed so event time, delivery guarantees, and schema contracts remain understandable when pipelines span multiple engines and clouds. Automated data-contract verification should test producers, consumers, replay compatibility, and lineage before deployment. Observability should evolve from infrastructure metrics toward verifiable business-event SLOs that measure freshness, completion, and correctness. Energy-aware scheduling, as explored by Li *et al*., (2025), should be integrated with cost and carbon objectives. Finally, architecture tools should reconstruct live producer-consumer relationships and detect semantic coupling before it becomes a migration barrier.

Rocha (2022) emphasises that event-driven microservices require deliberate patterns for concurrency, ordering, eventual consistency, reliability, and schema design. The academic evidence reviewed here supports that practical warning and extends it: next-generation enterprise systems need an explicit assurance model that connects event semantics with operational evidence. Architecture quality will increasingly be judged not by whether a platform can process millions of events, but by whether an organisation can explain, verify, and evolve what those events mean under continuous change.

10. CONCLUSION

Real-time big data streaming and event-driven architecture are fusing into a core enterprise computing model. The strongest designs use durable event transport to decouple producers and consumers, stateful stream processing to interpret continuous data, and domain-oriented services to convert events into business outcomes. Their success depends on correctness mechanisms that are often invisible in simple architecture diagrams: event-time

semantics, watermarks, checkpointed state, replay, idempotency, transaction boundaries, schema evolution, compensation, and observability.

The 2020-2025 literature shows that scalability is achievable across several modern frameworks, but technology choice remains workload-dependent. Event-driven microservices improve extensibility and independent scaling, yet transfer complexity into consistency, tracing, governance, and system evolution. Serverless and edge computing add elastic and low-latency options, while complex-event and low-code approaches enable real-time decisions more accessible.

Regarding next-generation enterprise systems, the practical goal should therefore be a governed event fabric rather than a collection of disconnected streaming tools. Transport, computation, domain logic, reliability, governance, and observability should be designed as separable but coordinated planes. When these planes are treated as one correctness-resilience loop, enterprises can build systems that are more than fast, but also recoverable, explainable, evolvable, and trustworthy Abstract.

Real-time enterprise computing is shifting from periodic data movement and tightly coupled request-response integration toward continuous event streams, independently scalable services, and stateful processing that reacts as business conditions change. This review synthesises research published from 2020 to 2025 on big data streaming, event-driven architectures, microservices, transactional stream processing, observability, and serverless execution. The aim is to explain how these areas converge into a practical architecture for next gen enterprise systems rather than treating streaming as an isolated analytics layer. A structured integrative review was carried out using DOI-verifiable peer-reviewed sources from major computing publishers and journals, with thirty studies retained because they addressed architectural design, streaming semantics, scalability, reliability, consistency, deployment, or operational governance. Event-time semantics, watermarks, durable logs, checkpointed state, idempotent consumption, schema evolution, distributed transaction models, and end-to-end observability collectively determine whether a system can produce timely and trustworthy outcomes. Comparative evidence also indicates that no stream-processing framework is universally superior; workload shape, state intensity, deployment topology, and cost constraints materially affect engineering choices. Event-driven microservices improve autonomy and extensibility, but they transfer complexity into data uniformity, asynchronous failure recovery, tracing, and governance. Serverless and edge patterns can lessen operational burden and network delay, yet cold starts and distributed state remain limiting factors. The review proposes a layered enterprise event fabric and a correctness-resilience loop that unify transport, stream computation, domain choreography, data governance, and operational assurance. The resulting research agenda emphasises adaptive state management, portable semantics, energy-aware scheduling, automated data contracts, and verifiable real-time service-level objectives.

REFERENCES

- Akidau, T., Begoli, E., Chernyak, S., Hueske, F., Knight, K., Knowles, K., Mills, D., & Sotolongo, D. (2021). Watermarks in stream processing systems: Semantics and comparative analysis of Apache Flink and Google Cloud Dataflow. Proceedings of the VLDB Endowment, 14(12), 3135-3147. https://doi.org/10.14778/3476311.3476389
- Apolinario, D. R. F., & de Franca, B. B. N. (2021). A method for monitoring the coupling evolution of microservice-based architectures. Journal of the Brazilian Computer Society, 27, 17. https://doi.org/10.1186/s13173-021-00120-y
- Bharti, U., Goel, A., & Gupta, S. C. (2023). ReactiveFnJ: A choreographed model for fork-join workflow in serverless computing. Journal of Cloud Computing, 12, 63. https://doi.org/10.1186/s13677-023-00429-3
- Bogner, J., Fritzsch, J., Wagner, S., & Zimmermann, A. (2021). Industry practices and challenges for the evolvability assurance of microservices. Empirical Software Engineering, 26, 104. https://doi.org/10.1007/s10664-021-09999-9
- Bordin, M. V., Griebler, D., Mencagli, G., Geyer, C. F. R., & Fernandes, L. G. (2020). DSPBench: A suite of benchmark applications for distributed data stream processing systems. IEEE Access, 8, 222900-222917. https://doi.org/10.1109/ACCESS.2020.3043948
- Camilli, M., & Russo, B. (2022). Modeling performance of microservices systems with growth theory. Empirical Software Engineering, 27, 39. https://doi.org/10.1007/s10664-021-10088-0
- Fragkoulis, M., Carbone, P., Kalavri, V., & Katsifodimos, A. (2024). A survey on the evolution of stream processing systems. The VLDB Journal, 33(2), 507-541. https://doi.org/10.1007/s00778-023-00819-8
- Gonzalez-Aparicio, M. T., Younas, M., Tuya, J., & Casado, R. (2023). A transaction platform for microservices-based big data systems. Simulation Modelling Practice and Theory, 123, 102709. https://doi.org/10.1016/j.simpat.2022.102709
- Henning, S., & Hasselbring, W. (2021). Theodolite: Scalability benchmarking of distributed stream processing engines in microservice architectures. Big Data Research, 25, 100209. https://doi.org/10.1016/j.bdr.2021.100209

- Henning, S., & Hasselbring, W. (2024). Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud. Journal of Systems and Software, 208, 111879. https://doi.org/10.1016/j.jss.2023.111879
- Karabey Aksakalli, I., Celik, T., Can, A. B., & Tekinerdogan, B. (2021). Deployment and communication patterns in microservice architectures: A systematic literature review. Journal of Systems and Software, 180, 111014. https://doi.org/10.1016/j.jss.2021.111014
- Khriji, S., Benbelgacem, Y., Cheour, R., El Houssaini, D., & Kanoun, O. (2022). Design and implementation of a cloud-based event-driven architecture for real-time data processing in wireless sensor networks. The Journal of Supercomputing, 78, 3374-3401. https://doi.org/10.1007/s11227-021-03955-6
- Kohyarnejadfard, I., Aloise, D., Azhari, S. V., & Dagenais, M. R. (2022). Anomaly detection in microservice environments using distributed tracing data analysis and NLP. Journal of Cloud Computing, 11, 25. https://doi.org/10.1186/s13677-022-00296-4
- Kontaxakis, A., Giatrakos, N., Sacharidis, D., & Deligiannakis, A. (2023). And synopses for all: A synopses data engine for extreme scale analytics-as-a-service. Information Systems, 116, 102221. https://doi.org/10.1016/j.is.2023.102221
- Li, H., Li, J., Duan, X., & Xia, J. (2025). Energy-aware scheduling and two-tier coordinated load balancing for streaming applications in Apache Flink. Future Generation Computer Systems, 166, 107681. https://doi.org/10.1016/j.future.2024.107681
- Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., Shen, J., & Babar, M. A. (2021). Understanding and addressing quality attributes of microservices architecture: A systematic literature review. Information and Software Technology, 131, 106449. https://doi.org/10.1016/j.infsof.2020.106449
- Mayr-Dorn, C., Pereszteghy, B., Holzweber, J., & Mayrhofer, M. (2021). Distributed tracing of OPC UA method calls. Elektrotechnik & Informationstechnik, 138, 349-354. https://doi.org/10.1007/s00502-021-00903-3
- Moina-Rivera, W., Garcia-Pineda, M., Claver, J. M., & Gutierrez-Aguado, J. (2023). Event-driven serverless pipelines for video coding and quality metrics. Journal of Grid Computing, 21, 20. https://doi.org/10.1007/s10723-023-09647-0
- Moreno-Vozmediano, R., Huedo, E., Montero, R. S., & Llorente, I. M. (2023). Latency and resource consumption analysis for serverless edge analytics. Journal of Cloud Computing, 12, 108. https://doi.org/10.1186/s13677-023-00485-9
- Ntentos, E., Zdun, U., Plakidas, K., Genfer, P., Geiger, S., Meixner, S., & Hasselbring, W. (2021). Detector-based component model abstraction for microservice-based systems. Computing, 103, 2521-2551. https://doi.org/10.1007/s00607-021-01002-z
- Ortiz, G., Boubeta-Puig, J., Criado, J., Corral-Plaza, D., Garcia-de-Prado, A., Medina-Bulo, I., & Iribarne, L. (2022). A microservice architecture for real-time IoT data processing: A reusable Web of Things approach for smart ports. Computer Standards & Interfaces, 81, 103604. https://doi.org/10.1016/j.csi.2021.103604
- Overeem, M., Spoor, M., Jansen, S., & Brinkkemper, S. (2021). An empirical characterization of event sourced systems and their schema evolution: Lessons from industry. Journal of Systems and Software, 178, 110970. https://doi.org/10.1016/j.jss.2021.110970
- Poojara, S. R., Dehury, C. K., Jakovits, P., & Srirama, S. N. (2022). Serverless data pipeline approaches for IoT data in fog and cloud computing. Future Generation Computer Systems, 130, 91-105. https://doi.org/10.1016/j.future.2021.12.012
- Rocha, H. F. O. (2022). Practical event-driven microservices architecture: Building sustainable and highly scalable event-driven microservices. Apress. https://doi.org/10.1007/978-1-4842-7468-2
- Rosa-Bilbao, J., Boubeta-Puig, J., & Rutle, A. (2023). CEPEDALoCo: An event-driven architecture for integrating complex event processing and blockchain through low-code. Internet of Things, 22, 100802. https://doi.org/10.1016/j.iot.2023.100802
- Toliopoulos, T., Gounaris, A., Tsichlas, K., Papadopoulos, A., & Sampaio, S. (2020). Continuous outlier mining of streaming data in Flink. Information Systems, 93, 101569. https://doi.org/10.1016/j.is.2020.101569
- van Dongen, G., & Van den Poel, D. (2020). Evaluation of stream processing frameworks. IEEE Transactions on Parallel and Distributed Systems, 31(8), 1845-1858. https://doi.org/10.1109/TPDS.2020.2978480
- Wadhwa, R. (2023). Designing scalable enterprise systems using event-driven microservices and distributed data architecture for high-throughput business applications. International Journal of Computer Engineering and Technology, 14(3), 323-337. https://doi.org/10.34218/IJCET_14_03_030
- Xue, G., Deng, S., Liu, D., & Yan, Z. (2021). Reaching consensus in decentralized coordination of distributed microservices. Computer Networks, 187, 107786. https://doi.org/10.1016/j.comnet.2020.107786
- Zhang, S., Soto, J., & Markl, V. (2024). A survey on transactional stream processing. The VLDB Journal, 33, 451-479. https://doi.org/10.1007/s00778-023-00814-z.