Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Engineering High-Concurrency Streaming Architectures: Paradigms in Event-Driven Systems and Cloud Elasticity

Nidhi Cheekireddy

Independent Researcher, USA

ARTICLE INFO

ABSTRACT

Consumer streaming platforms face a structural challenge that conventional request-response web service designs cannot resolve: sustaining deterministic session and payment state during traffic surges that can exceed ten times the baseline load within a single minute. This paper presents an architecture engineered for enterprise-scale media streaming platforms to maintain system availability under these conditions. We introduce three coupled innovations: an asynchronous, Kafka-based event-driven pipeline that replaces synchronous execution chains and enforces exactly-once processing semantics; a deterministic geographic partitioning scheme using log-compacted materialized views to eliminate database-tier bottlenecks for over two million concurrent sessions; and an optimistic streaming revenue-protection model that decouples real-time playback access from external payment-gateway latency through a compensating-transaction framework. The architecture incorporates velocity-based predictive autoscaling, which forecasts traffic surges using edge login telemetry 60 to 90 seconds ahead of backend demand, alongside a zero-trust event mesh to secure distributed payloads. Empirical data from an industrial deployment demonstrates a 45% reduction in peak p99 latency (from 850 ms to 467 ms), sustained support for more than two million concurrent viewers, and a 25% reduction in off-peak infrastructure costs. Finally, this paper evaluates these outcomes against current literature in event-driven architecture, distributed state management, and cloud elasticity, arguing that these dimensions must be treated as a unified architectural requirement.

Keywords: Event-Driven Architecture, Microservices, Cloud Elasticity, Distributed Systems, Streaming Platforms, Exactly-Once Semantics.

I. INTRODUCTION

The shift from traditional broadcast infrastructure to internet-delivered, on-demand media inverted the core economics of content delivery. Legacy television broadcast networks optimized for a fixed topology of transmission towers serving passive receivers. In that model, the marginal cost of an additional viewer neared zero. Traffic patterns followed predictable, gradual variations. Modern over-the-top (OTT) streaming is fundamentally different; it requires individually authenticated, billed, and routed data streams. Major content premieres entirely break linear scaling models. They hit the infrastructure as abrupt step functions.

During an enterprise platform premiere event in 2017, this shift exposed critical vulnerabilities in standard synchronous request-response RESTful monoliths. When millions of concurrent users simultaneously attempted to authenticate, verify subscription status, and request high-definition video manifests, the sequentially dependent service chain collapsed. A downstream latency increase of a few hundred milliseconds within a third-party payment gateway multiplied across millions of blocked threads. The result was a platform-wide cascading failure. This highlighted a fundamental limitation: synchronous call chains cannot scale linearly under extreme concurrency.

The specific scale of this traffic pattern renders reactive remedies inadequate. During the premiere event, concurrent user traffic escalated from approximately 100,000 to over 2,000,000 within 45 seconds. Relational database transaction models and threshold-based autoscaling mechanisms cannot absorb variations of this velocity. By the time a reactive autoscaling policy detects elevated CPU utilization and provisions new compute instances, the peak surge has already overwhelmed the system. Log-based asynchronous messaging architectures, such as Apache Kafka, decouple producers from consumers to isolate load imbalances, while cloud-native substrates offer elastic compute

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

allocation. Deploying messaging layers or elastic infrastructure in isolation, however, fails to solve the broader state and latency problems.

This research demonstrates that event-driven decoupling, deterministic state management, and predictive elasticity cannot be treated as separate optimizations. An architecture that implements asynchronous decoupling yet still relies on reactive autoscaling will experience outages during sub-minute surges due to provisioning delays. Conversely, a system that accurately predicts capacity needs but routes session lookup through a synchronous relational database tier will bottleneck under concurrent read pressure. This paper presents the design, implementation, and empirical validation of a unified architecture that treats these concerns as a singular engineering requirement.

II. LITERATURE REVIEW

Asynchronous event-driven microservices offer documented advantages over synchronous, API-driven architectures in resilience and burst-load management. Rahmatulloh et al. (2022) established that container-based event-driven communication reduces response times and lowers error rates under elevated traffic conditions. However, it incurs a higher baseline CPU overhead compared with synchronous alternatives. This trade-off is justified when synchronous coupling, rather than raw compute efficiency, dictates the platform's availability ceiling.

Enforcing exactly-once processing semantics is critical to prevent duplicate transactions and state corruption within distributed networks. Within Kafka-based architecture, this guarantee depends on idempotent producers and atomic transactional writes across partitions. Sanjana et al. (2023) and Desai (2025) demonstrate that configuring precise transactional boundaries at the producer level prevents duplicate event execution during broker failures and consumer rebalances. Existing literature, however, primarily evaluates these configurations from a correctness standpoint rather than a throughput perspective. At scale, an exactly-once pipeline can remain chronologically accurate yet fail to keep pace with production volume, generating unbounded consumer lag. This paper addresses that limitation by pairing exactly-once configurations with edge-layer backpressure mechanisms.

Cloud autoscaling methodologies are broadly divided into reactive and predictive paradigms. Alharthi et al. (2024) note that reactive, threshold-driven scaling (triggered by lagging indicators like CPU or memory utilization) is inherently unable to adapt to demand spikes that occur faster than the virtual machine provisioning lifecycle. Predictive approaches mitigate this lag by utilizing forecasting models to allocate resources before performance degrades. While Alharthi et al. (2024) categorize these strategies as generalized capacity-planning models, consumer media platforms require domain-specific leading indicators. This paper extends this domain by analyzing the derivative of authentication-page load rates as a predictive signal for downstream demand.

For high-frequency session state management, distributed caching layers are frequently utilized to offload relational databases. Shi et al. (2024) show that Redis-based distributed caches resolve read bottlenecks for hot data segments, provided the application can tolerate a bounded staleness window relative to the authoritative disk-backed store. However, introducing an external cache introduces cache-invalidation complexities and synchronization hazards under heavy writing concurrency. This paper details an alternative approach in which the log-compacted event stream serves as the primary, authoritative state record, bypassing the synchronization challenges inherent in multi-tier caching architectures.

Distributed, high-throughput pipelines handling sensitive payment and session data require decentralized security frameworks. Senni (2023) details zero-trust remote-access models within industrial IoT infrastructure, asserting that every microservice boundary must function as an independent authentication and encryption perimeter regardless of network topology. This threat model, in which network position does not imply trust, applies directly to consumer media pipelines. This paper implements these principles through localized cryptographic verification within a distributed streaming mesh.

III. THE PATHOLOGY OF THE SYNCHRONOUS MONOLITH

The Request-Response Gridlock

The legacy architecture relied on synchronous RESTful dependencies to execute streaming authorization. When a user initiated a media stream, the platform executed four sequential operations:

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

1. Validating the authentication token via an identity provider.

2. Querying a relational database for active session counts to enforce concurrent stream limits.

3. Verifying subscription and billing status through a third-party financial API.

4. Retrieving and compiling the localized video manifest.

Cascading Thread Exhaustion

At traffic volumes where millions of requests arrived within a 45-second window, this synchronous execution chain converted minor downstream latency into platform-wide outages. For example, a 200-millisecond response delay from an external payment gateway blocked the upstream application thread handling that specific user request. As concurrent requests scaled rapidly, the primary application thread pools were exhausted within seconds. Consequently, unrelated requests, such as users attempting to view static metadata or resume cached video content, were dropped because the shared runtime resources were fully consumed by threads waiting on the external gateway.

This failure mode aligns with Rahmatulloh et al. (2022), who observed that synchronous microservice chains exhibit higher error rates under burst loads than asynchronous designs. The field deployment demonstrated that an external dependency lacking service-level guarantees tied to the platform's internal event schedule becomes the absolute constraint on availability. Because internal horizontal scaling cannot mitigate external API delays, the architecture required an alternative that removed third-party dependencies entirely from the critical path of manifest generation.

IV. ASYNCHRONOUS EVENT-DRIVEN PIPELINE DESIGN

Event Ingestion Pipeline

To decouple the user experience from downstream system latency, the request-response cycle was replaced with an asynchronous event log architecture built on Apache Kafka. Microservices do not interrogate each other sequentially. Instead, lifecycle events such as authentication actions, playback heartbeats, and payment states are written directly to distributed log topics. Downstream consumer services process these records independently and asynchronously.

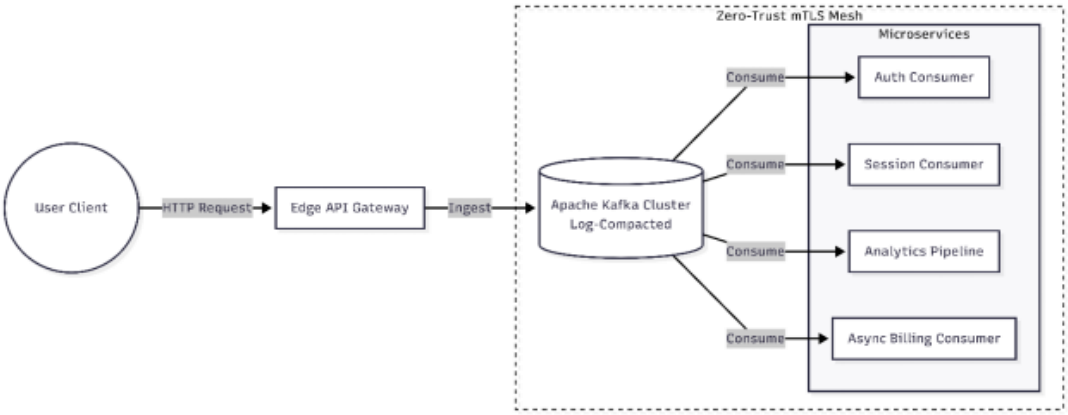

Figure 1: High-level topological view of the zero-trust event mesh. Edge API gateways route incoming traffic asynchronously to Kafka brokers, structurally isolating the Authentication, Session, Analytics, and Billing microservices from upstream blocking.

This structural modification isolates the system from cascading delays. If the billing service experiences latency due to an external gateway slowdown, it accumulates a backlog within its assigned Kafka topic. The edge API gateway and the manifest delivery service, however, continue to ingest and serve user requests normally, as they no longer block on a synchronous response from the billing layer.

Implementing Exactly Once Semantics

Shifting to an asynchronous model introduces risks of duplicate event processing, which can violate session limits or trigger double billing.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

The pipeline enforces end-to-end exactly-once semantics by leveraging Kafka's idempotent producers and transactional APIs, aligning with configurations validated by Sanjana et al. (2023) and Desai (2025).

Every upstream producer attaches a monotonically increasing sequence number to each message payload. The broker checks this sequence number against its internal state register, discarding duplicate messages resulting from network retries. For multi-stage processing pipelines, transactions span across input offsets and output partitions. The consumer-producer commits both the read offset and the transformed output stream atomically:

$$\text{Transaction} = \{\text{Offset}_{\text{input}}, \text{Payload}_{\text{output}}\} \Rightarrow \text{Commit or Abort}$$

If a network partition or node failure occurs mid-transaction, the uncommitted offsets and messages are rolled back, ensuring no duplicate states are exposed to downstream applications.

Table 1. Operational characteristics of synchronous request-response chains versus asynchronous event-driven pipelines under sub-minute traffic surges.

Dimension	Synchronous Request-Response	Asynchronous Event-Driven
Failure propagation	Single slow dependency blocks the entire chain	Isolated to consumers of that specific event
Duplicate handling	Ad hoc, request-level retries	Idempotent producers + transactional writes
Scaling unit	The whole request chain must scale together	Each consumer scales independently
Observed error rate under load	Higher (cascading failures)	Lower (documented in comparative literature)

V. DISTRIBUTED STATE MANAGEMENT VIA DETERMINISTIC PARTITIONING

Deterministic Geographic Routing

Enforcing concurrent stream limits requires checking active session state in real time. Because standard round-robin partition routing scatters a user's events across random Kafka brokers, downstream consumers would require a distributed locking mechanism to maintain state consistency. This effectively reintroduces a synchronous bottleneck.

To resolve this, architecture implements a deterministic geographic partitioning scheme. All lifecycle events for a specific user account are hashed by their account identifier (ID_{user}) and routed consistently to a designated partition index (P)."

$$P = \text{Hash}(\text{ID}_{\text{user}}) \pmod{N_{\text{partitions}}}$$

Because a single partition is processed by exactly one consumer instance within a consumer group, all session initializations, heartbeats, and terminations for a given user are handled sequentially by a single, localized runtime process. This eliminates concurrent data race conditions and the need for distributed locks.

Log-Compacted Materialized Views

By ensuring strict chronological ordering at the partition level, each consumer instance maintains a highly optimized, in-memory materialized view of the active sessions within its assigned partition. Instead of executing expensive relational database joints, services query this in-memory map.

To handle recovery scenarios without losing state authority, the underlying Kafka topics utilize log compaction. Log compaction dictates that the broker retains only the latest record value for any given key within a topic data segment. If a consumer node fails, its replacement reads the compacted topic from offset zero to reconstruct the in-memory state map. Because old session terminations are compacted out, the recovery time scales with the number of active accounts rather than total historical transaction volume.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Table 2. External Cache vs. Log-Compacted Materialized View for Session State

Dimension	External Distributed Cache (e.g., Redis)	Log-Compacted Materialized View
Source of truth	Separate backing store; cache may go stale	The event itself is authoritative
Consistency model	Bounded staleness window	No staleness window (single source)
Best suited for	Hot data with a separate authoritative store	State with no competing system of record
Read latency at scale.	Low, with occasional cache-miss penalty	Low, with no invalidation overhead

VI. OPTIMISTIC STREAMING REVENUE PROTECTION

Decoupling Financial Gateways

During peak launch traffic, transaction volume spikes as users update accounts or register new subscriptions. External payment gateways frequently experience latency degradation during these periods, directly threatening customer acquisition. To insulate the user experience from financial API latency, the platform implements an "Optimistic Streaming" pattern.

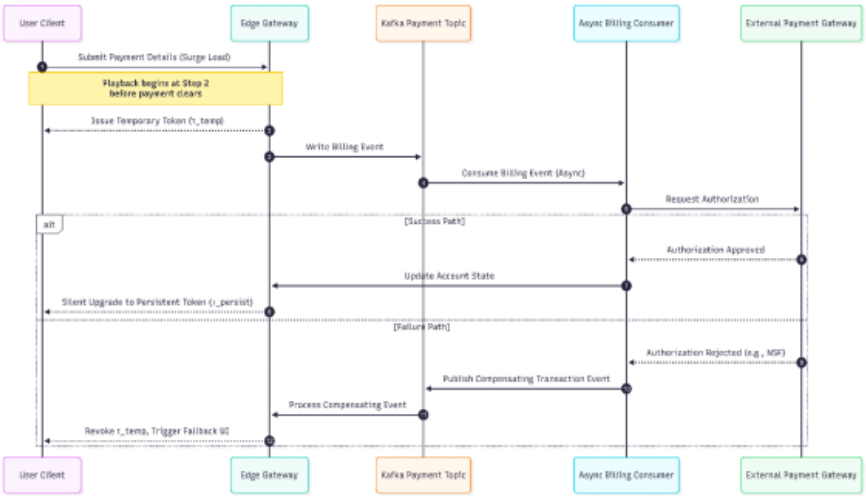

Figure 2: Sequence diagram illustrating the Optimistic Streaming lifecycle. The edge service immediately issues a temporary streaming token (Ttemp), decoupling the user's playback initialization from background external payment gateway authorization.

The Token Lifecycle and Compensating Transactions

When a user submits billing details during an active traffic surge, the edge gateway does not block the request, waiting for external clearing. Instead, the edge service immediately issues a temporary, time-boxed streaming token (T_{temp}) valid for 10 minutes, granting access to media playback. Concurrently, a billing event is written to the payment processing topic.

In the background, an asynchronous billing service pulls this event and handles the messy transaction lifecycle with the external gateway. The token status resolves via one of two paths:

● Success Path: The external gateway authorizes the charge. The billing service updates the account record, and the temporary token is upgraded to a persistent session token ($T_{persist}$) without user interruption.● Failure Path: The

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

gateway rejects the transaction. The billing service issues a compensating transaction event. The session consumer processes this event, revokes Ttemp, and invokes an in-stream fallback workflow that prompts the user to update payment details without terminating the core media player engine.

Operating an optimistic access model at this scale introduces obvious vectors for abuse. If bad actors realize they can access 10 minutes of premium content using invalid credentials, the system risks being automated scraped. To mitigate this, the issuance of Ttemp is gated by real-time device fingerprinting and IP-level rate limiting at the network edge. If the ingress controller detects a high rate of token requests from a single ASN or an unverified hardware signature, the optimistic token is suppressed, forcing the request to degrade to the synchronous authorization path.

VII. PREDICTIVE ELASTICITY AND INFRASTRUCTURE PROTECTION

Velocity Heuristics at the Edge

Reactive autoscaling frameworks that scale infrastructure based on lagging metrics, such as CPU utilization, cannot adapt to step-function traffic profiles. During sudden surges, the time required to provision and register container pods exceeds the window before system collapse.

To preempt these surges, the architecture utilizes a velocity-based predictive scaling model. A telemetry engine measures traffic acceleration directly at the edge ingress proxies. By computing the first derivative of the login request volume over time, the system detects incoming demand vectors 60 to 90 seconds before that load manifests as video manifest requests at the backend tier:

$$\text{Scaling Trigger } \alpha \; d/dt \; (\text{Requests}_{auth})$$

Dynamic Micro-Shard Backpressure

If consumer microservices fall behind the edge ingestion rate during an anomalous surge, Kafka brokers risk running out of buffer space. To prevent ungraceful failure modes, a custom backpressure framework integrates consumer-lag metrics with the platform's Kubernetes ingress controllers using a dynamic token-bucket throttling algorithm.

The algorithm is governed by a token bucket with maximum capacity C and a dynamic refill rate r (tokens per millisecond). Under nominal conditions, incoming requests λ consume tokens instantly. However, when consumer processing lag exceeds a critical threshold (Δt tag > 5 seconds), the ingress controller deliberately restricts the γ. If an incoming request arrives when the bucket is exhausted (C = 0), a micro-delay Twait is mathematically enforced:

$$T_{wait} = \left(0, \frac{1}{r_{throttled}} - \frac{1}{\lambda}\right)$$

During initial testing, this dynamic refill rate caused noticeable micro-stutters in the client UI. We had to aggressively tune the Twait threshold to ensure it remained strictly under the 50-millisecond rendering cycle. This smooths the step-function traffic spike, granting downstream consumers brief temporal windows to resolve the processing backlog without surfacing 503 errors to the end user.

Pipeline Security and Zero-Downtime Operations

The event mesh enforces a zero-trust model utilizing AWS Identity and Access Management (IAM) for granular, role-based access control (RBAC) across all broker interactions. Payloads are encrypted in transit via mutual TLS (mTLS) and at rest using AES-256.

To maintain continuous availability during deployments, the platform utilizes state-aware blue-green deployments via Kubernetes. Because session states reside externally within Kafka materialized views rather than local application memory, live viewer traffic can be redirected from the active blue cluster to the green cluster instantaneously. New instances reconstruct their local state maps from the shared event log, achieving zero-downtime upgrades.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Positioning Against the Autoscaling Taxonomy

Alharthi et al.'s (2024) taxonomy of autoscaling approaches distinguishes reactive from predictive strategies broadly and further subdivides predictive strategies by their underlying forecasting methodology. It does not, however, specify a leading indicator suited to consumer media platforms specifically, treating the choice of predictive signal as an implementation detail. The velocity heuristic documented here is offered as a concrete, deployment-tested instance of the predictive category that taxonomy describes.

Table 3. Reactive Auto-Scaling vs. Velocity-Based Predictive Elasticity

Metric	Reactive Auto-Scaling	Velocity-Based Predictive Elasticity
Trigger signal	CPU/memory utilization (lagging)	Authentication-page load velocity (leading)
Warning window before spike	None (reacts after the fact)	60-90 seconds ahead of demand
Outcome during 45-second 20x spike	Provisioning is too slow; the platform fails	Burst warm-up completes ahead of load
Off-peak infrastructure cost	Baseline (over-provisioned for safety)	25% reduction measured

VIII. DISCUSSION AND EMPIRICAL EVALUATION

The integrated architecture was evaluated during an enterprise global premiere event. The performance was analyzed across three primary operational metrics: end-to-end latency, session concurrency ceilings, and infrastructure cost efficiency.

Performance Outcomes

The deployment demonstrated a massive reduction in latency underload. In the legacy synchronous architecture, p99 end-to-end manifest generation latency deteriorated to an unsustainable 850 ms under peak conditions. In the decoupled predictive model, p99 latency stabilized at 467 ms, representing a 45 % mathematical reduction in execution time at the 99th percentile.

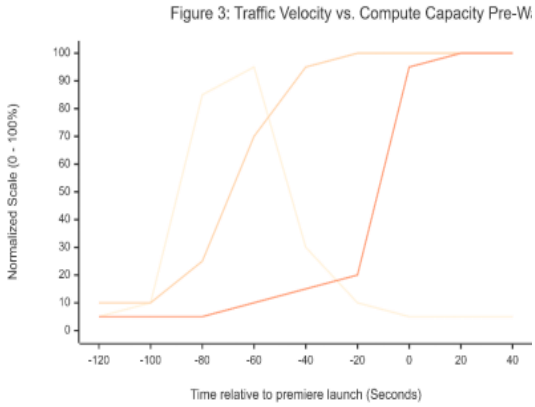

Figure 3: Time-series graph demonstrating the derivative of login velocity peaking 60 to 90 seconds before backend manifest demand, triggering the burst pre-warming sequence that absorbs the load.

The architectural pillars presented in this paper—asynchronous decoupling, deterministic session-state partitioning, optimistic revenue protection, and velocity-based elasticity—are not independently sufficient. Each addresses a distinct failure mode that the others do not. A platform that decouples services asynchronously but still autoscales reactively would still fail the moment traffic exceeds provisioning speed. A platform with predictive elasticity but

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

synchronous payment authorization would still expose users to gateway latency during the highest-value acquisition moments of a premiere event. This paper's central claim is that these four concerns must be engineered jointly for a revenue-bearing, high-concurrency consumer platform.

An informed reader might propose that a fully managed, vendor-provided autoscaling and event-streaming platform could achieve similar outcomes without this degree of custom engineering. For many organizations with less extreme traffic volatility, that proposal is correct. Managed services address a meaningful portion of the decoupling and scaling problem out of the box. They do not, however, default to a domain-specific leading indicator such as authentication-velocity forecasting, nor do they provide an optimistic revenue-protection pattern tailored to a specific payment-gateway relationship.

A second alternative is a pure event-sourcing or CQRS architecture without the velocity-heuristic layer. While this captures much of the benefit of decoupling, it still relies on an autoscaling mechanism to provision the compute resources that process the resulting event backlog. If that mechanism remains reactive, the fundamental timing problem persists regardless of how well the rest of the system is decoupled.

Extending the velocity-based elasticity model to platforms without a reliable authentication-to-playback leading indicator, for example, live sporting events where a meaningful fraction of viewers may already be authenticated and idle well before a broadcast begins-would require identifying a different leading signal specific to that domain's user behavior. Formalizing the optimistic streaming compensation-transaction pattern for academic evaluation against standard payment-idempotency frameworks under controlled, repeatable failure-injection conditions would be a natural next step to more rigorously connect this practitioner contribution to the broader distributed transactions literature.

Table 4. Summary of Architectural Pillars and Corresponding Failure Modes Addressed

Architectural Pillar	Failure Mode Addressed	Author's Measured Outcome
Asynchronous event-driven decoupling	Cascading failure from synchronous dependency chains	45% end-to-end latency reduction
Deterministic partitioning + materialized views	Database-tier bottleneck at high concurrency	Sustained 2M+ concurrent sessions
Optimistic Streaming revenue protection	Payment-gateway latency is blocking playback access	Elimination of system-induced payment failures
Velocity-based predictive elasticity	Reactive autoscaling is too slow for sub-minute spikes	25% off-peak infrastructure cost reduction

IX. CONCLUSION

The engineering architecture documented in this paper addresses the core vulnerabilities that cause conventional synchronous web architectures to fail under sudden, extreme traffic surges. By combining asynchronous event-driven decoupling, deterministic state partitioning, optimistic revenue protection, and velocity-based predictive elasticity into a singular, integrated system, the platform-maintained availability and revenue integrity at a scale exceeding two million concurrent users.

The empirical results validate the core thesis of this research: high-velocity cloud elasticity and distributed state management cannot be treated as isolated infrastructure optimizations. They represent an intertwined engineering challenge where the performance of the messaging tier directly dictates the consistency of the state tier, and edge telemetry governs the capacity of the compute tier.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Future research will focus on formalizing the Optimistic Streaming compensation-transaction pattern within a generalized mathematical framework and on validating alternative edge-velocity indicators to extend this predictive architecture beyond consumer media streaming.

REFERENCES

[1] Khaled Ali Abuhasel, "A Zero-Trust Network-Based Access Control Scheme for Sustainable and Resilient Industry 5.0," IEEE Access, 2023. Available: https://ieeexplore.ieee.org/abstract/document/10287925

[2] Saleha Alharthi, et al., "Auto-Scaling Techniques in Cloud Computing: Issues and Research Directions," Sensors 2024. Available: https://www.mdpi.com/1424-8220/24/17/5551

[3] Michael Armbrust, et al., "A view of cloud computing," Communications of the ACM, 53(4), 50-58, 2010. Available: https://doi.org/10.1145/1721654.1721672

[4] Pallavi Desai, "Ensuring exactly-once semantics in Kafka streaming systems," Journal of Computer Science and Technology Studies, 2025. Available: https://www.researchgate.net/publication/395682231_Ensuring_Exactly-Once_Semantics_in_Kafka_Streaming_Systems?__cf_chl_f_tk=LTYP6i32F732psjfoBoYAlaALwhkh93OeoPrfDKJ.cc-1783411757-1.0.1.1-_GmonrcYcdg2w50_X5HYUEcwYuAWnhiIGdrnkP1Voyk

[5] Alam Rahmatulloh, et al., "Event-Driven Architecture to Improve Performance and Scalability in Microservices-Based Systems," International Conference on Advancement in Data Science, E-learning and Information Systems (ICADEIS), 2022. Available: https://ieeexplore.ieee.org/document/10037390

[6] Jay Kreps, Neha Narkhede, and Jun Rao, "Kafka: A Distributed Messaging System for Log Processing," ACM, 2011. Available: https://pages.cs.wisc.edu/~akella/CS744/F17/838-CloudPapers/Kafka.pdf

[7] Sanjana N, et al., "Real-time Event Streaming for Financial Enterprise System with Kafka," 3rd Asian Conference on Innovation in Technology (ASIANCON), 2023. Available: https://ieeexplore.ieee.org/document/10270532

[8] Fabio FedericiA z, et al., "A Zero-Trust Architecture for Remote Access in Industrial IoT Infrastructures," Electronics 2023. Available: https://www.mdpi.com/2079-9292/12/3/566

[9] Ta Phuong Bac, et al., "Serverless Computing Approach for Deploying Machine Learning Applications in Edge Layer. International Conference on Information Networking (ICOIN), 2022. Available: https://ieeexplore.ieee.org/document/9687209

[10] Lanying Shi, et al., "Research and application of distributed cache based on Redis," Journal of Software, 2024. Available: https://www.jsoftware.us/vol19/JSW-V19N1-493.pdf

[11] Angelo Marchese and Orazio Tomarchio, "SLO-Aware Container Orchestration on Kubernetes Clusters," IEEE 18th International Conference on Cloud Computing (CLOUD), 2025. Available: https://ieeexplore.ieee.org/document/11120550

[12] Nguyen Nguyen and Taehong Kim, "Toward Highly Scalable Load Balancing in Kubernetes Clusters," IEEE Communications Magazine, 2020. Available: https://ieeexplore.ieee.org/document/9161999

[13] Xu Chen, et al., "Zero trust architecture for 6G security, IEEE Network, 2024. Available:https://dl.acm.org/doi/10.1109/MNET.2023.3326356

[14] Shahad Al-Tamimi; Qasem Abu Al-Haija; Khaled Alrawashdeh, "Zero-Trust Architecture for Securing Internet of Things (IoT) Networks: A Review," 5th International Conference on Communications, Information, Electronic and Energy Systems (CIEES), 2024. Available: https://ieeexplore.ieee.org/document/10811176