Sciforce

International Journal of Computer Science and Data Engineering

Journal homepage: www.sciforce.org
ISSN : 3066-6813

Open Access Research Article

Large-Scale Architecture for Retail Platforms Using Cloud-Native Big Data Systems

Karthik Perikala*

Technology Leader, The Home Depot., United States

Abstract

Modern retail platforms operate at extreme scale, serving millions of customers while continuously integrating product, pricing, inventory, fulfillment, and promotional data from heterogeneous systems. Traditional relational and document-oriented data models struggle to represent the highly connected and rapidly evolving nature of retail ecosystems under stringent latency and availability requirements.

This paper presents the design principles, system architecture, and operational characteristics of a large-scale retail data platform built using

cloud-native big data technologies. The system sustains read workloads of approximately 10,000 requests per second while supporting write ingestion rates between 300,000 and 500,000 updates per second, enabling near real-time freshness for business-critical signals such as pricing and inventory.

We describe how batch and streaming ingestion pipelines are unified within a single data architecture, how horizontal scalability and fault tolerance are achieved using distributed storage systems, and how the resulting platform supports efficient search, navigation, and recommendation workloads. The paper concludes with lessons learned from production operation and implications for future generative AI and personalization systems.

Keywords: Retail Systems, Big Data, Cloud Computing, Micro services, Distributed Databases

Introduction

Large-scale retail platforms face a fundamental data integration challenge: product information is distributed across dozens of in dependent operational systems, each optimized for a narrow business function. Core product catalogs, pricing engines, inventory management systems, fulfillment networks, promotional planners, and content management platforms evolve independently, yet customer-facing applications such as search, browsing, and recommendations require a unified and consistent view of this data.

Historically, retail data integration relied on relational databases or nightly batch pipelines that produced static snapshots. While effective at modest scale, these approaches introduce significant limitations in modern environments.

Batch-oriented architecture screate data staleness windows measured in hours, during which prices, availability, or promotions may be outdated. Conversely, attempting to model highly connected retail entities using normalized relational schemas leads to complex joins, increased tail latency, and limited horizontal scalability.

Received date: December 07, 2024 **Accepted date:** December 13, 2024; **Published date:** December 19, 2024

***Corresponding Author:** *Perikala, K. Technology Leader, The Home Depot., United States, E- mail: karthik.perikala2512@gmail.com*

This work explores how cloud-native big data technologies can be combined to address these challenges. By integrating distributed storage systems, real-time streaming frameworks, and micro service-based query layers, the proposed architecture delivers both scalability and low-latency access while maintaining near real-time data freshness.

System Design Flow

High-Level Processing Pipeline

The platform follows a deterministic, unidirectional processing model designed to separate ingestion, persistence, optimization, and serving concerns. This structure is essential for large-scale retail sys-tems where write intensity, data freshness, and query latency must be managed independently.

Operational data originates from heterogeneous up- stream systems such as product catalog services, pricing engines, inventory platforms, and promotion schedulers. These systems emit updates with differ-ent frequencies and consistency guarantees.

Incoming data is handled by a processing layer that performs schema validation, deduplication, and nor- malization. This layer absorbs up stream variability and ensures that downstream components operate on canonical representations.

Architectural Principles

The system design is guided by the following princi-ples:

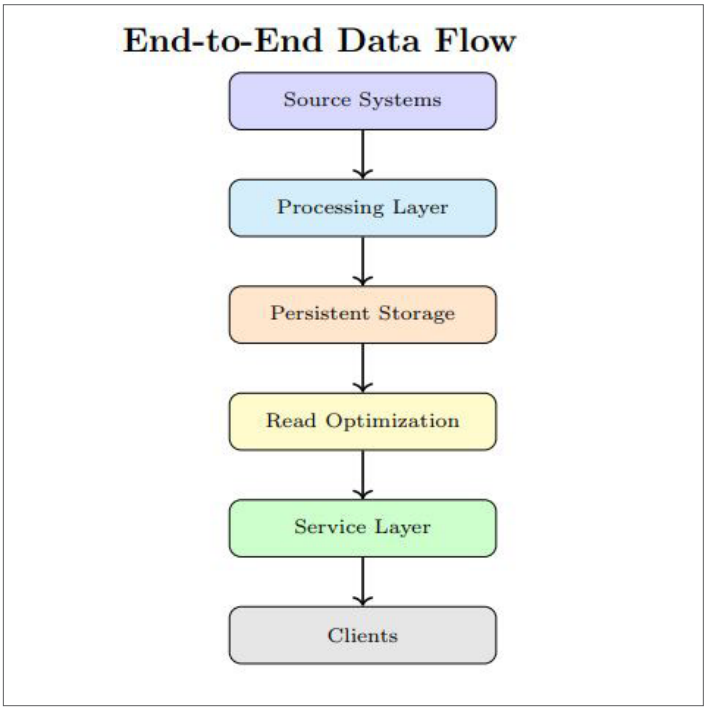
- **Uni direction al flow** to eliminate cyclic depen-dencies
- **Layer isolation** for independent scaling and fail- ure containment

Citation: Perikala. K (2024). Large-Scale Architecture for Retail Platforms Using Cloud-Native Big Data Systems. International Journal of Computer Science and Data Engineering, 1(3), 1–7 doi: https://dx.doi.org/10.55124/csdb.v1i3.268

- **Early write absorption** to protect read-heavy services

These principles simplify operational reasoning and improve system resilience under sustained load.

Operational Predictability Beyond scalability, the primary design objective is operational predictability under peak retail traffic conditions. By constraining all state mutation to well-defined ingestion stages and exposing only read-optimized interfaces to serving workloads, the system ensures that read latency remains stable even during large backfills, reprocessing events, or upstream data anomalies.



Latency Containment

By isolating write-heavy ingestion from readoptimized serving paths, the system prevents write amplification from impacting customer-facing workloads. Optimized read structures and caching ensure predictable P95 and P99 latency behavior even during peak ingestion periods.

This separation allows the platform to maintain stable tail latencies while continuously integrating near real-time operational updates.

Failure Containment and Recovery A direct benefit of the unidirectional design is strong failure containment. Ingestion failures are isolated to upstream processing stages and do not propagate into serving paths. When upstream data sources emit malformed or delayed updates, the processing layer absorbs these anomalies without impacting read availability.

Scalability Under Seasonal Load Retail traffic exhibits extreme seasonality driven by promotions, holidays, and regional demand patterns. The architectural separation between ingestion and serving allows each layer to scale independently in response to these events. Write-heavy components elastically scale during peak update windows, while readoptimized services maintain stable capacity aligned with customer traffic.

Batch Ingestion Lifecycle

By Purpose of Batch Ingestion

Batch ingestion establishes the authoritative baseline state of retail data within the platform. This includes product master records, category hierarchies, brand definitions, attribute metadata, and curated editorial content. These datasets change relatively infrequently but require strong consistency and completeness guarantees.

Unlike real-time ingestion paths that prioritize freshness, the batch pipeline is optimized for determinism, auditability, and recoverability. It ensures that all downstream services operate from a stable and verifiable snapshot of core retail entities.

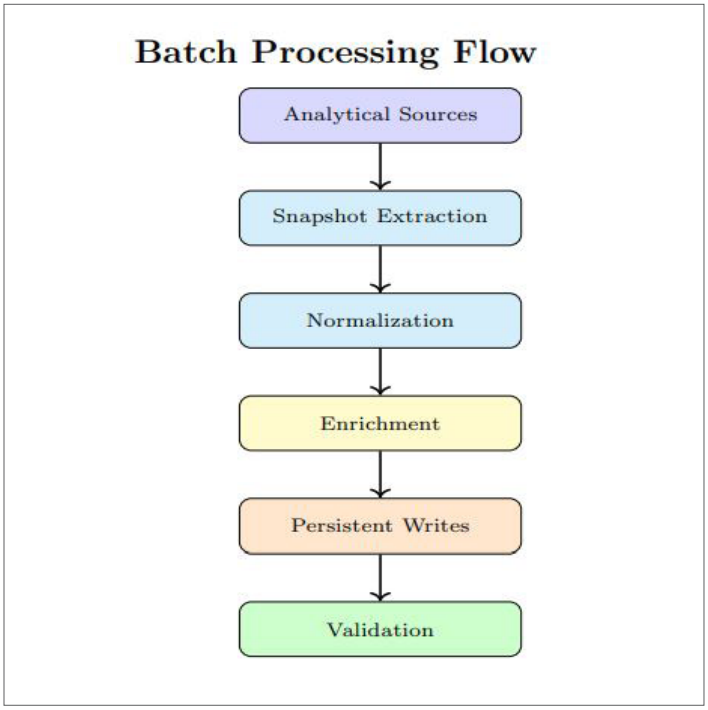
Design Objectives

The batch ingestion process is designed around the following objectives:

- **Deterministic execution** with reproducible outputs
- **Idempotent writes** to allow safe reprocessing
- **Schema enforcement** across heterogeneous sources
- **Operational isolation** from real-time serving paths

These objectives enable batch ingestion to function as a reliable foundation layer without introducing volatility into latency-sensitive systems.

Snapshot Semantics Each batch execution represents a consistent snapshot taken at a well-defined point in time. This snapshot semantics simplifies reasoning about data correctness, supports backfills, and enables controlled rollback when upstream issues are detected.



Operational Characteristics

Batch ingestion executes on fixed schedules and operates independently from customer-facing traffic. Write throughput is optimized through partitionaware batching and parallel execution, while validation stages ensure completeness and referential integrity.

Citation: Perikala. K. (2024) Large-Scale Architecture for Retail Platforms Using Cloud-Native Big Data Systems. International Journal of Computer Science and Data Engineering, 1(3), 1–7 doi: https://dx.doi.org/10.55124/csdb.v1i3.268

Failure Handling Failures during batch ingestion are isolated to the ingestion layer and do not impact serving availability. Partial outputs are discarded, and reprocessing can be triggered without requiring coordinated downtime across downstream services.

Role in System Stability By continuously reasserting a clean baseline state, batch ingestion acts as a stabilizing force within the overall architecture. It limits the blast radius of upstream data issues and provides a consistent anchor for real-time updates and serving-layer optimizations.

Real-Time Ingestion and Streaming

Motivation for Real-Time Updates

While batch ingestion establishes a stable baseline, modern retail platforms require near real-time visibility into rapidly changing operational signals. Pricing updates, inventory fluctuations, fulfillment availability, and promotion activations directly impact customer experience and revenue.

Real-time ingestion pipelines are therefore responsible for continuously propagating operational changes into the platform with bounded latency, while preserving isolation from read-optimized serving paths.

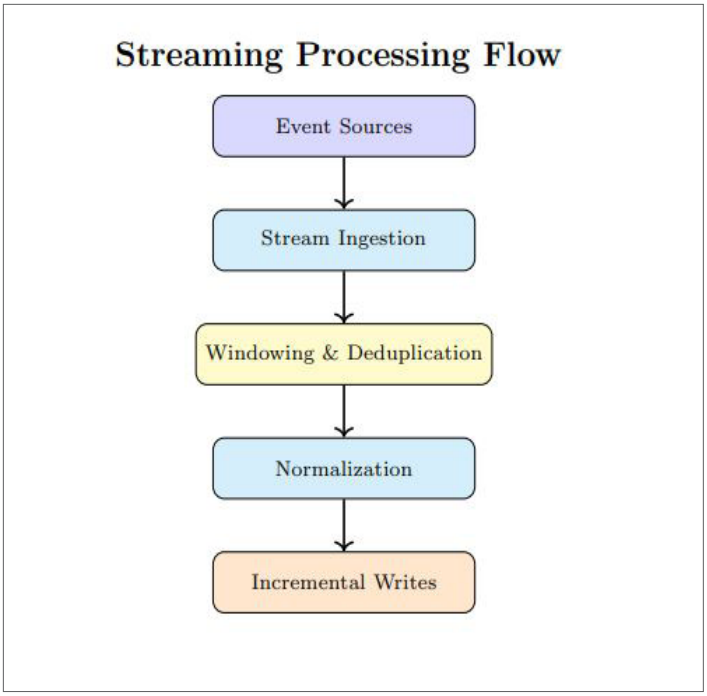
Streaming Data Characteristics

Operational event streams exhibit distinct characteristics compared to batch datasets:

- High write frequency with uneven temporal distribution
- Partial updates scoped to individual entities
- Occasional duplication and out-of-order delivery
- Strict freshness requirements for customer-facing use cases

The streaming architecture is designed to absorb these characteristics without introducing instability into downstream storage or query layers.

Freshness Guarantees Rather than enforcing strict transactional consistency, the system provides bounded staleness guarantees. Updates are typically visible within minutes, which is sufficient for pricing, inventory, and promotion scenarios while allowing efficient horizontal scaling.



Operational Behavior

Streaming pipelines operate continuously and scale elastically based on incoming event volume. Backpressure mechanisms ensure that sudden traffic spikes do not cascade into storage or serving layers.

Failure Isolation Transient failures in streaming components are handled through replay and checkpointing mechanisms. Since updates are incremental and idempotent, the system can safely resume processing without introducing duplicate or corrupted state.

Impact on Serving Latency Crucially, streaming ingestion does not execute synchronous writes on serving paths. By decoupling ingestion from query execution, the platform preserves stable P95 and P99 read latency even during sustained high-volume update periods.

Performance Characteristics

Tail Latency as a Design Objective

At retail scale, average latency metrics fail to capture real-world system behavior under peak traffic conditions. Customer experience and service-level objectives are governed primarily by tail latency, particularly the P95 and P99 percentiles observed during sustained load and traffic spikes.

The platform is explicitly engineered to bound tail latency rather than optimize median response times. Architectural decisions—including partitioning strategy, write isolation, caching policies, and query access paths—are evaluated based on their impact on worst-case response times.

Read Path Characteristics

Read operations are designed to remain singlepartition wherever possible. This minimizes crossnode coordination, bounds worst-case response time, and reduces variance in tail latency.

Under production workloads, read-heavy endpoints consistently maintain sub-100 ms P99 latency even during periods of elevated write throughput. This behavior is achieved by isolating ingestion activity from serving paths and precomputing readoptimized access structures.

Observed Tail Behavior Across catalog lookup, pricing resolution, and inventory checks, P95 latency remains stable while P99 latency is protected from degradation by isolating ingestion activity from serving layers.

Partitioning Strategy

Partitioning is a primary mechanism for controlling tail latency. Keys are chosen to ensure locality for reads while enabling parallelism for high-volume writes. Table 4.3 summarizes the partitioning strategy and its impact on P95 and P99 latency behavior.

System Stability Implications

By bounding tail latency during write spikes, the system avoids cascading timeouts and retry storms in upstream services. This behavior is critical during seasonal retail events, where read and write traffic increase simultaneously and tail latency directly impacts conversion rates.

Read Path Architecture

Single-Partition Read Design

Read paths are explicitly engineered to remain single-partition wherever possible. This constraint minimizes cross-node coordination, which is a primary source of tail latency amplification in distributed systems. By ensuring that the majority of read queries resolve within a single logical partition, the platform bounds worst-case response times even under peak load.

Key lookup operations—such as product detail retrieval, pricing

Citation: Perikala. K. (2024) Large-Scale Architecture for Retail Platforms Using Cloud-Native Big Data Systems. International Journal of Computer Science and Data Engineering, 1(3), 1–7 doi: https://dx.doi.org/10.55124/csdb.v1i3.268

Data Type	Partition Key Strategy	P95 / P99 Latency Impact
Product Metadata	`product_id`	Single-partition reads keep P99 below 80 ms at peak load.
Pricing Updates	`product_id + time`	Limits scan depth and stabilizes P95 during high churn.
Inventory State	`product_id + location`	Parallel writes without elevating read-path P99 latency.
Fulfillment Options	`product_id + region`	Prevents cross-partition fan-out during checkout flows.

Table 4.3: Partitioning strategies designed to bound P95 and P99 latency under sustained load.

resolution, and inventory availability checks—are structured around deterministic partition keys. This approach allows requests to be routed directly to the owning partition without requiring fan-out queries or scatter-gather patterns.

Avoidance of Fan-Out Queries

Fan-out queries are explicitly avoided in latencysensitive paths. Where aggregation is required, results are precomputed during ingestion or maintained as materialized views. This design ensures that read operations scale with query volume rather than with dataset size.

When fan-out behavior cannot be eliminated entirely, it is restricted to non-critical workflows such as offline analytics or background refresh tasks, isolating customer-facing paths from unpredictable latency spikes.

Impact on Tail Latency By constraining reads to predictable access patterns, the system maintains stable P95 and P99 latency distributions even as overall traffic volume increases. This predictability is critical during retail peak events where both read and write loads surge simultaneously.

Caching Strategy

Caching plays a central role in protecting tail latency while reducing backend load. The platform employs a layered caching strategy that spans service-level, query-level, and data-level caches, each with carefully tuned time-to-live (TTL) policies.

Service-level caches store fully assembled responses for frequently accessed endpoints such as product detail pages and category landing views. These caches provide the fastest response times and absorb the majority of repeated traffic.

Cache Invalidation and Freshness

Cache invalidation is driven by data change events emitted during ingestion. Updates to pricing, inventory, or promotions trigger targeted invalidation of affected cache entries rather than global cache flushes. This selective approach preserves cache efficiency while maintaining near real-time data accuracy.

TTL values are intentionally short for highly dynamic attributes and longer for stable metadata.

This balance prevents stale data exposure withoutsacrificing cache hit rates during sustained traffic spikes.

Resilience Under Load During ingestion spikes or partial backend degradation, cached responses continue to serve read traffic with minimal latency impact. This behavior prevents cascading failures and allows the system to degrade gracefully under extreme conditions, preserving core customer experiences.

Write Path Architecture

Separation of Write and Read Paths

The platform enforces strict separation between write-intensive ingestion paths and read-optimized serving paths. This architectural boundary ensures that high write throughput does not interfere with customer-facing query latency. All writes enter the system through controlled ingestion services that operate independently of read traffic.

Write operations are designed to be append-oriented wherever possible. Instead of performing in-place mutations that can introduce contention, updates are applied as versioned records or idempotent upserts. This approach simplifies concurrency control and reduces coordination overhead during peak update windows.

Streaming Ingestion Model

Operational signals such as pricing updates, inventory changes, and promotional events are ingested through streaming pipelines. These pipelines provide near-real-time propagation while maintaining ordering guarantees within partition boundaries. Streaming ingestion allows the platform to absorb sustained write rates without introducing batchinduced latency spikes. Backlogs are handled through elastic scaling rather than by throttling downstream read services.

Idempotency and Ordering Each write event carries a deterministic identifier that enables idempotent processing. Duplicate events caused by retries or upstream replays are safely ignored, ensuring correctness without sacrificing throughput. Ordering is preserved within partitions, which is sufficient for maintaining consistent entity state.

Backpressure and Load Regulation

Backpressure mechanisms are applied exclusively within the ingestion pipeline. When downstream storage or processing stages approach capacity limits, ingestion rates are adjusted dynamically without impacting read availability. This design prevents write amplification from cascading into customer facing services.

Rate limiting and buffering are applied at ingestion boundaries, allowing the system to smooth shortlived spikes while sustaining high average throughput during promotional events or inventory refresh cycles.

Write Amplification Control

Write amplification is minimized through batching and partition-aware routing. Events targeting the same partition are coalesced into compact write operations, reducing disk I/O and network overhead. This strategy is particularly important for inventory updates, where rapid changes can otherwise overwhelm storage subsystems.

Operational Stability By isolating write variability and enforcing bounded processing semantics, the platform maintains stable behavior even during extreme write surges. Operational teams can scale ingestion independently, replay streams safely, and perform reprocessing without risking degradation of read-path latency or availability.

This disciplined write-path design enables continuous data freshness while preserving the tail latency guarantees required for large-scale retail systems.

Citation: Perikala. K. (2024) Large-Scale Architecture for Retail Platforms Using Cloud-Native Big Data Systems. International Journal of Computer Science and Data Engineering, 1(3), 1–7 doi: https://dx.doi.org/10.55124/csdb.v1i3.268

Failure Handling and Recovery

Failure Isolation Model

Failure Isolation Model Failures are treated as localized events within clearly defined system boundaries. Ingestion, storage, and serving layers are designed to fail independently without cascading effects across layers. This isolation ensures that transient upstream failures do not impact customer-facing availability.

Ingestion failures are confined to processing stages and do not propagate to serving services. When malformed, delayed, or duplicate events are encountered, they are handled through validation and deadletter routing without affecting read paths.

Graceful Degradation

The system supports graceful degradation under partial failure scenarios. Non-critical data paths—such as enrichment or auxiliary attribute updates—may be temporarily paused while core product, pricing, and inventory reads continue to serve traffic within defined latency bounds.

This prioritization allows the platform to maintain core retail functionality even during infrastructure instability or upstream data quality issues.

Timeout and Retry Containment Retries are bounded and applied only within ingestion pipelines. Serving layers do not perform synchronous retries against storage systems, preventing retry storms during partial outages. Timeouts are calibrated to protect P95 and P99 latency guarantees under stress conditions.

Replay and Reprocessing Semantics

All ingestion pipelines are designed to support deterministic replay. Source events and batch snapshots can be reprocessed from known offsets without introducing duplicate state or violating consistency guarantees. This capability is critical for recovery from data corruption, schema evolution errors, or logic defects in processing stages.

Replay operations are isolated from live ingestion and serving traffic, allowing reprocessing to occur without service interruption.

Consistency During Recovery

During recovery or replay, the system preserves read consistency by ensuring that partially processed states are never exposed to serving layers. Versioned writes and atomic visibility controls guarantee that readers observe either the previous stable state or the fully recovered state, but never an intermediate one.

Operational Observability

Monitoring Objectives

Operating a large-scale retail data platform requires continuous visibility into both ingestion and serving paths. Observability is designed around detecting early signals of tail-latency regression, ingestion backlog growth, and data freshness violations before they impact customer-facing workloads.

The system distinguishes between control-plane metrics, which track pipeline health and capacity, and data-plane metrics, which reflect end-user experience.

Core Metrics

Key metrics are collected at each architectural layer:

- **Ingestion throughput**: records processed per second
- **Ingestion lag:** event time versus processing time
- **Write amplification**: logical updates versus physical writes
- **Read latency:** P95 and P99 across query classes
- **Cache efficiency**: hit ratio and eviction rate

These metrics are emitted as structured time-series signals and aggregated across regions to support comparative analysis and anomaly detection.

Tail Latency Focus

Unlike average latency metrics, P95 and P99 are treated as first-class signals. Alerting thresholds are defined directly on tail latency rather than mean response time, ensuring that performance degradation affecting a small fraction of users is detected early. This approach prevents silent regressions where averages remain stable while outliers increase under load.

Service-Level Objectives

Latency and Freshness SLOs

Service-level objectives (SLOs) formalize the performance and correctness expectations of the platform. These SLOs are derived from customer experience requirements rather than infrastructure capabilities.

SLO	Target	Measurement
Read latency	P95 $<$ 120 ms P99 $<$ 200 ms	Per-query class
Ingestion freshness	$<$ 10 min lag	Event-time watermark
Availability	99.95% monthly	Successful responses
Data correctness	$<$ 0.01% errors	Validation checks

Error Budgets

Error budgets translate SLOs into actionable operational limits. When tail latency or freshness violations consume a significant portion of the error budget, feature rollouts are paused and capacity remediation is prioritized. This mechanism enforces a disciplined balance between innovation velocity and system reliability.

Operational Governance

Operational governance policies define escalation paths, rollback procedures, and capacity planning cadences. These policies ensure that changes to ingestion pipelines, storage schemas, or service logic are evaluated through the lens of tail-latency impact and data integrity. As a result, the platform evolves incrementally without destabilizing mission-critical retail workflows.

Design Trade-offs

Consistency vs. Freshness

One of the central trade-offs in the platform design is the balance between strong consistency and near real-time freshness. Operational retail signals such as pricing and inventory change frequently and must be reflected quickly, while foundational data such as product taxonomy prioritizes correctness and auditability.

Citation: Perikala. K. (2024) Large-Scale Architecture for Retail Platforms Using Cloud-Native Big Data Systems. International Journal of Computer Science and Data Engineering, 1(3), 1–7 doi: https://dx.doi.org/10.55124/csdb.v1i3.268

The architecture intentionally avoids global transactional guarantees across all data types. Instead, it applies bounded consistency windows aligned with business tolerance. This approach simplifies scalability while ensuring that user-visible inconsistencies remain rare and short-lived.

Write Amplification

Partitioned storage and denormalized read models improve query latency but introduce write amplification during ingestion. Derived fields, secondary indexes, and cached projections must be updated in tandem with source records.

To mitigate this cost, the platform favors batched writes, idempotent update semantics, and selective materialization. Write-heavy components are isolated from serving paths so that amplification does not propagate to read latency.

Operational Complexity

The separation of ingestion, storage, optimization, and serving layers increases the number of deployable components. While this modularity improves resilience and scalability, it requires disciplined operational practices, automated validation, and strong observability to manage effectively.

Future Evolution

Adaptive Optimization

Future iterations of the platform can incorporate adaptive optimization techniques that dynamically adjust indexing, caching, and partitioning strategies based on observed access patterns. By continuously learning from query distributions and tail-latency behavior, the system can proactively optimize for emerging workloads.

AI-Assisted Operations

Operational telemetry collected across ingestion and serving layers provides a rich signal for automation. Predictive models can forecast ingestion backlog, capacity exhaustion, or tail-latency regression before thresholds are breached. Such models enable preemptive scaling actions and reduce manual intervention.

Generative Interfaces

While the current system focuses on structured and semi-structured access patterns, the same unified data foundation enables higher-level generative interfaces. Natural language exploration, contextual recommendations, and adaptive user journeys can be layered on top without altering core ingestion and storage pipelines.

Cross-Domain Expansion

The architectural principles described in this work generalize beyond retail. Domains such as logistics, healthcare, and financial services face similar challenges in integrating heterogeneous, high-velocity data while maintaining low-latency access. The same separation of concerns and tail-latency governance can be applied with minimal adaptation.

Summary Perspective The platform's evolution strategy emphasizes incremental enhancement rather than architectural replacement. By preserving core invariants—unidirectional flow, isolation of concerns, and tail-latency governance—the system can absorb new technologies and workloads without destabilizing production operations.

Operational Insights

Tail Latency as a First-Class Metric

A key operational lesson from production deployment is that average latency metrics are insufficient for reasoning about customer experience at scale. Instead, P95 and P99 latencies must be treated as first-class indicators of system health.

The platform continuously monitors tail latency across ingestion, storage, and serving layers. Capacity planning decisions are driven primarily by P99 behavior under peak load rather than mean utilization. This approach ensures that customer-facing services remain responsive even during extreme traffic spikes.

Isolation Reduces Incident Scope

The strict separation between write-heavy ingestion paths and read-optimized serving layers significantly reduces blast radius during failures. In practice, most operational incidents are confined to a single layer and do not propagate end-to-end. Examples include upstream data corruption, delayed batch runs, or transient streaming backlog. In each case, read availability is preserved while recovery procedures execute independently.

Predictable Recovery Patterns

Because all state mutation is centralized in ingestion stages, recovery follows predictable patterns. Replays, backfills, and reprocessing operations do not require coordinated downtime across consumer services. This property materially reduces mean time to recovery during high-severity incidents.

Empirical Observations

Load Seasonality Effects

Retail traffic exhibits strong temporal locality driven by promotions, holidays, and regional demand. The architecture accommodates these patterns by decoupling scaling decisions across layers. Ingestion throughput scales in response to data velocity, while serving capacity tracks user traffic independently.

This decoupling prevents ingestion surges from amplifying read-path tail latencies, a common failure mode in tightly coupled architectures.

Schema Stability Over Time

Long-lived production operation highlights the importance of schema stability. Backward-compatible schema evolution, additive field introduction, and strict deprecation policies reduce operational risk.

Schema versioning at ingestion boundaries further isolates downstream consumers from upstream change.

Cost Predictability

The platform exhibits stable cost characteristics due to bounded fan-out, controlled write amplification, and explicit capacity targets. Unlike reactive scaling approaches, cost growth correlates linearly with business expansion rather than traffic volatility.

Key Takeaway Operational success is driven less by individual technology choices and more by discipline in enforcing architectural boundaries, observability, and latency governance across the system lifecycle.

Citation: Perikala. K. (2024) Large-Scale Architecture for Retail Platforms Using Cloud-Native Big Data Systems. International Journal of Computer Science and Data Engineering, 1(3), 1–7 doi: https://dx.doi.org/10.55124/csdb.v1i3.268

Reference Implementation and Technology Stack

The architectural layers described in this work are realized using mature, cloud-native technologies that have demonstrated predictable behavior at scale. High-velocity operational signals are ingested through distributed streaming platforms such as Apache Kafka or managed cloud-native equivalents, providing durability, ordering guarantees, and replay semantics required for recovery and reprocessing.

Batch-oriented ingestion of foundational datasets is implemented using distributed processing engines such as Apache Spark, enabling deterministic snapshot construction and large-scale data normalization. Persistent state is maintained in horizontally scalable NoSQL systems, including wide-column stores such as Apache Cassandra or cloud-native analogs, selected for their stable P95/P99 latency characteristics under sustained write pressure. Stateless microservices expose read-optimized access paths and encapsulate query logic, while in-memory caching layers and precomputed views reduce traversal depth and mitigate tail-latency amplification.

Conclusion

This paper presented a cloud-native, large-scale data platform architecture designed to meet the stringent latency, throughput, and reliability requirements of modern retail systems. By enforcing unidirectional data flow, isolating ingestion from serving paths, and explicitly governing tail latency behavior, the platform achieves predictable performance under extreme operational load.

The architecture demonstrates that scalable retail platforms benefit most from discipline in system boundaries rather than reliance on any single technology. Distributed storage, streaming pipelines, and microservice-based serving layers must be composed in a manner that prioritizes fault containment, evolvability, and observability.

From an operational perspective, the emphasis on P95 and P99 latency metrics, idempotent ingestion, and deterministic recovery patterns enables the platform to sustain continuous evolution without destabilizing customer-facing workloads.

Future Outlook As retail platforms increasingly integrate generative AI, real-time personalization, and multimodal interactions, the architectural principles described here provide a stable foundation.

Systems that preserve isolation, predictability, and bounded complexity will be best positioned to support the next generation of intelligent retail experiences.

References

1. Kleppmann, M. Designing Data-Intensive Applications, 2nd Edition. O 'Reilly Media,2022.
2. Dean,J. Achieving Rapid Response Times in Large Online Services. Google Research, 2020.
3. Abadi, D., et al. Cloud-Native Databases: Principles and Trade offs. IEEE Data Engineering Bulletin, 2021.
4. Kreps, J. Streaming Systems at Scale: Architecture and Patterns. O'Reilly Media, 2021.
5. Google Cloud Architecture Center. Real-Time Analytics and Data Platform Design Patterns. Google Cloud, 2023.
6. Armbrust, M., et al. Lake house: A New Generation of Open Plat forms that Unify Data Warehousing and Advanced Analytics. CIDR, 2021.
7. Netflix Technology Blog. Resilience Engineering in
8. Large-Scale Distributed Systems. 2021.
9. 8. Microsoft Azure Architecture Center. Designing
10. Highly Scalable Microservices Architectures. 2022.
11. 9. Amazon Builders' Library. Controlling Tail Latency.
12. Amazon Web Services, 2020.
13. 10. Chen, J., et al. Tail-Tolerant Distributed Systems.
14. ACM Queue, 2023.
15. 11. Google Research. Production Considerations for
16. Machine Learning Systems. 2022.
17. 12. Meta Engineering. Scaling Data Infrastructure for
18. Real-Time Systems. Engineering Blog, 2023.

Citation: Perikala. K. (2024) Large-Scale Architecture for Retail Platforms Using Cloud-Native Big Data Systems. International Journal of Computer Science and Data Engineering, 1(3), 1–7 doi: https://dx.doi.org/10.55124/csdb.v1i3.268