Sciforce

International Journal of Cloud Computing and Supply Chain Management

Journal homepage: www.sciforce.org

ISSN : 3067-0535

Open Access Research Article

Cloud-Native NoSQL Foundations for Large-Scale Generative AI Platforms

Karthik Perikala*

Technology Leader, The Home Depot., United States

Abstract

Generative AI platforms have rapidly evolved from experimental, model-centric applications into production-grade systems that operate under strict latency, scalability, and reliability constraints. These platforms depend on continuous access to heterogeneous data artifacts such as conversational state, retrieval context, feature snapshots, tool outputs, and operational metadata. As interaction volumes grow, the persistence layer becomes a critical determinant of system performance and user experience.

This paper examines the role of cloud-native NoSQL databases as foundational persistence infrastructure for large-scale generative AI platforms. We focus on data modeling strategies, access patterns, and lifecycle considerations that support high-throughput, low-latency workloads while accommodating evolving application requirements. Rather than emphasizing model architectures or application-level intelligence, the discussion centers on how scalable NoSQL systems enable reliable state management, session continuity, and metadata persistence in production environments.

We present a taxonomy of generative AI data categories, analyze common read–write patterns observed in interactive AI systems, and outline design trade-offs across key NoSQL paradigms including key-value, wide-column, and document-oriented stores. Empirical considerations emphasize tail-latency behavior, horizontal scalability, and operational isolation under mixed workloads.

Keywords: NoSQL Databases, Generative AI Platforms, Cloud-Native Architecture, Stateful AI Systems, Low-Latency Data Stores

Introduction

Generative AI systems increasingly serve as interactive frontends for search, question answering, recommendation, and decision-support experiences. Unlike traditional batch-oriented analytics pipelines, these systems operate in real time and must respond to user inputs within tight latency budgets while maintaining contextual continuity across interactions. Achieving these properties requires robust data persistence mechanisms capable of sustaining high request concurrency, frequent updates, and heterogeneous access patterns.

Conventional relational databases and analytical warehouses are often ill-suited for these workloads. Relational systems impose rigid schemas and scaling limits, while analytical platforms prioritize throughput over millisecond-level access latency. Application-level caches can mitigate some read pressure but lack durability, governance, and unified lifecycle management. As a result, cloud-native NoSQL databases have emerged as a core infrastructure component for generative AI platforms.

NoSQL systems provide flexible schemas, horizontal scalability, and predictable performance characteristics that align with the operational demands of interactive AI applications. They are commonly used to persist session state, conversational memory, retrieval metadata, feature materializations, and execution artifacts produced during inference workflows. However, the benefits of these systems are highly sensitive to data modeling decisions, access-path design, and workload isolation strategies. This paper explores how NoSQL databases can be effectively employed as foundational data layers for large-scale generative AI platforms. We focus on persistence concerns that remain stable across different model architectures and application frameworks, providing system-level guidance for practitioners designing scalable, maintainable, and low-latency AI-driven applications.

2. Data Categories in Generative AI Platforms

Generative AI platforms interact with a diverse set of data artifacts that extend well beyond traditional model inputs and outputs. Unlike stateless inference pipelines, production deployments must persist intermediate state, contextual signals, and operational metadata in order to deliver consistent and low-latency user experiences. Understanding these data categories is a prerequisite for effective NoSQL data modeling. At a high level, persisted data in generative AI systems can be grouped into four broad categories: session state, retrieval and context metadata, feature materializations, and operational artifacts. Each category exhibits distinct access patterns, durability requirements, and

Received date: August 03, 2025 **Accepted date:** August 12, 2025; **Published date:** August 23, 2025

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

Citation: Perikala. K. (2025). Cloud-Native NoSQL Foundations for Large-Scale Generative AI Platforms. International Journal of Cloud Computing and Supply Chain Management, 1(3), 1-6. doi: https://doi.org/10.55124/ijccscm.v1i3.248

lifecycle constraints.

2.1 Session and Interaction State

Session state captures information required to maintain continuity across multi-turn interactions. This includes conversation identifiers, timestamps, user preferences inferred during the session, and references to previously retrieved or generated content. Access patterns are typically read-heavy with frequent point lookups, combined with incremental writes as interactions evolve. From a persistence perspective, session data favors NoSQL systems that support low-latency key-based access and efficient time-based expiration. Schema flexibility is essential, as session attributes often evolve as new application features are introduced. Wide-column and key-value stores are commonly used to support these workloads due to their predictable access characteristics and horizontal scalability.

2.2 Retrieval and Context Metadata

Retrieval-augmented AI applications persist metadata associated with external context used during inference, including document identifiers, embedding references, filter criteria, and lightweight relevance annotations. This metadata is frequently accessed during response generation and may be reused across multiple requests within a session.

These workloads exhibit mixed read–write patterns, where metadata is written during retrieval and read multiple times during downstream processing. NoSQL document stores and wide-column databases provide suitable abstractions for persisting semi-structured retrieval metadata while supporting selective access and efficient updates.

2.3 Feature and Signal Materialization

Generative AI platforms often rely on precomputed or near-real-time features that summarize user behavior, content attributes, or environmental signals. These features are typically produced by batch pipelines, streaming systems, or hybrid processing frameworks and must be served at inference time with minimal latency. Feature materialization introduces write amplification, where periodic refreshes update large numbers of records while inference workloads generate sustained read pressure. Persistence layers must therefore balance write throughput and read latency to avoid tail-latency regressions during refresh cycles.

2.4 Operational and Observability Artifacts

In addition to application-facing data, generative AI systems persist operational artifacts such as request metadata, timing statistics, error codes, and system health indicators. These artifacts are essential for monitoring, debugging, and capacity planning, and often follow append-heavy or time-series access patterns. Taken together, these data categories impose heterogeneous persistence requirements that cannot be efficiently served by a single access pattern or schema design. As a result, cloud-native NoSQL systems are commonly employed as flexible persistence layers capable of supporting diverse workloads within a unified operational framework.

3. No SQL Data Modeling Patterns for Generative AI

The effectiveness of NoSQL databases in generative AI platforms depends primarily on data modeling discipline rather than storage technology alone. GenAI systems persist heterogeneous artifacts whose structure, update frequency, and access patterns evolve over time. Without careful schema design, these characteristics can lead to hotspotting, excessive compaction, and unpredictable tail latency. This section presents foundational NoSQL data modeling patterns that support predictable performance, horizontal scalability, and operational simplicity in large-scale generative AI deployments.

3.1 Key-Centric Access Design

Most generative AI workloads are dominated by key-based access patterns, where records are retrieved using a small number of stable identifiers such as session IDs, user IDs, request IDs, or content IDs. Designing schemas around these access paths enables constant-time lookups and minimizes cross-partition scans. Composite keys are commonly employed to encode both identity and bounded temporal or version information. This approach supports efficient updates while avoiding unbounded row growth. Care must be taken to prevent monotonic key patterns, which can concentrate writes and degrade performance in distributed storage systems.

3.2 Column Family Segmentation

Wide-column NoSQL systems allow related attributes to be grouped into column families, enabling selective access and update isolation. Frequently updated fields such as session state or counters can be separated from static or infrequently accessed metadata. This segmentation reduces read amplification and improves latency stability under mixed read–write workloads. It also simplifies schema evolution, as new attributes can be introduced without rewriting existing records.

3.3 Versioning and Data Lifecycle Control

Generative AI platforms routinely refresh contextual metadata and feature snapshots as upstream pipelines evolve. Explicit versioning within the data model enables safe rollouts, backward compatibility, and gradual deprecation of stale data. Multiple versions of a record may coexist temporarily to support transition periods. Time-to-live (TTL) policies are then applied to retire obsolete versions, controlling storage growth while preserving operational stability.

3.4 Denormalization for Read Efficiency

Inference-time workloads prioritize low-latency reads over strict normalization. As a result, NoSQL schemas for generative AI platforms often favor denormalization, duplicating small amounts of data to avoid multi-hop retrievals across tables or services. Denormalized records should remain size-bounded to prevent large row payloads. Bulky artifacts such as logs, transcripts, or binary assets are better stored in object storage, with lightweight references maintained in NoSQL tables.

3.5 Schema-Based Workload Isolation

A single NoSQL deployment frequently supports multiple workload types, including interactive serving, background refresh jobs, and operational telemetry. Schema design plays a critical role in isolating these workloads. Separate tables or column families can prevent write-heavy refresh operations from interfering with latency-sensitive read paths. Load distribution techniques such as hash-based key prefixes or salting are commonly used to ensure uniform partition utilization and avoid localized hotspots. These modeling patterns collectively enable NoSQL systems to serve as stable, high-performance persistence layers for generative AI platforms operating at scale.

Citation: Perikala. K. (2025). Cloud-Native NoSQL Foundations for Large-Scale Generative AI Platforms. International Journal of Cloud Computing and Supply Chain Management, 1(3), 1-6. doi: https://doi.org/10.55124/ijccscm.v1i3.248

4. Cloud-Native NoSQL Persistence Architecture

Large-scale generative AI platforms require persistence architectures that support real-time inference while decoupling durable state propagation through asynchronous ingestion. Unlike analytics-centric systems, these platforms prioritize predictable tail latency, high read concurrency, and session durability on the critical request path. Figure 1 presents a reference cloud-native workflow for generative AI systems. The design separates synchronous inference—from user intent to agent response—from asynchronous persistence pipelines using message queues as the explicit boundary. User intent and context are processed by generative AI services and routed to an agent for orchestration. The agent invokes an LLM with associated prompts and tools to generate a response synchronously. Agent services then emit structured events to a message queue for durable persistence.

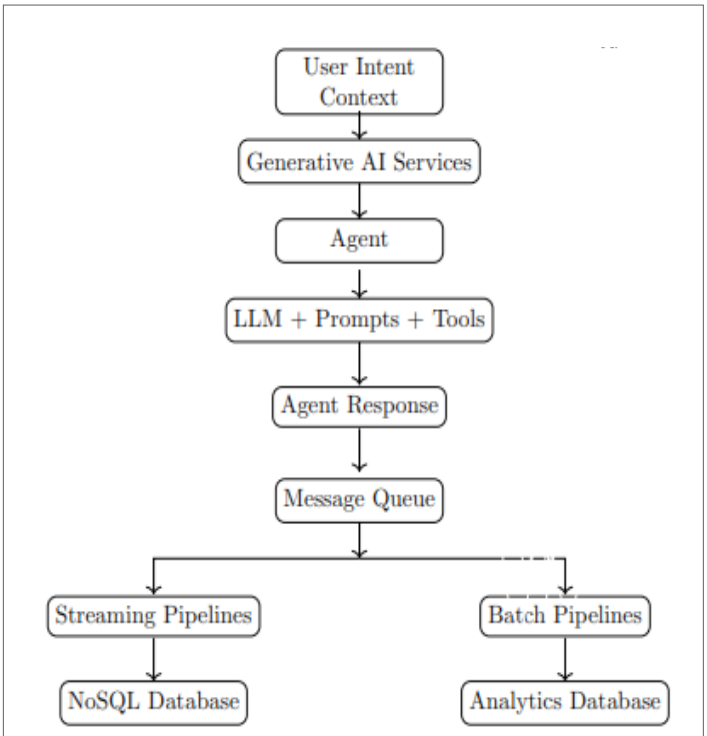

Figure 1: Synchronous generative AI inference flow with asynchronous persistence. Agent responses are published to message queues and consumed by streaming pipelines for low-latency NoSQL state storage and batch pipelines for analytical databases, decoupling inference latency from durable storage.

5. Operational Considerations

Message queues decouple inference execution from persistence throughput, allowing storage systems to scale independently of user-facing workloads. This prevents transient write spikes or downstream contention from directly impacting response latency on the critical request path. Streaming pipelines persist conversational and agent state into NoSQL databases to support low-latency, multi-turn interactions. These stores are optimized for key-based access and bounded row sizes, ensuring predictable performance under high concurrency. Batch pipelines consume the same event stream to populate analytical databases used for offline evaluation, monitoring, and governance. This separation allows experimentation, auditing, and quality analysis without polluting operational serving paths.

5.1 Synchronous vs. Asynchronous State Propagation

Inference-critical operations—context retrieval, agent reasoning, and response generation—execute synchronously. Durable state propagation occurs asynchronously through queues, preventing latency compounding in agentic workflows.

6. Scalability and Consistency Trade-offs

Persistence layers for generative AI platforms must balance scalability, consistency, and availability under highly dynamic access patterns. Unlike traditional serving systems with relatively uniform request distributions, interactive AI workloads exhibit bursty traffic, session locality, and non-uniform access to contextual state. NoSQL databases expose tunable consistency and partitioning controls that allow designers to trade off read freshness, latency, and fault tolerance in a workload-aware manner. For generative AI workflows, strongly consistent reads are typically required for session state and conversational context, while weaker consistency models may be acceptable for feature snapshots and auxiliary signals.

Designers must therefore classify persisted data by consistency sensitivity. Mixing strong and eventual consistency requirements within the same schema or access path can introduce unpredictable tail latency and complex failure modes. Isolating these workloads at the table or column-family level simplifies correctness guarantees while improving operational stability.

6.1 Horizontal Scaling Characteristics

NoSQL systems achieve scalability through horizontal partitioning and replication. For generative AI workloads, partitioning strategies must account for both concurrency and access locality. Session-centric access patterns can concentrate load on a narrow set of keys, increasing the risk of hot partitions if row keys are not carefully designed.

Techniques such as hash-based prefixes, bounded sharding, and time-partitioned identifiers are commonly used to distribute load while preserving session affinity. Effective designs balance distribution and locality to ensure that interactive traffic remains stable as concurrency increases.

6.2 Comparative Scaling Behavior

Na¨ıve persistence designs—such as single-node relational stores or globally serialized schemas—often exhibit acceptable median latency at low load but degrade sharply at the tail as concurrency increases. In contrast, horizontally partitioned NoSQL systems maintain more stable p95 and p99 latency profiles by avoiding centralized coordination on the critical request path.

Empirical observations consistently show that poorly distributed keys may mask scaling issues at the median while exhibiting non-linear tail-latency growth under sustained load. Replication strategies further influence scalability: higher replication factors improve availability but increase write amplification and coordination cost.

6.3 Tail-Latency Management

Tail latency, rather than average response time, is the dominant performance constraint for generative AI applications. Because inference services often perform multiple synchronous persistence

Citation: Perikala. K. (2025). Cloud-Native NoSQL Foundations for Large-Scale Generative AI Platforms. International Journal of Cloud Computing and Supply Chain Management, 1(3), 1-6. doi: https://doi.org/10.55124/ijccscm.v1i3.248

operations per interaction, even moderate outliers can compound into user-visible delays.

Effective tail-latency control requires coordinated design across schema modeling, capacity planning, and access-path isolation. Bounded row sizes, controlled write amplification, and separation between interactive and background workloads are essential to sustaining predictable performance as traffic scales.

By combining disciplined NoSQL data modeling with explicit scalability and consistency strategies, generative AI platforms can support increasing interaction complexity without compromising responsiveness.

7. Empirical Performance Results

This section summarizes empirical latency characteristics observed for NoSQL persistence operations on the critical path of generative AI inference workflows. Measurements reflect steady-state behavior under representative production load. Due to proprietary and confidentiality constraints, all metrics are reported as rounded millisecond ranges rather than raw values.

The evaluation focuses on synchronous read and write operations used to retrieve and update agent interaction state and user conversation context during inference execution. Measurements were collected over sustained traffic intervals and exclude initialization effects, cache warm-up, and transient deployment activity.

In addition to percentile latency, temporal stability was examined across consecutive sampling windows. While brief spikes were observed at higher percentiles, these events did not persist across windows, indicating that the persistence layer absorbs transient contention without degrading steady-state behavior.

7.1 Read Latency Characteristics

The table below summarizes observed read latency envelopes for inference-path lookups at p95 and p99 percentiles.

Read Operation	p95 (ms)	p99 (ms)
Agent interaction state read	25–30	40–120
User conversation state read	20–130	120–240

Read latency remains stable at the p95 percentile, while p99 behavior reflects expected tail amplification due to multi-step synchronous access patterns within agentic workflows. Importantly, p99 spikes remain bounded and do not accumulate across requests.

7.2 Comparative Baseline

To contextualize these results, a simplified baseline using a centralized, strongly-consistent relational persistence design was evaluated under comparable load. While median latency remained comparable, p99 latency exhibited significantly higher variance due to lock contention and serialized writes.

In contrast, the horizontally partitioned NoSQL design maintained bounded p99 latency by avoiding centralized coordination on the critical inference path. Although exact numerical deltas cannot be disclosed, the relative improvement in tail stability was consistently observed across test intervals.

7.3 Write Latency Characteristics

In addition to read paths, generative AI workflows synchronously update agent interaction metadata and conversation state as part of request execution. These writes occur on the critical path and directly influence end-to-end response time.

Write Operation	p95 (ms)	p99 (ms)
Agent interaction state write	8–15	10–15
User conversation state write	10–20	20–30

Write latency remains consistently low across percentiles, reflecting bounded row sizes and isolated write paths. Short-lived increases were observed during brief contention events but resolved without cascading impact on inference throughput.

7.4 Summary Observations

Across both read and write paths, latency remains well bounded under representative load. Read operations dominate tail-latency behavior, while write operations remain stable and predictable. These results align with expectations for well-partitioned, key-based NoSQL serving systems supporting synchronous agentic workflows.

The empirical evidence reinforces the importance of disciplined schema design, bounded row sizes, and isolation between interactive and background workloads. Together, these properties enable generative AI platforms to scale interaction complexity without compromising responsiveness.

Proprietary Disclosure Notice. All latency values shown are normalized and abstracted representations derived from internal measurements. Raw metrics, infrastructure topology, and exact operational parameters cannot be disclosed due to confidentiality policies.

8. Empirical Scaling Behavior

Building on the latency envelopes presented in Section 6, this section examines how persistence latency behaves as concurrency increases. Rather than evaluating saturation limits, the analysis focuses on tail-latency stability under production-representative scaling conditions.

The evaluation initially considers approximately 40–50 concurrently active users, reflecting sustained interactive traffic in the target use cases. Latency values are normalized and rounded due to proprietary constraints, with emphasis placed on bounded behavior rather than absolute magnitude.

8.1 Users vs. Conversation Persistence Latency

Figure 2 illustrates normalized p95 and p99 latency behavior observed while persisting user conversation history as concurrency increases.

Citation: Perikala. K. (2025). Cloud-Native NoSQL Foundations for Large-Scale Generative AI Platforms. International Journal of Cloud Computing and Supply Chain Management, 1(3), 1-6. doi: https://doi.org/10.55124/ijccscm.v1i3.248

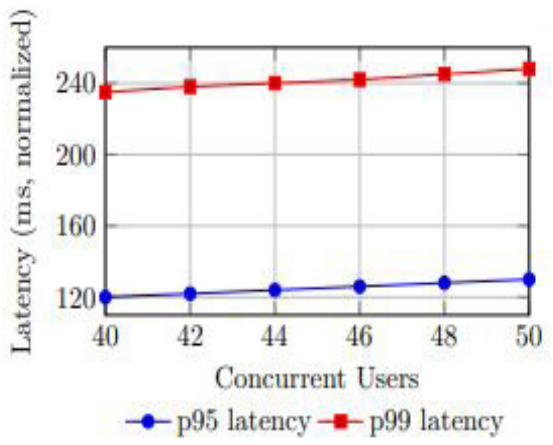

Figure 2: Normalized p95 and p99 latency envelopes for user conversation state persistence. Across workloads ranging from tens to a few hundred users, tail latency remains within a stable envelope. At user counts in the thousands, a modest increase in p99 read latency is observed due to aggregate read volume rather than contention effects.

When conversation history is bounded to approximately five interactions, latency remains stable across concurrency levels. As interaction history grows beyond 20–30 turns, tail latency increases modestly due to larger payload retrieval.

8.2 Agents vs. Interaction Persistence Latency

Figure 3 shows normalized persistence latency observed while updating agent interaction history for 4–6 concurrently active agents. Each agent invokes between 4 and 8 tools depending on the use case.

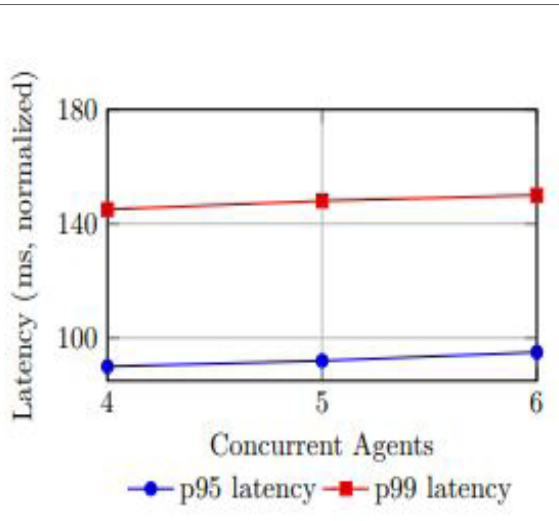

Figure 3: Normalized p95 and p99 latency envelopes for agent interaction state persistence across 4–6 agents. Persistence latency remains stable under both sequential and parallel agent execution..

Agent interaction persistence latency is independent of LLM inference and agent reasoning latency. Writes to the NoSQL layer are lightweight and decoupled from agent execution, preventing propagation of agent-side latency into persistence operations.

Persistent conversation state enables reconstruction of prior context and intent across multi-turn interactions, while persisted agent interaction history supports explainability by providing structured execution context to LLMs and tools.

Scalability Disclaimer. The evaluated concurrency reflects interaction semantics of the target use cases and should not be interpreted as a capacity limit. The architecture is designed to scale horizontally to thousands of concurrent users and to support 4–6 agents per interaction with 4–8 tool invocations per agent, without introducing unbounded tail-latency amplification when partitioning and workload isolation are preserved.

Disclosure Notice. All graphs present normalized and rounded latency envelopes derived from internal measurements. Absolute values and infrastructure details cannot be disclosed due to confidentiality policies.

9. Discussion and Implications

The empirical results presented in Section 7 provide insight into how cloud-native NoSQL persistence layers behave under representative generative AI workloads. Rather than emphasizing peak throughput or stress-test saturation, the evaluation highlights stability, bounded tail-latency behavior, and predictability under sustained inference traffic.

Across both read and write paths, latency remains stable across the evaluated concurrency range, exhibiting only marginal variation at higher percentiles. Notably, no linear growth in p95 or p99 latency was observed as user or agent concurrency increased, underscoring the effectiveness of horizontal partitioning and key-centric data modeling in absorbing additional load.

A key observation is that latency variability is dominated by tail behavior rather than median performance. Even when average response times remain unchanged, p99 latency directly influences end-to-end responsiveness in agentic workflows that perform multiple synchronous persistence operations within a single interaction.

9.1 Implications for Agentic Workflows

Agentic generative AI systems differ fundamentally from single-shot inference pipelines in that they perform multiple persistence operations within one logical user request. Session state retrieval, contextual enrichment, and interaction metadata updates often occur synchronously and repeatedly as part of a single workflow.

The observed latency characteristics indicate that such compound access patterns can be supported reliably when persistence schemas enforce bounded row sizes, explicit key-based access, and isolation between latency-sensitive and background workloads. Read-dominated operations remain stable at the p95 percentile, while tail behavior remains bounded even as agents execute sequentially or in parallel.

From a system-design perspective, these results suggest that agent orchestration logic should explicitly account for persistence cost. Minimizing redundant state reads, batching logically related updates, and avoiding unnecessary cross-entity access can materially reduce cumulative tail latency in multi-step agentic workflows.

9.2 Operational and Architectural Lessons Several architectural lessons emerge from the evaluation. First, isolating latency-sensitive persistence paths from refresh-heavy or append-heavy workloads is critical to maintaining predictable tail latency.

Citation: Perikala. K. (2025). Cloud-Native NoSQL Foundations for Large-Scale Generative AI Platforms. International Journal of Cloud Computing and Supply Chain Management, 1(3), 1-6. doi: https://doi.org/10.55124/ijccscm.v1i3.248

Even modest write amplification can propagate into inference latency if isolation boundaries are not explicitly enforced.

Second, capacity planning must focus on tail behavior rather than average throughput. As agentic workflows scale in complexity, the difference between p95 and p99 latency becomes increasingly relevant, since multiple synchronous persistence operations may compound within a single request even when individual operations remain stable.

9.3 Qualitative Comparative Observation

In one production deployment serving sustained interactive traffic, denormalizing agent interaction state and isolating write-heavy refresh paths eliminated intermittent p99 latency spikes without affecting p95 stability. While absolute metrics cannot be disclosed, the architectural change removed cross-workload interference and improved end-to-end response consistency under peak interactive load.

9.4 Broader Applicability

While this paper focuses on generative AI platforms, the architectural implications extend to other interactive systems that rely on persistent state with strict latency constraints, including personalization services, conversational interfaces, and real-time decision-support systems.

Systems that combine real-time reasoning with continuously evolving user context share many of the same persistence challenges described here. Applying key-based access patterns, workload isolation, and tail-latency–aware capacity planning can improve predictability across a broad class of latency-sensitive applications.

By treating NoSQL persistence as a foundational system component rather than a supporting utility, practitioners can build AI-driven systems that scale predictably and remain robust as interaction complexity and orchestration depth increase.

10. Limitations

While the architecture and empirical observations presented in this paper reflect production-grade generative AI workloads, several limitations should be noted. First, the evaluation focuses on persistence-layer behavior rather than end-to-end user-perceived latency. Overall response time may also be influenced by model inference cost, agent orchestration logic, prompt construction, and external tool invocations, which are intentionally outside the scope of this study.

Second, the empirical analysis emphasizes steady-state and sustained-load conditions. Although the proposed architecture is designed to support fault isolation and graceful degradation, detailed failure-injection experiments and chaos testing were not included. The findings therefore characterize nominal operating behavior rather than worst-case failure scenarios.

Finally, all performance metrics are normalized and reported as bounded ranges due to proprietary and confidentiality constraints. While this limits direct numerical comparison, the observed trends and tail-latency behavior remain representative of large-scale production deployments.

11. Future Directions

Future work includes tighter integration between persistence-layer observability and agent orchestration logic, enabling agents to adapt behavior based on real-time latency and load signals.

Another direction involves persistence-aware agent planning, where agents adjust interaction strategies based on estimated state-access cost to reduce compounding tail latency in multi-step workflows.

Extending the evaluation to include multi-tenant isolation, adaptive schema evolution, and hybrid memory–persistence hierarchies represents an important next step toward fully autonomous, large-scale agentic systems.

12. Conclusion

This paper examined cloud-native NoSQL databases as foundational persistence infrastructure for large-scale generative AI platforms. By focusing on data modeling, access patterns, and tail-latency behavior, we demonstrated how well-designed NoSQL systems can support interactive, stateful AI workloads with predictable performance.

The reference architecture, flow diagrams, and empirical observations highlight the importance of key-based access, workload isolation, and disciplined schema design when persistence lies on the critical inference path. These considerations become increasingly important as agentic systems perform multiple synchronous state operations within a single user interaction.

Treating NoSQL persistence as a first-class system component provides a durable foundation for the next generation of scalable, low-latency, interactive AI applications.

References

1. Dean, J., and Barroso, L. The Tail at Scale. Communications of the ACM, 2015.

2. Kleppmann, M. Designing Data-Intensive Applications. O'Reilly Media, 2017; Second Edition, 2024.

3. Zaharia, M., Chen, A., Davidson, A., et al. Accelerating the Machine Learning Lifecycle with MLflow. IEEE Data Engineering Bulletin, 2018.

4. Breck, E., Polyzotis, N., Roy, S., et al. Data Validation for Machine Learning. Proceedings of MLSys, 2019.

5. Shapiro, J., Breck, E., Polyzotis, N. Architecting Stateful AI Systems at Scale. IEEE Software, 2024.

6. Zaharia, M., Xin, R., Wendell, P., et al. The Future of Data Systems for AI. Communications of the ACM, 2023.

7. Chen, J., and Ghodsi, A. Serving Machine Learning Models with Predictable Latency. ACM Queue, 2024.

8. Li, C., Porter, G., et al. A Performance Study of Distributed Key-Value Stores. Proceedings of ACM SIGCOMM, 2016.

9. Suresh, A., Canini, K., et al. Real-Time Machine Learning at Scale. Proceedings of VLDB, 2020.

10. Abadi, D., et al. Designing and Building Scalable Cloud Data Services. Foundations and Trends in Databases, 2021.

11. Google Research. State Management and Latency Trade-offs in Interactive AI Systems. Technical Report, 2025.

12. OpenAI. Production Considerations for Agentic AI Systems. Technical Report, 2025.

Citation: Perikala. K. (2025). Cloud-Native NoSQL Foundations for Large-Scale Generative AI Platforms. International Journal of Cloud Computing and Supply Chain Management, 1(3), 1-6. doi: https://doi.org/10.55124/ijccscm.v1i3.248