NoSQL Databases: Models, Architectures, and Trends for Scalable Data Management

Elias Dritsas, Maria Trigka, and Phivos Mylonas
University of West Attica, Egaleo 12243, Greece
{idritsas,mtrigka,mylonasf}@uniwa.gr

***Abstract*—The increasing heterogeneity, volume, and velocity of modern data have exposed fundamental limitations in traditional relational database management systems (RDBMS), fueling the widespread adoption of Not Only Structured Query Language (NoSQL) databases. This survey provides a comprehensive examination of NoSQL systems, offering a structured analysis of their design motivations, classification schemes, architectural models, and performance characteristics. We explore how different storage engines, consistency models, and replication strategies shape system behavior under diverse workloads. A comparative study of benchmark results across representative NoSQL platforms is presented, complemented by a domain-driven mapping of use cases to database categories, which bridges architectural insight with deployment-centric perspectives. Additionally, we highlight ongoing research challenges related to interoperability, security, and intelligent query processing. This study aims to serve as both a reference and a decision-support tool for researchers and practitioners navigating the evolving NoSQL landscape.**

***Index Terms*—NoSQL databases, distributed data systems, storage engines, consistency models**

I. Introduction

The last two decades have witnessed a radical shift in data characteristics, with applications increasingly generating high-volume, high-velocity, and heterogeneous datasets. Traditional RDBMS, grounded in normalized schemas and strong consistency guarantees, were not designed to accommodate such scale and variability. Their limitations in horizontal scalability, schema flexibility, and eventual consistency tolerance have become evident in modern distributed environments [1].

In response, NoSQL databases have emerged as a diverse class of systems that abandon one-size-fits-all relational models in favor of specialized architectures optimized for distributed data management. These systems vary significantly in terms of data models, storage engines, replication strategies, and consistency semantics, resulting in a fragmented landscape that challenges selection, evaluation, and deployment. A structured investigation into their design and operational behavior is necessary to support informed decisions in both research and practice [2].

A. Motivation

Modern applications increasingly operate in distributed, high-throughput environments that demand scalability, fault tolerance, and schema flexibility. While traditional RDBMS architectures offer maturity and strong transactional guarantees, their reliance on fixed schemas, monolithic deployments, and strict consistency models limits their adaptability to dynamic workloads such as cloud-native services, Internet of Things (IoT) streams, and real-time analytics.

NoSQL databases emerged to address these limitations by embracing distribution, relaxed consistency, and diverse data models. Yet, the proliferation of NoSQL systems with distinct architectures and trade-offs has introduced complexity in evaluating their suitability. A unified, comparative perspective is essential to align architectural design with performance expectations and application needs, particularly as organizations seek tailored solutions for evolving data infrastructures.

B. Methodology

The study adopts a structured and reproducible approach to identify, select, and analyze recent literature on NoSQL database systems. Relevant works were retrieved from major scientific databases including IEEE Xplore, ACM Digital Library, SpringerLink, Elsevier ScienceDirect, and Google Scholar. The selection focused on peer-reviewed journals, conference proceedings, and survey articles published from 2020 onward, emphasizing architectural models, storage mechanisms, performance benchmarking, and application domains. Studies that merely introduced isolated prototypes, lacked architectural depth, or focused exclusively on relational systems were excluded. After eliminating duplicates and applying relevance filtering at the abstract level, a total of 45 publications were retained.

The review process was structured into three stages. In the first stage, keyword-based queries were used to capture literature involving terms such as "NoSQL databases," "data consistency," "distributed storage," and "scalable database systems." In the second stage, selected studies were manually examined to extract metadata related to system architecture, storage engines, consistency models, and performance metrics. In the third stage, the findings were synthesized into a unified taxonomy that supports comparative evaluation across system design, operational trade-offs, and domain-level applicability. This method ensures a comprehensive and balanced understanding of the NoSQL ecosystem and its evolving role in distributed data management.

C. Contribution

This survey offers a focused and structured analysis of NoSQL database systems by integrating architectural insights, performance evaluations, and domain-specific applications. It introduces a classification grounded in core design features, contrasts storage engine implementations and consistency trade-offs, and examines how different systems behave under benchmarking conditions. The study further maps NoSQL categories to real-world domains, clarifying practical deployment patterns. In contrast to existing surveys, which often emphasize either theoretical models or isolated performance metrics, this work connects internal system architecture with operational behavior across diverse contexts. Finally, it identifies open challenges and outlines future research directions in consistency management, query optimization, and secure data distribution.

The rest of the paper is organized as follows: Section II outlines NoSQL fundamentals and classification. Section III covers architectural models and design choices. Section IV reviews benchmarking methods and compares system performance. Section V maps NoSQL categories to real-world applications. Section VI discusses related surveys, challenges, and research directions. Section VII concludes the survey.

II. Background and Classification of NoSQL Systems

The term "NoSQL" refers to a broad class of non-relational data systems developed to address scalability and flexibility limitations of traditional relational databases. Understanding their architectural and performance traits requires examining their historical background, common design principles, and core classification categories.

A. From Relational Limits to New Demands

The rise of NoSQL databases stems from the limitations of traditional RDBMS in handling modern, data-intensive applications. Rooted in fixed schemas and SQL-based querying, classical RDBMS are reliable for transactional and structured workloads but struggle with the scale, heterogeneity, and unstructured nature of data from web platforms, IoT, and real-time analytics [3].

Their rigidity in schema evolution and dependence on vertical scaling hinder performance in distributed environments where horizontal scalability and availability are essential. Furthermore, JOIN complexity and atomicity, consistency, isolation, and durability (ACID) guarantees introduce latency overhead in high-throughput applications. These constraints led developers to seek alternatives that relax consistency for scalability, aligning with eventual consistency in large-scale systems [4].

This shift drove the adoption of NoSQL databases, designed for schema flexibility, distributed scalability, and efficient handling of semi-structured and unstructured data. As cloud and big data infrastructures matured, NoSQL systems reshaped data management strategies across sectors [5].

B. Common Characteristics of NoSQL Databases

Despite their heterogeneity, NoSQL systems share core characteristics that differentiate them from relational databases. Their schema-flexible design enables storage of heterogeneous records and supports dynamic evolution without costly migrations, an asset for applications with fluid or partially defined data formats [6].

NoSQL architectures prioritize horizontal scalability by distributing data across clusters through mechanisms such as sharding and replication, thereby achieving elasticity and fault tolerance. Instead of enforcing strict transactional guarantees, they adopt the available, soft state, eventually consistent (BASE) model over the traditional ACID approach. This design choice aligns with the consistency, availability, and partition tolerance (CAP) theorem, favoring availability and partition tolerance, particularly in workloads, such as personalization or session management, where immediate consistency is not critical [7].

NoSQL systems expose lightweight, model-aligned query interfaces that enhance speed and scalability, though they trade off the expressiveness and generality of traditional SQL [8].

C. Core Categories of NoSQL Systems

NoSQL databases are categorized by their underlying data models, each optimized for specific application semantics and access patterns. The four primary types are key-value stores, document databases, column-family stores, and graph databases [9].

Key-value stores (e.g., Redis, DynamoDB) pair unique keys with opaque values for ultra-fast access, ideal for caching and session storage, though they lack structure and secondary indexing. Document databases (e.g., MongoDB, Couchbase) store nested JavaScript object notation (JSON) and binary JSON (BSON) documents with field-level indexing and flexible queries, making them suitable for content management and dynamic application programming interfaces (APIs). Column-family stores (e.g., Cassandra, HBase) use sparse tables with tunable consistency and partitioning, optimized for high-throughput writes and time-series data. Graph databases (e.g., Neo4j, Neptune) model data as nodes and edges, enabling efficient relationship queries for applications like social networks and fraud detection [10].

Table I contrasts the four main NoSQL database categories by examining their data models, structural flexibility, query support, indexing features, and consistency models. This comparative view clarifies systems' architectural and operational differences, providing a foundation for the following architectural analysis.

III. Architectural Models and Design Choices

NoSQL architectures are designed for scalable, distributed, and flexible data management, diverging from the rigidity of traditional relational systems. This section outlines key components and architectural trade-offs that impact their scalability, performance, and consistency.

A. Consistency Models

A central design challenge in NoSQL systems is the trade-off among consistency, availability, and partition tolerance, as articulated by the CAP theorem. In distributed environments, it is impossible to guarantee all three simultaneously under network partitioning. Consequently, many NoSQL systems

TABLE I
COMPARISON OF CORE NOSQL CATEGORIES.

Category	Data Model	Structure Flexibility	Query Capabilities	Indexing Support	Consistency Model	Representative Systems
Key-Value Store	Key → Opaque value (e.g., binary, JSON)	Rigid at value level	Exact match by key	None (application-level only)	Tunable / Eventual	Redis, DynamoDB
Document Store	Key → Document (e.g., JSON/BSON)	High (semi-structured)	Field-based filtering, aggregation	Yes (field-level)	Tunable / Eventual or Strong	MongoDB, Couchbase
Column-Family Store	Row key → Column families	Sparse and variable per row	Range queries, limited joins	Yes (column or row-level)	Tunable / Quorum-based	Cassandra, HBase
Graph Database	Nodes and edges with properties	Flexible per node/edge	Graph traversal (e.g., path, subgraph)	Yes (property-based)	ACID (single-node) / Eventual (distributed)	Neo4j, Amazon Neptune

prioritize availability and partition tolerance, adopting weaker consistency models [11].

This trade-off is often embodied in the BASE approach, which contrasts with the strict ACID guarantees of relational systems. BASE allows continued operation during partial failures, with eventual replica convergence. Consistency models vary across NoSQL platforms. MongoDB and Cassandra support tunable consistency, enabling developers to balance read/write trade-offs according to workload demands [12].

B. Storage Engine Design: LSM-Trees vs. B-Trees

NoSQL systems vary in their choice of storage engines, directly impacting write amplification, read latency, and compaction behavior. Two dominant structures are Log-Structured Merge Trees (LSM-trees) and B-trees. LSM-trees, used in Cassandra and LevelDB, prioritize write throughput by buffering in-memory writes and flushing sorted batches to disk, minimizing random Input/Output. Though efficient for writes, they incur read amplification, often offset via Bloom filters and tiered caches [13].

B-Trees, prevalent in systems like Couchbase and traditional RDBMS, support in-place updates and fast lookups with low read amplification. However, they suffer from page-level locking and fragmented disk writes under write-intensive or concurrent loads. Their choice reflects a trade-off: LSM-trees suit write-heavy scenarios, while B-Trees favor read-intensive applications [14].

C. Data Distribution and Replication Strategies

Horizontal scalability in NoSQL systems is achieved through partitioning and replication to ensure balanced load and fault tolerance. Sharding distributes data across nodes using key ranges or consistent hashing, the latter employed in DynamoDB and Riak, allows minimal reshuffling during scaling events [15].

Replication strategies differ: synchronous replication ensures strong consistency but increases latency and reduces availability under failure; asynchronous replication favors availability and speed, with weaker consistency. Quorum-based replication, as in Cassandra, allows adjustable consistency levels by requiring a configurable number of replicas to confirm operations, enabling fine-grained trade-offs between consistency and performance [16].

D. Partitioning and Consistency Techniques

Effective partitioning is essential for maintaining system responsiveness and avoiding hotspots in distributed NoSQL deployments. Partitioning schemes typically fall into two categories: range-based and hash-based. Range partitioning enables ordered scans and range queries but risks uneven data distribution under skewed keys. Hash partitioning, while mitigating skew, complicates query execution involving key ranges or joins [17].

Consistency management across partitions introduces additional challenges. Systems must reconcile conflicting updates, often using strategies like last-write-wins (LWW), vector clocks, or custom conflict resolution logic. Conflict resolution mechanisms are embedded at the application or middleware layer in multi-master setups. The architectural decision around these techniques fundamentally shapes the balance between performance, correctness, and developer complexity [18].

E. Query Processing and Indexing Mechanisms

In contrast to relational systems that offer declarative SQL and cost-based optimizers, NoSQL databases provide streamlined, model-specific query interfaces. Query capabilities vary widely, reflecting indexing, consistency, and join support trade-offs. Key-value stores enable only primary key access, while document and column-family databases allow filtering, range queries, and aggregations. Secondary indexes, when present, may be global or per-shard, balancing lookup efficiency with update cost. Graph databases offer powerful traversal queries but face challenges in distributed execution due to inter-node dependencies [19].

Most NoSQL systems avoid join operations, favoring denormalized schemas for performance and scalability. While this enhances lookup speed and simplifies partitioning, it pushes join logic to the application layer. Some platforms have reintroduced limited join or multi-document transaction support, though typically at the expense of scalability [20].

Table II summarizes core architectural choices across NoSQL categories, including storage, replication, partitioning, indexing, and query support. These design dimensions shape system scalability, latency, and complexity under real-world workloads.

IV. BENCHMARKING AND PERFORMANCE

Benchmarking NoSQL systems demands flexible frameworks that account for architectural heterogeneity. Unlike

TABLE II
ARCHITECTURAL DESIGN CHARACTERISTICS OF NOSQL CATEGORIES.

Category	Storage Engine	Replication	Partitioning	Indexing	Query Support
Key-Value Store	LSM-tree or in-memory	Asynchronous: quorum-based	Consistent hashing	None or minimal	Key lookups only
Document Store	B-Tree or LSM hybrid	Configurable: synchronous or asynchronous	Hash or range-based	Optional field-level	Filtering, aggregation, limited joins
Column-Family Store	LSM-tree	Quorum-based	Hash partitioning	Row and column indexes (optional)	Range queries, batched scans
Graph Database	Custom or memory-mapped	Master-slave / eventual	Manual or semantic sharding	Node and edge property indexes	Pattern matching, graph traversal

relational databases, which benefit from standardized benchmarks like the Transaction Processing Performance Council Benchmark C (TPC-C), NoSQL workloads differ in data models, indexing strategies, and consistency semantics. The Yahoo! Cloud Serving Benchmark (YCSB) is the most widely adopted, offering tunable read/write ratios and request distributions to evaluate throughput, latency, and scalability. Real-world traces from domains such as telemetry and graph traversal complement synthetic benchmarks by capturing production-like behaviors [21], [22].

Performance characteristics vary across categories. Key-value stores like Redis achieve sub-millisecond latency via in-memory execution but lack expressive queries. Document stores such as MongoDB offer a balance of flexibility and performance, though deep nesting can introduce processing overhead. Column-family systems like Cassandra deliver high write throughput through log-structured storage and tunable consistency, typically outperforming HBase in ingestion-heavy workloads. Graph databases such as Neo4j excel in deep traversals, as measured by the Linked Data Benchmark Council Social Network Benchmark (LDBC SNB), but face challenges in distributed query execution [23], [24].

Scalability studies show that systems using consistent hashing and asynchronous replication, e.g., Cassandra and DynamoDB, scale linearly when partitioning and compaction are optimized. In contrast, master-slave architectures often incur coordination overhead. Consistent choices also shape performance: stronger guarantees increase latency and reduce throughput, while eventual consistency improves responsiveness but risks anomalies. Recent benchmarks increasingly include operational metrics and cost-awareness to reflect the demands of cloud-native, elastic environments [25], [26].

Table III outlines key performance traits of representative NoSQL systems, reflecting differences in latency, throughput, and scalability due to architectural and consistency choices. These disparities underscore the need for workload-aware tuning and complicate standardized benchmarking.

TABLE III
PERFORMANCE OF REPRESENTATIVE NOSQL SYSTEMS.

System	Read Latency	Write Throughput	Scalability	Consistency Impact
Redis	Very low (< 1 ms)	High (bounded by available memory)	High (limited by RAM capacity)	Minimal; lacks strong consistency
MongoDB	Moderate (1–10 ms)	Moderate to high	Moderate (enabled by sharding)	Tunable; strong consistency increases latency
Cassandra	Moderate to high	Very high (LSM-based)	Excellent (linear scalability)	Tunable; quorum configuration affects throughput
HBase	Moderate to high	High (optimized for batch writes)	High (integrated with HDFS)	Strong; coordination overhead increases latency
Neo4j	Low (graph traversals)	Low to moderate	Limited (optimal as single-node)	Strong (local ACID compliance); weaker consistency in clustered mode

V. APPLICATION DOMAINS AND USE CASES

The widespread adoption of NoSQL databases stems from their ability to support high-throughput workloads, heterogeneous data, and horizontally scalable architectures. Each category aligns with specific operational needs across diverse application domains [27].

Key-value stores like Redis and DynamoDB in web-scale services serve as low-latency caching layers and session backends, enabling real-time personalization and ephemeral state management in platforms like Amazon and Shopify. Document stores like MongoDB and Couchbase are used in content management, product catalogs, and mobile backends, offering schema flexibility for semi-structured JSON data and native support for distributed synchronization [28], [29].

Column-family systems, notably Apache Cassandra, are favored in telemetry, finance, and IoT infrastructures due to their write-optimized architecture and tunable consistency. They handle high-ingestion, append-only workloads and support geographically distributed deployments for real-time analytics [30].

Graph databases like Neo4j and Amazon Neptune power relationship-centric applications such as social networks, fraud detection, and knowledge graphs, offering expressive traversal queries essential for modeling complex interactions. In healthcare and genomics, hybrid NoSQL deployments combine document and graph models to manage clinical data and biological networks, supporting machine learning (ML) tasks like risk scoring and treatment recommendation [31], [32].

Cloud-native environments have further accelerated NoSQL adoption. Managed services like Firebase, DynamoDB, and Azure Cosmos DB enable elastic, globally distributed deployments with cost-efficient scaling. These capabilities make NoSQL systems integral to Software as a Service (SaaS) platforms and agile development [33].

The diversity and flexibility of NoSQL systems collectively allow them to meet application-specific requirements in ways traditional RDBMS cannot, particularly where decentraliza-

TABLE IV
APPLICATION DOMAINS AND USE CASES ACROSS NOSQL CATEGORIES.

Domain	NoSQL Category	Representative Use Case
Web-scale services	Key-Value Store	Session management, caching, real-time recommendation
E-Commerce; Content Management System	Document Store	Schema-flexible product catalogs, Content management system backends, API payload storage
IoT;Telemetry	Column-Family Store	High-ingestion time-series logging, telemetry aggregation
Social Networks	Graph Database	Social graph traversal, influence modeling
Cybersecurity;Fraud	Graph Database	Relationship-based anomaly detection, access pattern analysis
Mobile Applications	Document Store	Offline synchronization, schema-less mobile storage
Financial Systems	Column-Family Store	Append-heavy transaction pipelines, write-optimized storage
Healthcare;Genomics	Document + Graph Hybrid	Semi-structured medical records, gene-drug graphs
Cloud-native platforms	Multi-model (cloud-managed)	Multi-tenant SaaS, globally distributed microservices

tion, scalability, and schema agility are paramount [34].

Table IV summarizes how NoSQL database categories align with key application domains, highlighting typical deployments and use cases that leverage each model's strengths.

VI. DISCUSSION AND FUTURE TRENDS

Recent survey studies have extensively mapped the landscape of NoSQL systems, offering various perspectives on performance, security, data modeling, and domain-specific adaptation. The work [35] provides a comprehensive taxonomy of privacy risks in distributed NoSQL architectures, underscoring the absence of unified access control models. Also, [36] focuses on structural and transactional contrasts between SQL, NewSQL, and NoSQL systems, clarifying where NoSQL databases diverge from traditional ACID paradigms. [37] conducts a comparative study of MySQL and MongoDB, revealing that document stores offer flexibility and often underperform under transactional workloads. [38] assesses NoSQL scalability in cloud environments, documenting variability in latency and throughput across providers. On the application frontier, [39] highlights the relevance of NoSQL for geospatial information processing, noting graph and document models as key enablers. Similarly, [40] emphasizes NoSQL adoption in Industry 4.0, where hybrid data sources demand schema-agnostic storage. Finally, the survey [41] stresses that NoSQL systems must still evolve to match the analytical depth and maturity of RDBMSs, especially for mission-critical analytics.

Although existing surveys provide valuable insights into specific NoSQL aspects, they are often fragmented, isolating performance, architecture, or applications. Instead, this paper offers an integrated survey linking architectural principles (e.g., CAP trade-offs, storage and replication models) with empirical performance trends and deployment contexts. It provides a coherent view of consistency, scalability, and workload suitability by aligning design decisions with real-world constraints.

For practitioners, the comparative analysis of storage engines, query semantics, and replication strategies offers practical guidance under performance and consistency trade-offs. When scaling telemetry pipelines or multi-tenant SaaS, understanding LSM-tree behavior or CAP implications is essential. Theoretically, current security models lack formal, consistency-preserving access control. At the same time, the absence of benchmarks and cross-model queries highlights research gaps in meta-query translation across document, columnar, and graph systems [42], [43].

As data systems grow more complex, NoSQL research must move beyond optimization to address interoperability, introspection, and formal guarantees. Multi-model systems aim to integrate document, graph, and key-value semantics, yet face challenges in unified querying, indexing, and schema evolution. Future work should develop abstraction layers for accurate cross-model translation and adaptive cloud-native deployment. Promising directions include ML-based indexing, predictive caching, and explainable operations to support trustworthy artificial intelligence pipelines. Revisiting CAP/BASE under modern infrastructure may lead to hybrid models, while privacy-preserving mechanisms (e.g., differential privacy, secure computation) will be key to compliant, cross-organizational data sharing [44], [45].

VII. CONCLUSION

This survey presented a structured overview of NoSQL database systems, highlighting their evolution beyond traditional relational models and their alignment with the needs of modern data-intensive applications. Through a layered analysis of architectural models, storage mechanisms, replication strategies, and consistency trade-offs, we clarified how core design choices impact scalability, availability, and performance. Our classification of NoSQL systems and synthesis of benchmarking results offers a practical foundation for system selection across diverse application domains. Unlike prior surveys, which focus narrowly on theoretical aspects or performance metrics, this study bridges architectural understanding with real-world implications. Practitioners can benefit from the comparative insights into storage and query models, while researchers may find guidance in the open issues mapped across system categories. As data systems continue to evolve under the pressures of cloud elasticity, heterogeneous workloads, and trust-aware computation, we believe this work can serve as both a reference point and a springboard for further innovation.

REFERENCES

[1] E. Dritsas and M. Trigka, "Database systems in the big data era: Architectures, performance, and open challenges," *IEEE Access*, vol. 13, pp. 95 068–95 084, 2025.

[2] W. Khan, T. Kumar, C. Zhang, K. Raj, A. M. Roy, and B. Luo, "Sql and nosql database software architecture performance analysis and assessments—a systematic literature review," *Big Data and Cognitive Computing*, vol. 7, no. 2, p. 97, 2023.

[3] S. R. S. Al Maamari and M. Nasar, "A comparative analysis of nosql and sql databases: Performance, consistency, and suitability for modern applications with a focus on iot," *East Journal of Computer Science*, vol. 1, no. 2, pp. 10–15, 2025.

[4] L. F. Da Silva and J. V. Lima, "An evaluation of relational and nosql distributed databases on a low-power cluster," *The Journal of Supercomputing*, vol. 79, no. 12, pp. 13 402–13 420, 2023.

[5] A. Ali, S. Naeem, S. Anam, and M. M. Ahmed, "A state of art survey for big data processing and nosql database architecture," *International Journal of Computing and Digital Systems*, vol. 14, no. 1, pp. 1–1, 2023.

[6] A. Hillenbrand and U. Störl, "Managing schema migration in nosql databases: Advisor heuristics vs. self-adaptive schema migration strategies," in *International Conference on Model-Driven Engineering and Software Development*. Springer, 2021, pp. 230–253.

[7] M. K. Goyal and R. Chaturvedi, "The role of nosql in microservices architecture: Enabling scalability and data independence," *European Journal of Advances in Engineering and Technology*, vol. 9, no. 6, pp. 87–95, 2022.

[8] M. Kaufmann and A. Meier, "Sql and nosql databases," *Switzerland: Springer. doi*, vol. 10, pp. 978–3, 2023.

[9] E. Gupta, S. Sural, J. Vaidya, and V. Atluri, "Enabling attribute-based access control in nosql databases," *IEEE transactions on emerging topics in computing*, vol. 11, no. 1, pp. 208–223, 2022.

[10] I. Carvalho, F. Sá, and J. Bernardino, "Performance evaluation of nosql document databases: couchbase, couchdb, and mongodb," *Algorithms*, vol. 16, no. 2, p. 78, 2023.

[11] O. Khoshaba, V. Grechaninov, T. Molodetska, A. Lopushanskyi, and K. Zavertailo, "Study of the workspace model in distributed structures using cap theorem," in *International scientific-practical conference*. Springer, 2022, pp. 229–242.

[12] A. Gorbenko, A. Romanovsky, and O. Tarasyuk, "Interplaying cassandra nosql consistency and performance: A benchmarking approach," in *Dependable Computing-EDCC 2020 Workshops: AI4RAILS, DREAMS, DSOGRI, SERENE 2020, Munich, Germany, September 7, 2020, Proceedings 16*. Springer, 2020, pp. 168–184.

[13] Y. Qiao, X. Chen, N. Zheng, J. Li, Y. Liu, and T. Zhang, "Closing the b+-tree vs.{LSM-tree} write amplification gap on modern storage hardware with built-in transparent compression," in *20th USENIX Conference on File and Storage Technologies (FAST 22)*, 2022, pp. 69–82.

[14] C. Riegger and I. Petrov, "Storage management with multi-version partitioned btrees," *Information Systems*, vol. 125, p. 102403, 2024.

[15] M. Elhemali, N. Gallagher, N. Gordon, J. Idziorek, R. Krog, C. Lazier, E. Mo, A. Mritunjai, S. Perianayagam, T. Rath *et al.*, "Amazon {DynamoDB}: A scalable, predictably performant, and fully managed {NoSQL} database service," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 1037–1048.

[16] S. Ferreira, J. Mendonça, B. Nogueira, W. Tiengo, and E. Andrade, "Impacts of data consistency levels in cloud-based nosql for data-intensive applications," *Journal of Cloud Computing*, vol. 13, no. 1, p. 158, 2024.

[17] P.-J. Liu, C.-P. Li, and H. Chen, "Enhancing storage efficiency and performance: A survey of data partitioning techniques," *Journal of Computer Science and Technology*, vol. 39, no. 2, pp. 346–368, 2024.

[18] N. Preguiça, C. Baquero, and M. Shapiro, "Conflict-free replicated data types (crdts)," in *Encyclopedia of Big Data Technologies*. Springer, 2022, pp. 1–10.

[19] N. Naseri Seyedi Noudoust, S. Adabi, and A. Rezaee, "A quorum-based data consistency approach for non-relational database," *Cluster Computing*, vol. 25, no. 2, pp. 1515–1540, 2022.

[20] J. Idziorek, A. Keyes, C. Lazier, S. Perianayagam, P. Ramanathan, J. C. Sorenson III, D. Terry, and A. Vig, "Distributed transactions at scale in amazon {DynamoDB}," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 705–717.

[21] S. Ferreira, J. Mendonça, B. Nogueira, W. Tiengo, and E. Andrade, "Benchmarking consistency levels of cloud-distributed nosql databases using ycsb," *IEEE Access*, 2025.

[22] Z. Zhu, A. Saha, M. Athanassoulis, and S. Sarkar, "Kvbench: a key-value benchmarking suite," in *Proceedings of the Tenth International Workshop on Testing Database Systems*, 2024, pp. 9–15.

[23] X. Cui and W. Chen, "Performance comparison test of hbase and cassandra based on ycsb," in *2021 IEEE/ACIS 19th International Conference on Computer and Information Science (ICIS)*. IEEE, 2021, pp. 70–77.

[24] B. Lyu, X. Zhou, L. Lai, Y. Yang, Y. Lou, and Y. Liu, "Enhancing neo4j query efficiency with seamless integration of the gopt optimization framework," *Proceedings of the VLDB Endowment. ISSN*, vol. 2150, p. 8097, 2024.

[25] A. Krechowicz, S. Deniziak, and G. Łukawski, "Consistent, highly throughput and space scalable distributed architecture for layered nosql data store," *Scientific Reports*, vol. 15, no. 1, pp. 1–20, 2025.

[26] C. Araújo, M. Oliveira Jr, B. Nogueira, P. Maciel, and E. Tavares, "Performability evaluation of nosql-based storage systems," *Journal of Systems and Software*, vol. 208, p. 111885, 2024.

[27] M. Salahuddin, S. Majeed, S. Hira, and G. Mumtaz, "A systematic literature review on performance evaluation of sql and nosql database architectures," *Journal of Computing & Biomedical Informatics*, vol. 7, no. 02, 2024.

[28] S. Sethi and S. Panda, "Sql or nosql—practical aspect and rational behind choosing data stores," *Journal of Computer and Communications*, vol. 12, no. 8, pp. 155–174, 2024.

[29] I. Carvalho, F. Sá, and J. Bernardino, "Nosql document databases assessment: Couchbase, couchdb, and mongodb." in *DATA*, 2022, pp. 557–564.

[30] M. Silva-Muñoz, A. Franzin, and H. Bersini, "Automatic configuration of the cassandra database using irace," *PeerJ Computer Science*, vol. 7, p. e634, 2021.

[31] G. A. Atemezing, "Empirical evaluation of a cloud-based graph database: the case of neptune," in *Knowledge Graphs and Semantic Web: Third Iberoamerican Conference and Second Indo-American Conference, KGSWC 2021, Kingsville, Texas, USA, November 22–24, 2021, Proceedings 3*. Springer, 2021, pp. 31–46.

[32] S. Timón-Reina, M. Rincón, and R. Martínez-Tomás, "An overview of graph databases and their applications in the biomedical domain," *Database*, vol. 2021, p. baab026, 2021.

[33] D. Preuveneers and W. Joosen, "Automated configuration of nosql performance and scalability tactics for data-intensive applications," in *Informatics*, vol. 7, no. 3. MDPI, 2020, p. 29.

[34] S. Belefqih, A. Zellou, and M. Berquedich, "Schema extraction in nosql databases: a systematic literature review," *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, vol. 17, no. 8, pp. 92–104, 2024.

[35] S. Sicari, A. Rizzardi, and A. Coen-Porisini, "Security & privacy issues and challenges in NoSQL databases," *Computer Networks*, vol. 206, p. 108828, 2022.

[36] T. N. Khasawneh, M. H. AL-Sahlee, and A. A. Safia, "SQL, NewSQL, and NoSQL databases: A comparative survey," in *2020 11th International Conference on Information and Communication Systems (ICICS)*. IEEE, 2020, pp. 013–021.

[37] S. Palanisamy and P. SuvithaVani, "A survey on RDBMS and NoSQL databases MySQL vs MongoDB," in *2020 international conference on computer communication and informatics (ICCCI)*. IEEE, 2020, pp. 1–7.

[38] T. H. Shareef, K. H. Sharif, and B. N. Rashid, "A survey of comparison different cloud database performance: Sql and nosql," *Passer Journal of Basic and Applied Sciences*, vol. 4, no. 1, pp. 45–57, 2022.

[39] D. Guo and E. Onstein, "State-of-the-art geospatial information processing in NoSQL databases," *ISPRS International Journal of Geo-Information*, vol. 9, no. 5, p. 331, 2020.

[40] V. F. de Oliveira, M. A. d. O. Pessoa, F. Junqueira, and P. E. Miyagi, "SQL and NoSQL databases in the context of industry 4.0," *Machines*, vol. 10, no. 1, p. 20, 2021.

[41] M. Arshad, M. N. Brohi, T. R. Soomro, T. M. Ghazal, H. M. Alzoubi, and M. Alshurideh, "NoSQL: Future of bigdata analytics characteristics and comparison with RDBMS," in *The Effect of Information Technology on Business and Marketing Intelligence Systems*. Springer, 2023, pp. 1927–1951.

[42] A. K. Y. S. Mohamed, D. Auer, D. Hofer, and J. Küng, "A systematic literature review of authorization and access control requirements and current state of the art for different database models," *International Journal of Web Information Systems*, vol. 20, no. 1, pp. 1–23, 2024.

[43] A. Raman, A. Huynh, J. Lu, and M. Athanassoulis, "Benchmarking learned and lsm indexes for data sortedness," in *Proceedings of the Tenth International Workshop on Testing Database Systems*, 2024, pp. 16–22.

[44] H. Lei, C. Li, K. Zhou, J. Zhu, K. Yan, F. Xiao, M. Xie, J. Wang, and S. Di, "X-stor: A cloud-native nosql database service with multi-model support," *Proceedings of the VLDB Endowment*, vol. 17, no. 12, pp. 4025–4037, 2024.

[45] T. Biswas, "Privacy-preserving llm integration with scientific nosql repositories: A differential privacy approach," *World Journal of Engineering and Technology*, vol. 13, no. 2, pp. 329–345, 2025.