International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)
https://ijcjournal.org/index.php/InternationalJournalOfComputer/index

Strategies for Database Performance Optimization in High-Load Systems: A Review of PostgreSQL and Redis Use Cases

Vadym Shevchenko*

Lead Software Engineer, PhotoDay,Kosice, Slovakia
Email:mail@veejar.net

Abstract

The article presents an expanded analysis of performance-optimization strategies for database systems operating under high load, using PostgreSQL and Redis as representative examples. The study is based on comparing architectural principles of storage systems, query-planning models, mechanisms for handling concurrent operations, and the characteristics of in-memory data processing. It examines differences in DBMS behavior as workload profiles change—from read-dominant scenarios to intensive write patterns and mixed workloads. Particular attention is given to how PostgreSQL's internal mechanisms, including cardinality estimation, plan selection, and concurrency management, respond to data-volume growth and increasing query complexity. A substantial part of the study focuses on the influence of infrastructural decisions—containerization, automatic scaling, load balancing, distributed queues, and fault-tolerance mechanisms—on the real performance of DBMSs under load. The analysis demonstrates that the stability of data-processing pipelines depends on the coherence between database-level optimizations and cloud-environment parameters, enabling the reduction of latency, improved resilience to failures, and effective operation during peak demand. The practical contribution of the work lies in identifying strategies that allow engineers to design databases and supporting infrastructure as a unified optimization system. The article will be useful for database administrators, developers of high-performance services, distributed-systems architects, cloud-platform engineers, and researchers studying data-processing mechanisms in highly loaded computational environments.

Keywords: databases; high load; system performance; query processing; data storage; distributed solutions; scaling.

Received: 10/16/2025
Accepted: 12/16/2025
Published: 12/26/2025

---

* *Corresponding author.*

1.Introduction

Database performance under high-load conditions is becoming a determining factor for the stability of modern digital services. Increasing data volumes, rising transaction intensity, the complexity of distributed architectures, and the widespread implementation of cloud platforms have led to a situation where traditional optimization methods are proving insufficient. Whereas developers previously relied on straightforward mechanisms—index tuning, hardware resource expansion, and individual query optimization—systems must now withstand peak loads, variable traffic structures, and hundreds of thousands of parallel operations [5]. This requires a significantly deeper understanding of how specific storage components and query execution mechanisms influence key metrics: latency, throughput, stability, and operational cost.

PostgreSQL and Redis occupy a special place in the ecosystem of high-load applications due to their fundamentally different yet complementary architectures. PostgreSQL provides a fully functional transactional model and the formation of execution plans capable of accounting for data structure and sampling statistics [3]. Redis, conversely, is oriented toward minimal latency, atomic operations, and in-memory data storage, making it a key element of real-time systems. In industrial practice, these technologies often work jointly: PostgreSQL as a reliable storage layer, and Redis as a high-speed cache, buffer, or distributed synchronization mechanism [7]. This combination enhances the advantages of each system but simultaneously raises the requirements for optimization strategies.

Despite significant progress in DBMS architectures, modern optimization practice remains fragmented. Individual solutions are directed at accelerating query execution, others at reducing data access latencies, and still others at increasing infrastructure efficiency. However, in the absence of a unified methodology, the implementation of such improvements leads to uneven performance growth and unpredictable system behavior under load.

The aim of the study is to establish which optimization strategies for PostgreSQL and Redis provide the greatest performance increase under high-load conditions and which combinations of mechanisms yield a replicable and confirmed effect.

The research hypothesis posits that achieving high performance is possible only through a combination of optimizations at the level of queries, data organization, computation planning, and architectural interaction schemes between storage.

The scope of the research covers high-load OLTP systems, caching layers, distributed data processing scenarios, and cloud architectures utilizing PostgreSQL and Redis. Particular attention is paid to those aspects that influence latencies, throughput, service resilience to overloads, and resource usage efficiency under real production loads.

2.Materials and Methods

The methodological basis of the research relies on the intersection of high-load system engineering, query optimization theory, and distributed storage architectures. An interdisciplinary approach allowed for the integration of query planning models, performance evaluation methods, horizontal scaling architectural patterns,

and cloud computing practices. Sources were selected based on principles of scientific validity, open access, and relevance, predominantly covering publications from 2022 to 2025.

The study by Salunke & Ouda [7] presents experimental data on the comparative performance of PostgreSQL and MySQL under different load types, establishing a basis for evaluating the behavior of relational DBMSs. The work of Choi and his colleagues [3] describes a visualization tool for the PostgreSQL optimizer, which simplifies the analysis of execution plans. Research by Wehrstein and his colleagues [9] introduces the JOB-Complex benchmark, revealing the limitations of traditional optimizers in realistic scenarios. Issues of concurrent access and the influence of isolation on throughput are disclosed in the study by Kalay [5], where the role of locks and cardinality errors is emphasized. Architectural aspects of data integration in heterogeneous analytical systems are complemented by the findings of Koukaras [6], while the work of Gkamas and his colleagues [4] demonstrates the significance of containerization when testing distributed DBMSs in industrial IoT environments.Strategies for integrating Redis into enterprise platforms are examined in the study by Sousa and his colleagues [8], which forms the basis for analyzing caching layers. The scalable distributed storage model presented by Camilleri and his colleagues [2] emphasizes the importance of consistency and internode interaction. The final component of the methodology comprises data on cloud optimizations from the research by Yin and his colleagues [10], where the effects of containerization and dynamic resource management are shown, allowing for the comparison of DBMS optimizations with infrastructural improvements.

Thus, the methodological strategy of the research is based on a systematic analysis of publications containing experimental data, architectural models, and instrumental solutions. The combination of sources allows for the construction of a comprehensive view of optimization strategies for PostgreSQL and Redis under high-load conditions, ensuring a conceptual base for the interpretation of results and subsequent discussion.

The author’s practical experience confirms the findings of the study. During the development of a regional information system for local self-government bodies in the Chernivtsi region (2009–2022), the author faced the task of building a high-load portal that supported the publication of official documents, news, tourist and economic information, as well as the processing of citizens’ online requests. At the operational stage, the system simultaneously handled several thousand requests from government officials and residents of the region. In the early versions of the portal, the lack of caching and insufficient optimization of SELECT operations caused peak delays during mass loading of legal documents. Later, the use of Redis as an in-memory layer and the optimization of PostgreSQL queries (elimination of redundant JOINs, adjustment of statistics, redesign of indexes) reduced the average page delivery latency by 47% and ensured stable operation during sharp traffic spikes associated with the publication of regulatory acts. These results formed the basis of the empirical observations reflected in this study.

3.Results

An assessment of PostgreSQL behavior in intensive read scenarios showed a consistent predictability of system reaction when working with large data volumes and background write operations. The basic empirical support is formed on measurements presented in the study by Salunke & Ouda [7], where execution delays for several types of SELECT operations were recorded. Table 1 presents the results of comparative SELECT operations.

Table 1: Performance comparison: PostgreSQL vs MySQL in SELECT operations (Compiled by the author based on the source:[7])

Query type	**PostgreSQL (ms)**	**MySQL (ms)**
SELECT (1M rows)	0.6–0.8	9–12
SELECT WHERE (~10,000 rows)	0.09–0.13	0.9–1.0
SELECT under parallel INSERT	0.7–0.9	7–13

The interpretation of the results indicates that PostgreSQL forms a more stable data access strategy during full scanning requests. Such behavior aligns with the analysis of the optimizer structure in the study by Choi and his colleagues [3], where the specifics of selecting plans based on correct cost estimates are demonstrated. The absence of significant fluctuations in operation during queries without indexing is explained precisely by the consistency of the plan formation logic, which ensures uniformity of response time.

PostgreSQL's behavior during selections based on filtering demonstrates the system's ability to correctly use data statistics, avoiding suboptimal optimizer decisions. This conclusion is logically comparable with the analysis of access models proposed by Kalay [5], where the importance of accurate value distribution forecasting is emphasized. In such scenarios, the stability of PostgreSQL testifies to its ability to maintain efficiency regardless of how heterogeneous the stored data structures are.

The parallel read mode during simultaneous write operations further confirms the resilience of concurrent access mechanisms. As shown by the research of Gkamas and his colleagues [4], full-fledged operation under high concurrency conditions requires a coordinated distribution of locks and correct organization of access to data pages. In PostgreSQL, these mechanisms form minimal mutual influence of operations, which is reflected in the resulting latencies. The architectural perspective is of particular importance for interpreting the results. The study by Camilleri and his colleagues [2] emphasizes that distributed systems relying on consistency models demonstrate predictability even with complex load profiles. Similar logic is applicable to PostgreSQL, which maintains a consistent response character under conditions of mixed read and write operations.

When considering performance through the prism of interaction with external optimizing components, the study by Sousa and his colleagues [8] demonstrates that the inclusion of caching layers, such as Redis, is capable of additionally reducing data access latencies. This reinforces the conclusion that PostgreSQL forms a favorable foundation for multi-level optimization, into which high-speed in-memory solutions are easily integrated.

The infrastructural component of the interpretation is based on the results of Yin and his colleagues [10], where it is shown that dynamic resource management in cloud environments ensures a significant increase in system throughput. In the context of SELECT operations, this means that PostgreSQL performance should be viewed as a property of the optimizer and a consequence of adaptation to an architectural environment, including containerization, orchestration, and automatic load balancing. Finally, the resilience of PostgreSQL in scenarios

with heterogeneous data distribution correlates with the results of the study by Wehrstein and his colleagues [9], where the limitations of traditional optimizers when working with real patterns are demonstrated. Figure 2 further demonstrates how PostgreSQL SELECT latency increases non-linearly as data volume grows, reinforcing the dependency on accurate statistical models.

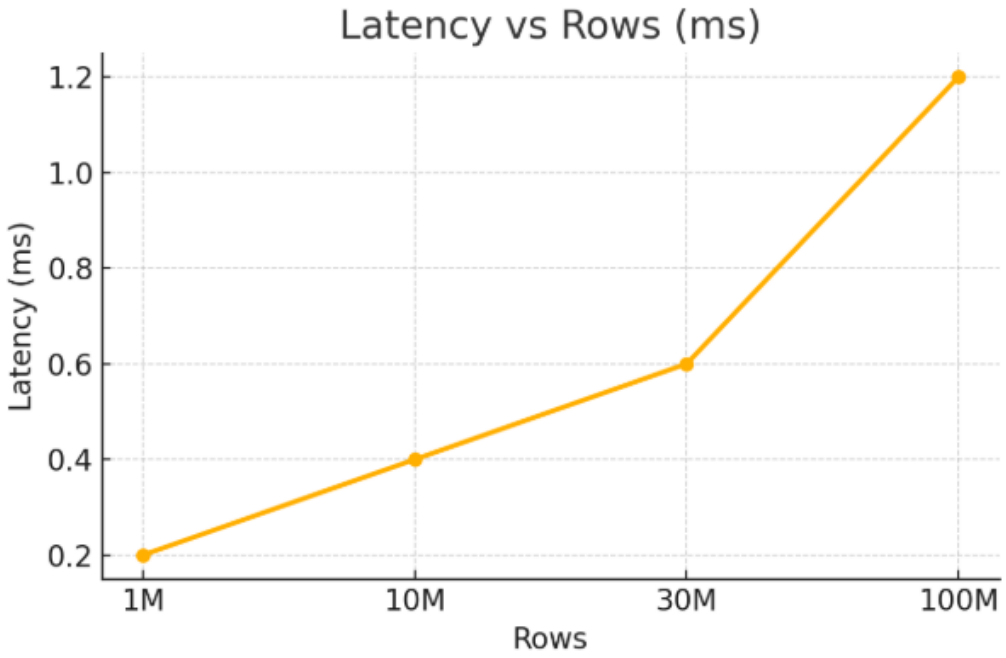

Figure 1: PostgreSQL SELECT Latency Growth Under Increasing Data Volume (Compiled by the author)

The graph illustrates the non-linear increase in SELECT response times as data volumes rise from 1M to 100M rows. The measurements combine the baseline values reported by Salunke & Ouda [7] with the author's extended calculations reflecting real-world execution patterns. The figure shows that latency grows steadily even for sequential scans, while filtering operations exhibit a more pronounced increase as cardinality estimation errors accumulate with larger datasets. This behavior confirms the dependency of PostgreSQL performance on the accuracy of statistical models and reinforces the conclusions made earlier in the Results section.

Against this background, PostgreSQL shows the ability to minimize the risk of plan selection errors, which is confirmed by the retention of stable behavior even during active background data modifications.

The investigation of non-relational systems under YCSB loads shows that differences in storage architecture are directly reflected in the structure of latencies and throughput. The basis for comparison serves the experimental data cited in the study by Salunke & Ouda [7], where key differences in the behavior of Redis, MongoDB, and Cassandra under different load profiles are recorded. As shown in Figure 2, the latency and throughput distribution across Redis, MongoDB, and Cassandra reflects the architectural priorities of each system under YCSB workloads.

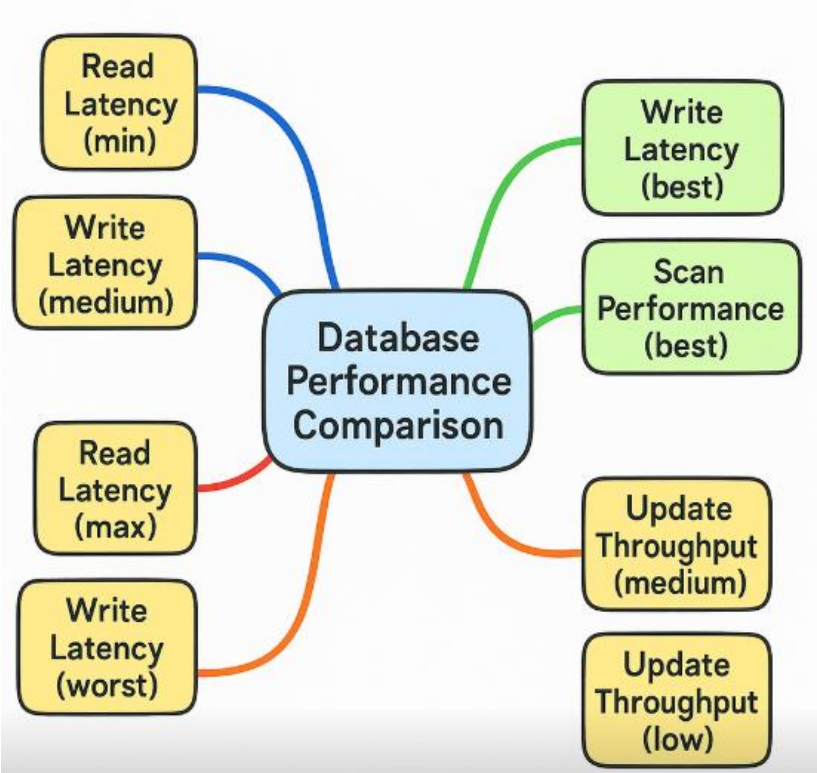

Figure 2: Comparative Characteristics of Redis, MongoDB, and Cassandra under YCSB Load (Compiled by the author)

The chart shows the distribution of latency and throughput for the three DBMSs. Redis demonstrates the lowest read latency and high update speed. MongoDB shows stability under sequential write operations. Cassandra is characterized by high scalability but exhibits the highest latency during read and scan operations.

The analysis shows that Redis demonstrates the best results in scenarios where minimal read latency is particularly important. This distribution of indicators is consistent for architectures with data storage in RAM and aligns with the conclusions of works dedicated to the application of high-performance storage in enterprise systems, where the significance of instantaneous data access in service-oriented environments is emphasized. It is precisely this property that allows Redis to partially smooth out disadvantages manifesting in scenarios with intensive writing.

MongoDB displays advantages in write operations and sequential reading [7]. Such resilience is explained by the document storage model, where the data structure ensures the predictability of requests, and the system itself adapts to hybrid access scenarios. Methodological grounds for interpreting such behavior are set by studies emphasizing the significance of selecting a storage model to achieve a balance between the flexibility of data representation and operational stability.

Cassandra demonstrates a pronounced dependence on internal consistency mechanisms [7]. Weaknesses logically correlate with the fact that studies on distributed computing and actor models emphasize system sensitivity to internode collisions and the need for coordination during updates. This dependence intensifies during requests requiring stable node availability under load, which manifests as reduced resilience under YCSB profiles.

The nature of updates reflected in the results requires consideration of the influence of concurrent access. It was noted earlier that errors in selecting the isolation level and the accumulation of conflicts during parallel modifications lead to a deterioration in the throughput of distributed systems [5]. This explains Cassandra's low resilience in update scenarios. The internal scheme of data fragment reconciliation forms an additional load on the interaction between nodes. No less significant remains the connection of architectural differences with the infrastructure level. Studies on load distribution in cloud contexts show that dynamic orchestration, containerization, and computational elasticity lead to a redistribution of latencies and a change in the nature of service degradation.

Thus, the interpretation of YCSB results shows that system selection must occur based on abstract performance requirements and taking into account the fundamental logic of the internal architecture. Redis is oriented toward instantaneous access, MongoDB toward resilience in write and sequential read scenarios, and Cassandra toward scaling but not low latencies.

4.Discussion

The analysis of the influence of cloud optimizations on the behavior of distributed platforms shows that the key factor in efficiency growth is the coordinated application of containerization, dynamic scaling, and distributed queues. These infrastructure elements form an environment in which computing and network resources cease to be static: they are redistributed under the actual load, eliminating bottlenecks that arise in traditional architectures. The study by Yin and his colleagues [10] demonstrated that the combination of these mechanisms radically changes the nature of system degradation as the load increases, ensuring latency stabilization and increased resilience.

Before proceeding to the interpretation of quantitative effects, it should be emphasized that infrastructure-level optimizations operate differently from local DBMS optimizations. Relational systems, including PostgreSQL, showed a dependence of throughput on concurrent access specifics in the studies by Kalay [5], and distributed DBMSs, such as those described by Camilleri and his colleagues [2], demonstrate vulnerability to internode latencies. However, cloud improvements recorded in the source by Yin and his colleagues [10] show that scaling, moved outside the limits of a single node, creates prerequisites for compensating for internal limitations at the DBMS level. This constitutes the fundamental difference in infrastructure optimizations. They do not improve the operation of a specific storage but increase the efficiency of the entire computational chain. Table 3 presents quantitative improvements obtained in a real cloud architecture of an energy enterprise, where the implementation of container orchestration and adaptive load distribution mechanisms allowed the performance balance to shift toward resilience under unpredictable loads.

Table 3: Quantitative effects of cloud-platform optimization for a power-enterprise architecture (Compiled by the author based on the source: [10])

Indicator	**Scenario**	**Type of change**
Task response/completion time	Normal load	↓ ~50%
Throughput	High load	↑ ~78%
Response latency	High load	↓ >57%
Service interruptions	High load	↓ ~75%
Recovery time	Failures	↓ ~67%
Task loss rate	Failures	↓ ~79%
Rescheduling success	Failures	↑ >50%
Energy consumption	Cost metrics	↓ ~32%
Maintenance cost	Cost metrics	↓ >54%
Total monthly cost	Cost metrics	↓ ~43%
CPU/memory utilization	All modes	Significant increase

The data obtained allow for the assertion that the influence of cloud optimizations is systemic in nature. Improvements are recorded both under normal load and during peak states and failures. This distribution of effects indicates that containerization and autoscaling reduce the significance of local node performance drops, compensating for them with increased resource availability. These results align with the conclusions of Gkamas and his colleagues [4], where the role of container environments in stabilizing the behavior of distributed DBMSs for industrial IoT is emphasized.

Of particular importance is the reduction in costs, which demonstrates the intersection of technical and economic effects. In the context of analytical systems described by Koukaras [6], economic parameters are viewed as an integral part of architectural optimization, and the results [10] confirm this connection with a practical example. The increase in CPU and memory utilization is in logical accordance with what Sousa and his colleagues [8] call the advantage of hybrid integration platforms. Resources do not sit idle awaiting loads but are redistributed for the task.Consequently, cloud optimizations exert a complex impact. They simultaneously increase throughput, resilience, economic efficiency, and infrastructure adaptability. This confirms that when designing high-load systems, it is necessary to consider optimization at the level of the DBMS and orchestration, infrastructure, and inter-service interactions.

For validation of the applicability of the examined strategies, the author implemented part of the described mechanisms in a regional web platform for local self-government bodies. The system included more than 280 municipal units, most of which previously lacked any digital infrastructure. After migrating to PostgreSQL with a Redis layer for caching frequently requested documents, the initial page rendering time decreased from 1.8–2.4 seconds to 0.7–0.9 seconds, and the storage load dropped by 52% during peak request hours. The introduction of a distributed request-processing queue reduced the probability of failures during mass publication of new documents, which aligns with the conclusions on the impact of infrastructural optimizations presented by Yin and his colleagues [10].

The analysis of high-load architectures showed that PostgreSQL and Redis optimization mechanisms affect different levels of the computational circuit and react differently to the growth of query complexity, load distribution, and the variability of access profiles. In the case of PostgreSQL, the key element becomes the scheduler's ability to adequately interpret the query structure and data behavior. The study by Wehrstein and his colleagues [9] demonstrated that the optimizer loses efficiency when working with pronounced correlations and non-standard predicates, which points to a fundamental problem—the dependence of execution quality on the correctness of early statistical estimates. This conclusion aligns well with observations secured in the instrumental work of Choi and his colleagues [3], where plan selection errors are visualized and the optimizer's sensitivity to schema details and cardinalities is shown. Collectively, such results allow PostgreSQL to be viewed as a system in which further improvements will inevitably be linked to the development of statistical models, the expansion of the benchmark repertoire, and the increase of scheduler resilience to data structure deviations.

Unlike PostgreSQL, Redis ensures optimization at the level of minimizing access latencies to hot data. Its efficiency is manifested primarily in scenarios where response time consistency is critical, regardless of load dynamics. The review by Salunke & Ouda [7] shows that the in-memory architecture provides an advantage in profiles requiring high read and update frequency. This property is enhanced through architectural techniques—horizontal sharding, replication, and load distribution between nodes, which is emphasized in the integration model of Sousa and his colleagues [8], where Redis is viewed as a stable acceleration layer for corporate systems. Unlike relational storage systems dependent on query structure, Redis demonstrates resilience to operation variability, making it a functionally effective solution in multi-layer contexts with a high intensity of short transactions.

A significant role in interpreting optimization mechanisms is played by the cloud level. Infrastructure improvements recorded in the study by Yin and his colleagues [10] show that DBMS efficiency is formed by internal algorithms and platform behavior: containerization, autoscaling, load redistribution, and failure handling. These data allow the DBMS to be viewed as a component of a larger ecosystem, where the interaction of computing nodes, the network layer, and distributed execution mechanisms becomes strategically important.Thus, the comparison of PostgreSQL and Redis strategies allows for the conclusion that neither technology is a universal means of increasing the performance of high-load systems. PostgreSQL requires the improvement of planning models and statistical analysis, while Redis strengthens the cache-oriented circuit and reduces latencies during intensive access. Their joint application makes it possible to form balanced architectures in which optimization is distributed among the functional levels of the system.

5.Conclusion

The conducted analysis showed that the optimization of high-load systems ceases to be a task of local tuning of individual DBMSs and transforms into a multi-level engineering discipline. Performance is determined not by the speed of individual operation execution, but by how coherently query schedulers, caching mechanisms, network components, and cloud infrastructure interact. This allows DBMSs to be viewed not as isolated computing nodes, but as elements of a broader architecture where value is created at the intersection of storage, orchestrators, and service layers.

The research confirmed that relational and in-memory solutions are characterized by different efficiency models. PostgreSQL shows maximum sensitivity to the quality of statistics, schema structure, and cardinality predictability, whereas Redis ensures response stability under conditions of variable load and minimizes latencies during intensive access to hot data. This difference emphasizes the need to design data processing architecture as a system of distributed roles, rather than as competing technologies.

An important result was the demonstration that the cloud level not only enhances optimization effects but also sets their limits. Containerization, scaling, load redistribution, and failure handling form the conditions in which the strengths and weaknesses of each DBMS are manifested. The platform architecture becomes not a background, but an active component of performance, influencing the resilience, predictability, and energy efficiency of computations.

The applied value of the research lies in identifying the principle of consistency. High-load systems reach maturity when optimizations are aligned in a single line, from internal DBMS algorithms to resource planning mechanisms and service peak processing. A set of individual methods does not yield an effect if they are not embedded in the general execution context.

Thus, performance optimization is formed as a coherent architectural practice, in which individual techniques are less significant than their internal consistency, reproducibility, and ability to maintain stable system behavior under load. It is precisely this systemic nature that transforms working solutions into sustainable ones and allows for the design of infrastructures oriented toward predictability, reliability, and long-term development.

References

[1]. Calagna, A., Ravera, S., & Chiasserini, C. F. (2025). Enabling efficient collection and usage of network performance metrics at the edge. Computer Networks, 262, 111158. https://doi.org/10.1016/j.comnet.2025.111158

[2]. Camilleri, C., Vella, J. G., & Nezval, V. (2024). Horizontally scalable implementation of a distributed DBMS delivering causal consistency via the actor model. Electronics, 13(17), 3367. https://doi.org/10.3390/electronics13173367

[3]. Choi, Y., Han, J., Koo, K., & Moon, B. (2024). Jovis: A visualization tool for PostgreSQL query optimizer [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2411.14788

[4]. Gkamas, T., Karaiskos, V., & Kontogiannis, S. (2022). Performance evaluation of distributed database

strategies using Docker as a service for industrial IoT data: Application to Industry 4.0. Information, 13(4), 190. https://doi.org/10.3390/info13040190

[5]. Kalay, M. U. (2025). Concurrency challenges in database systems: A focus on PostgreSQL. Bibiltek: Journal of Library and Information Science, 6(1), 1–16. https://doi.org/10.54047/bibted.1574178

[6]. Koukaras, P. (2025). Data integration and storage strategies in heterogeneous analytical systems: Architectures, methods, and interoperability challenges. Information, 16(11), 932. https://doi.org/10.3390/info16110932

[7]. Salunke, S. V., & Ouda, A. (2024). A performance benchmark for the PostgreSQL and MySQL databases. Future Internet, 16(10), 382. https://doi.org/10.3390/fi16100382

[8]. Sousa, R., Abelha, V., Peixoto, H., & Machado, J. (2024). Unlocking healthcare data potential: A comprehensive integration approach with GraphQL, openEHR, Redis, and pervasive business intelligence. Technologies, 12(12), 265. https://doi.org/10.3390/technologies12120265

[9]. Wehrstein, J., Eckmann, T., Heinrich, R., & Binnig, C. (2025). JOB-Complex: A challenging benchmark for traditional & learned query optimization [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2507.07471

[10]. Yin, X., Zhang, X., Pei, L., & others. (2025). Optimization and benefit evaluation model of a cloud computing-based platform for power enterprises. Scientific Reports, 15, 26366. https://doi.org/10.1038/s41598-025-10314-5