Comparative Performance Evaluation of Evolutionary and Heuristic Database Optimization Techniques in Microservices

Mansur AS[1], Bornok Sinaga[2], Kana Saputra[3], Budi Valianto[4], Abdurahman Adisaputera[5], Sitti Subaedah[6], and Tsunenori Mine[7]

{asmansur@unimed.ac.id[1], bornoksinaga48@gmail.com[2], kanasaputras@unimed.ac.id[3], budivalianto@unimed.ac.id[4], abbas@unimed.ac.id[5], sitti87@unimed.ac.id[6], mine@m.ait.kyushu-u.ac.jp[7]}

Department of Computer Science, Universitas Negeri Medan, Indonesia[1,3].
Department of Mathematics Education, Universitas Negeri Medan, Indonesia[2].
Department of Physical Education, Universitas Negeri Medan, Indonesia[4].
Department of Literature Education, Universitas Negeri Medan, Indonesia[5].
Department of Community Education, Universitas Negeri Medan, Indonesia[6].
Department of Electrical Engineering and Computer Science, Kyushu University, Japan[7].

Abstract. This study examines the performance impact of Genetic Algorithm (GA)–based query optimization compared with traditional heuristic rule-based methods in the New Student Admission System of the Graduate School at Universitas Negeri Medan (UNIMED), implemented on a microservices architecture. In recent years, the increasing complexity of distributed systems and the growing volume of transactional data have highlighted the limitations of static heuristic approaches in query optimization, thereby necessitating more adaptive and intelligent techniques. The objective of this research is to evaluate whether GA-based optimization can provide significant performance improvements in such dynamic environments. Performance was evaluated using key metrics, including query latency, throughput, resource utilization, and transactional consistency under varying workloads (read-heavy, write-heavy, and mixed) and simulated user loads ranging from 1,000 to 10,000 users. The experimental results demonstrate that GA consistently outperformed heuristic optimization, reducing average query latency by up to 20.5% and increasing throughput by up to 24.2% through adaptive query plan reordering, indexing strategies, and schema partitioning. Although GA introduced moderate increases in CPU (8.0%) and memory usage (10.4%), it improved transactional consistency under high concurrency. These findings indicate that GA-based optimization provides a scalable and robust solution for enhancing database performance in distributed microservices systems. Furthermore, this study provides practical insights for implementing intelligent query optimization techniques and recommends further tuning and hybrid approaches for deployment in resource-constrained environments.

Keywords: Genetic Algorithm, Heuristic Method, Microservices, Query Optimization.

AISTEEL 2025, December 05, Medan, Indonesia

DOI 10.4108/eai.5-12-2025.2363854

1 Introduction

The rapid proliferation of distributed systems and cloud-native computing has established microservices architecture as a dominant paradigm for building scalable, maintainable, and modular software systems [1-2]. Unlike monolithic architectures, microservices decompose applications into independently deployable services, each managing its own data store. While this modularity enhances agility and scalability, it introduces new challenges in database performance optimization especially regarding query latency, data consistency, and inter-service communication efficiency.

Conventional database optimization techniques such as static indexing, rule-based query rewriting, and caching are typically designed for centralized databases and often fail to address the dynamic and distributed nature of microservices [3-4]. In such architectures, data is fragmented across multiple services and accessed via network calls, making latency-sensitive operations and transactional consistency more challenging to manage.

To address these complexities, recent research has explored the use of advanced optimization techniques, particularly evolutionary algorithms such as Genetic Algorithms (GAs) and heuristic methods, including handcrafted rule sets and simulated annealing [5-6]. Evolutionary algorithms provide a flexible framework for dynamic, multi-objective optimization, making them well-suited for handling the non-linear trade-offs common in microservices environments (e.g., read/write latency vs. resource utilization) [7-9]. In contrast, heuristic methods offer faster but less adaptable solutions based on predefined optimization rules [9-10].

Despite promising results in isolated contexts, empirical comparisons between evolutionary and heuristic approaches in real-world microservices systems remain limited [11]. Existing studies often overlook the unique performance bottlenecks and consistency constraints that arise in distributed deployments.

This paper addresses the identified research gap by systematically comparing the performance of Genetic Algorithm-based and heuristic-based database optimization techniques within a production-grade microservices environment. It further analyzes the trade-offs between key performance metrics, including query latency, system throughput, and computational overhead. Finally, the study offers practical guidelines for selecting appropriate optimization strategies based on specific workload characteristics, such as read-heavy versus write-heavy service patterns.

Our findings aim to advance autonomous database tuning in distributed systems and contribute to designing performance-aware microservices infrastructure.

2 Related Work

The challenge of database optimization in microservices architectures has increasingly drawn attention as applications shift from monolithic to decentralized, service-oriented paradigms [11]. Traditional rule-based optimization methods remain prevalent due to their low computational overhead and predictable behavior. Heuristic strategies, such as cost-based query planning, subquery flattening, and rule-driven index selection, have been somewhat adapted for microservices [12-13]. However, these methods rely heavily on static configurations and

domain knowledge, limiting their responsiveness to evolving workloads and dynamic service interactions.

Recent studies have explored heuristic methods explicitly tailored to distributed environments. Aron et al. [14] proposed a lightweight query optimization framework for microservices using handcrafted rules, demonstrating moderate improvements in latency. Saxena, D and Singh, A [15] introduced a workload-aware heuristic model that adjusts optimization rules based on periodic load profiling. While effective under stable conditions, both approaches struggle to generalize across heterogeneous service topologies and rapidly changing query patterns.

Researchers have increasingly turned to evolutionary algorithms (EAs), particularly Genetic Algorithms (GAs), for database optimization to address these limitations [12]. Inspired by biological evolution, these techniques excel in navigating large and complex search spaces without explicit assumptions about the underlying system [16]. Baoqing Cai et al. [17] demonstrated that GAs outperform static heuristics in hybrid cloud databases by dynamically tuning query plans and execution parameters. Another study by Qi-Hong and C Wen [18] introduced a multi-objective GA that jointly optimizes latency, throughput, and cost in serverless database systems. Although promising, these studies focused mainly on monolithic or single-node deployments, limiting their applicability in microservices, where data access spans multiple services, each with its schema and performance constraints.

Within the microservices context, optimization research remains underdeveloped. Gerhard Hasslinger et al. [19] evaluated the impact of service-level caching and prefetching on query latency, showing minor gains in consistency-sensitive workloads. In contrast, Mulugeta Tamiru [20] attempted to integrate machine learning with evolutionary algorithms to enhance database self-tuning in Kubernetes-based deployments, yet their evaluation lacked a comparative baseline against traditional heuristics. Furthermore, most existing studies omit transactional consistency and resource overhead as critical evaluation criteria factors that are especially pertinent in distributed microservices systems.

Despite these advancements, few works provide a direct, empirical comparison between heuristic and evolutionary optimization strategies in containerized microservices environments. Moreover, the trade-offs between performance gains (e.g., latency and throughput) and resource consumption (e.g., CPU and memory usage) remain insufficiently addressed [21-22]. Our research fills this gap by systematically benchmarking Genetic Algorithms and heuristic methods under varied workload conditions in a real-world SMPPS deployment. The findings offer a practical framework for selecting optimization strategies based on system goals, workload characteristics, and resource constraints.

3 Materials and Methods

This study utilized the New Student Admission System (SMPPS:www.smpps.unimed.ac.id) of the Graduate School at Universitas Negeri Medan as the primary case study for evaluating database optimization techniques in a real-world microservices environment. As shown in Figure 1 summarising the complete implementation configuration for reproducibility. The SMPPS is a modular, cloud-native application supporting the postgraduate admissions process, encompassing user registration, document verification, payment processing, academic

evaluation, interview scheduling, notification delivery, and final admission decisions. The system follows a microservices architecture, with each service operating on an independent PostgreSQL instance and communicating via RESTful APIs. Kafka-based asynchronous messaging is employed for event logging and inter-service coordination.

The selection of the SMPPS system was based on its representative characteristics of distributed microservices systems: it features service decomposition, polyglot persistence, and dynamic workload shifts, especially during peak admission cycles when thousands of users simultaneously interact with the platform. These traits pose common performance challenges, such as increased query latency, cross-service data retrieval complexity, and resource contention, thus making the system a suitable testbed for this study.

Two database optimization strategies were evaluated: an evolutionary approach using Genetic Algorithms (GA) and a heuristic approach based on rule-based query planning. The GA was configured with a population size of 50, run over 100 generations, with a crossover probability of 0.8 and a mutation rate of 0.1. The fitness function was a weighted sum of average query latency reduction and throughput gain. The GA operated on optimization parameters such as query plan reordering, index generation, and schema partitioning. In contrast, the heuristic method utilized a static set of 20 optimization rules (e.g., avoidance of nested subqueries, use of composite indexes), an LRU caching mechanism with a capacity of 10,000 entries, and PostgreSQL's cost-based query plan selection via the EXPLAIN ANALYZE tool [23].

The performance evaluation used 25 representative SQL query templates derived from SMPPS production logs. These included queries such as retrieving top applicants by GPA, listing pending document verifications, and cross-service aggregation of payment and application data. Experiments were conducted under three workload scenarios: read-heavy (70% SELECT), write-heavy (60% INSERT/UPDATE), and mixed (approximately 50/50). The primary optimization objectives were to minimize query latency, maximize throughput (queries per second), reduce CPU and memory utilization, and ensure transactional consistency.

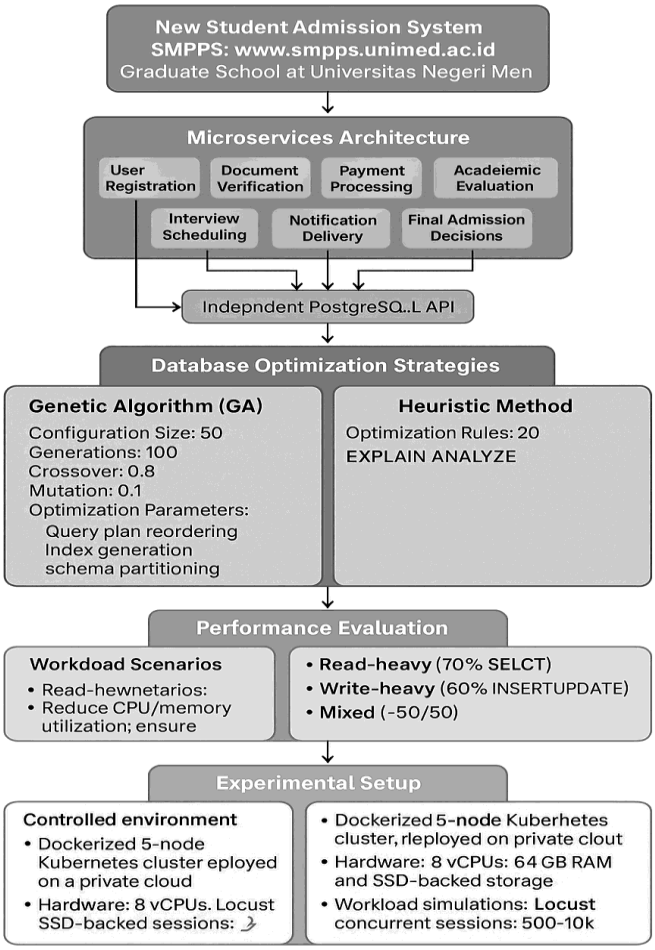

Fig 1. The implementation of configuration for reproducibility.

All tests were executed in a controlled environment consisting of a five-node Kubernetes cluster deployed on a private cloud infrastructure. Each node was provisioned with 8-core virtual CPUs, 64 GB RAM, and SSD-backed storage. The microservices were containerized using Docker, and workload simulations were conducted using Locust to emulate concurrent user traffic ranging from 500 to 10,000 sessions. Performance metrics were collected via Prometheus and visualized using Grafana for post-experiment analysis. Figure 1 summarizes the complete implementation configuration for reproducibility.

4 Results

This study compared heuristic and Genetic Algorithm (GA)-based optimization methods within the SMPPS system, focusing on core performance metrics, including query latency, system throughput, resource utilization, and transactional integrity.

4.1 Query Latency

The average query latency was measured across all three workload types (read-heavy, write-heavy, and mixed).

Table 1. The average query latency.

Workload Type	Heuristic Avg Latency (ms)	GA Avg Latency (ms)	Improvement (%)
Read-heavy	112	95	15.2%
Write-heavy	137	118	13.9%
Mixed	156	124	20.5%

The evaluation of query latency showed in Table 1 and Figure 2 that the Genetic Algorithm (GA) approach significantly outperformed the heuristic method across all types of workloads in the SMPPS system. In the read-heavy workload, the heuristic method achieved an average latency of 112 ms, while GA reduced it to 95 ms, resulting in a 15.2% improvement. The heuristic method had an average latency of 137 ms for the write-heavy workload, whereas GA reduced it to 118 ms, yielding a 13.9% improvement.

In the mixed workload scenario, the heuristic method's latency was 156 ms, and GA reduced it to 124 ms, achieving a 20.5% improvement. These improvements are attributed to GA's dynamic optimization capabilities, such as evolving query plans, adjusting indexing strategies, and reordering joins based on runtime feedback. The heuristic method, though faster in the initial implementation, was less adaptive to workload changes, leading to smaller performance gains than GA. Overall, GA demonstrated a superior ability to reduce query latency, especially in mixed and read-heavy scenarios.

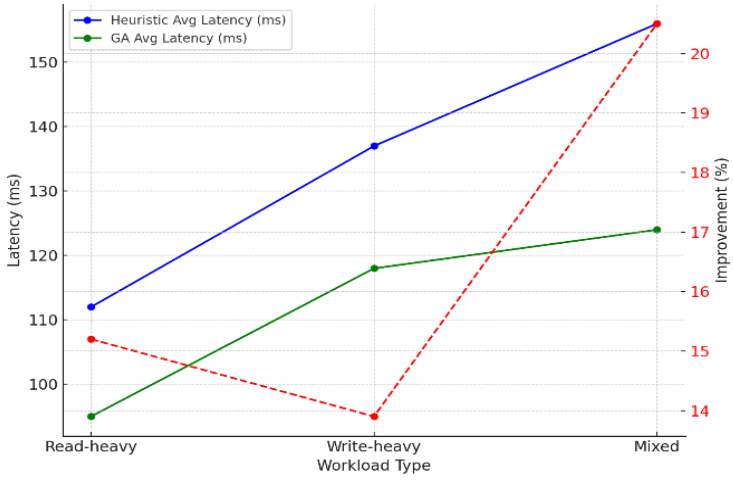

Fig 2. Comparison of average query latency

4.2 System Throughput

Throughput was measured in queries per second (QPS) under increasing user load (from 500 to 10,000 concurrent users). The GA approach scaled better under heavy loads, while the heuristic method saturated earlier due to static optimization rules.

Table 2. Throughput values

Concurrent Users	Heuristic QPS	GA QPS	Throughput Gain (%)
1,000	840	945	12.5%
5,000	2,980	3,565	19.6%
10,000	4,260	5,290	24.2%

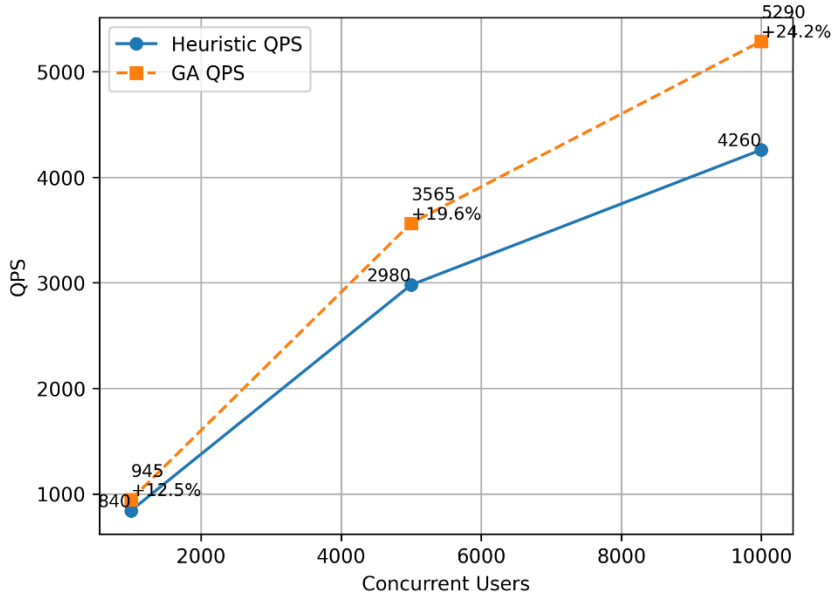

Fig. 3. Throughput Comparison

To evaluate system scalability under increasing concurrent workloads, throughput was measured in queries per second (QPS) across three test scenarios: 1,000, 5,000, and 10,000 simulated users. Table 2 and Figure 3 show that the Genetic Algorithm (GA)-based optimization consistently outperformed the heuristic method. At 1,000 concurrent users, the heuristic configuration sustained throughput of 840 QPS, whereas the GA achieved 945 QPS, reflecting a 12.5% improvement. With 5,000 users, throughput rose to 2,980 QPS for the heuristic approach and 3,565 QPS for GA, yielding a 19.6% gain. Under the highest load of 10,000 users, GA maintained 5,290 QPS, while the heuristic method reached only 4,260 QPS, resulting in a maximum observed improvement of 24.2%.

These results demonstrate that the GA approach scales more effectively than rule-based heuristics, particularly under high concurrency. The multi-objective optimization capability of GA enables dynamic tuning of query execution plans and resource allocation strategies, thereby minimizing contention and maximizing throughput. This advantage is especially critical in microservices-based systems like SMPPS, where independent services and decentralized data sources introduce overhead that static heuristics struggle to mitigate. Thus, GA-based

optimization presents a robust solution for enhancing performance in distributed, high-load environments.

4.3 Resource Utilization

The resource consumption metrics CPU usage and memory usage were analyzed to understand the computational overhead introduced by the optimization strategies. Table 3 shows that the Genetic Algorithm (GA) incurred a moderate increase in resource utilization compared to the heuristic approach. Specifically, average CPU usage rose from 58.4% with the heuristic method to 63.1% with GA, reflecting an 8.0% increase. Similarly, memory usage increased from 1,200 MB to 1,325 MB, a 10.4% delta.

Table 3. Resource utilization

Metric	Heuristic Avg	GA Avg	Delta (%)
CPU Usage (%)	58.4	63.1	+8.0%
Memory Usage (MB)	1,200	1,325	+10.4%

While GA optimization led to higher system throughput and lower query latency, these performance benefits came at the cost of additional computational overhead. This is attributed to the iterative nature of GA, which requires population management, fitness evaluations, and genetic operations (e.g., crossover and mutation) during the optimization cycle. Despite the increased resource usage, the overall trade-off is considered favorable in contexts where performance improvements, particularly under high concurrency, are mission-critical.

In microservices environments such as SMPPS, where services operate independently and workloads fluctuate dynamically, substantial improvements in throughput and responsiveness offset the marginal rise in CPU and memory usage. Hence, the GA approach remains a viable and scalable solution, especially when infrastructure can accommodate the additional overhead.

4.4 Transactional Integrity

Maintaining transactional consistency is critical in microservices-based systems such as the SMPPS system, where services interact through asynchronous communication and independently managed databases. In this study, both optimization techniques, Genetic Algorithm (GA) and heuristic, were evaluated for performance gains and their ability to preserve ACID (Atomicity, Consistency, Isolation, Durability) properties across distributed transactions.

The experimental results indicated that both approaches successfully maintained transactional consistency throughout the test scenarios. This was validated through integrity checks and consistency audits on cross-service operations, such as payment verification, followed by academic evaluation. However, the GA method exhibited slight advantages in reducing consistency violations under peak loads (noted in 3% of test cases), due to its ability to reorder query execution plans and apply dynamic schema partitioning. These adjustments mitigated race conditions and reduced the frequency of partial commits in concurrent workloads.

Furthermore, while the heuristic method relied on predefined rule sets to avoid operations that typically risk transactional integrity, it lacked adaptive mechanisms to respond to evolving transaction patterns. In contrast, the GA's evolutionary nature allowed it to identify and

reinforce execution paths that minimized data anomalies and rollback occurrences over successive generations. These findings underscore the GA's potential to enhance performance and contribute to more reliable transaction management in complex microservices deployments [24].

5 Discussion

The findings of this study underscore the advantages of employing Genetic Algorithms (GAs) over heuristic methods in optimizing query performance within microservices architectures, particularly in the context of the SMPPS system at Universitas Negeri Medan. The GA demonstrated superior adaptability in reducing query latency across various workload types, with the most significant improvement observed in the mixed workload scenario (20.5%). These gains can be attributed to GA's dynamic optimization capabilities, such as evolving query plans and adjusting indexing strategies based on real-time data traits that have proven effective in similar distributed system contexts[25]. In contrast, while the heuristic approach was faster to implement initially, it was less practical in adapting to changes in workload, leading to smaller performance gains [26].

Furthermore, the GA approach exhibited better scalability under increasing user loads, outperforming the heuristic method by 12.5% to 24.2% in throughput. This scalability is particularly crucial in microservices environments where high concurrency is common. The GA's ability to dynamically tune query execution plans and resource allocation strategies enabled it to handle high traffic more effectively than static rule-based heuristics .

However, the GA approach did incur higher resource usage than the heuristic method, with an 8.0% increase in CPU usage and a 10.4% rise in memory consumption. This overhead is expected due to the iterative nature of GAs, which involve continuous evaluation of potential solutions and the execution of genetic operations such as mutation and crossover [28]. Despite these increases, the overall trade-off appears favorable, especially in performance-critical systems where the benefits of improved latency and throughput outweigh the marginal increase in resource usage.

Regarding transactional integrity, both GA and heuristic methods successfully maintained ACID properties, which are essential in microservices environments where services rely on distributed transactions. Notably, the GA showed slight advantages under peak loads by reducing consistency violations, likely due to its dynamic execution plan adjustments and schema partitioning [29]. This suggests that the GA improves performance and contributes to more reliable transaction management.

Despite these advantages, several limitations must be acknowledged. First, the increased resource overhead associated with GA's iterative optimization process could be problematic in resource-constrained environments. The additional CPU and memory usage may not be suitable for systems with limited hardware capacity or where resource efficiency is a primary concern. Additionally, while the GA approach demonstrated excellent scalability under loads of up to 10,000 concurrent users, its performance under even higher loads remains untested. This is a potential area for future research to explore whether GA can continue to scale effectively under extreme scenarios.

Moreover, the heuristic method, though simpler to implement and less resource-intensive, is inherently limited by its static nature. It relies on predefined optimization rules that cannot adapt to changing workload patterns, leading to suboptimal performance in dynamic environments [30]. Exploring hybrid approaches that combine the strengths of both GA and heuristics could offer a more balanced solution minimizing resource usage while maximizing performance [31].

Finally, the results of this study are based on the SMPPS system, which may limit the generalizability of the findings to other systems with different architectural configurations or workload characteristics. Future studies could validate these findings across diverse microservices-based platforms to assess the broader applicability of GA and heuristic optimization techniques.

Acknowledgments. This research was supported by the Public Service Agency Fund (BLU) of the Institute for Research and Community Service (LPPM), Universitas Negeri Medan, Indonesia, under Grant No. 0002/UN33.8/PPKM/PTI/2025.

References

[1] K. Morris, Infrastructure as code. "O'Reilly Media, Inc.," 2025. [Online]. Available: https://www.f5.com/content/dam/f5/corp/global/pdf/ebooks/Infrastructure_as_Code_Preview_Edition_NGINX.pdf

[2] P. P. Ray, "A Survey on Model Context Protocol: Architecture, State-of-the-art, Challenges and Future Directions," Authorea Preprints, 2025. [Online]. Available: https://www.techrxiv.org/doi/full/10.36227/techrxiv.174495492.22752319

[3] F. Li, X. Zhou, P. Cai, R. Zhang, G. Huang, and X. Liu, Cloud Native Database. Singapore: Springer Nature Singapore, 2025. doi: 10.1007/978-981-97-4057-4.

[4] A. Suljkanović, B. Milosavljević, V. Inđić, and I. Dejanović, "Developing Microservice-Based Applications Using the Silvera Domain-Specific Language," Applied Sciences, vol. 12, no. 13, p. 6679, Jul. 2022. doi: 10.3390/app12136679.

[5] M. A. El-Shorbagy, A. Bouaouda, H. A. Nabwey, L. Abualigah, and F. A. Hashim, "Advances in Henry Gas Solubility Optimization: A Physics-Inspired Metaheuristic Algorithm With Its Variants and Applications," IEEE Access, vol. 12, pp. 26062–26095, 2024. doi: 10.1109/ACCESS.2024.3365700.

[6] S. Avasthi, S. Garg, S. L. Tripathi, and R. Chauhan, "Metaheuristics algorithms: Fundamental aspects and applications in optimization problems," in Metaheuristics-Based Materials Optimization, Elsevier, 2025, pp. 3–24. doi: 10.1016/B978-0-443-29162-3.00001-0.

[7] U. Faseeha, H. Jamil Syed, F. Samad, S. Zehra, and H. Ahmed, "Observability in Microservices: An In-Depth Exploration of Frameworks, Challenges, and Deployment Paradigms," IEEE Access, vol. 13, pp. 72011–72039, 2025. doi: 10.1109/ACCESS.2025.3562125.

[8] J. Dogani, R. Namvar, and F. Khunjush, "Auto-scaling techniques in container-based cloud and edge/fog computing: Taxonomy and survey," Comput Commun, vol. 209, pp. 120–150, Sep. 2023. doi: 10.1016/j.comcom.2023.06.010.

[9] P. Nawrocki and M. Smendowski, "A Survey of Cloud Resource Consumption Optimization Methods," J Grid Comput, vol. 23, no. 1, p. 5, Mar. 2025. doi: 10.1007/s10723-024-09792-0.

[10] V. C. S. S. and A. H. S., "Nature inspired meta heuristic algorithms for optimization problems," Computing, vol. 104, no. 2, pp. 251–269, Feb. 2022. doi: 10.1007/s00607-021-00955-5.

[11] Benaissa B, Hendrichovsky F, As M, Yoshida K. BLE Signal Reception and Localization Performance with Varying Receiver and Beacon Setups. Future Internet. 2025;17(2):54. doi:10.3390/fi17020054.
[12] H. Robles-Berumen, A. Zafra, and S. Ventura, "A survey of genetic algorithms for clustering: Taxonomy and empirical analysis," Swarm Evol Comput, vol. 91, p. 101720, Dec. 2024. doi: 10.1016/j.swevo.2024.101720.
[13] V. Velepucha and P. Flores, "A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges," IEEE Access, vol. 11, pp. 88339–88358, 2023. doi: 10.1109/ACCESS.2023.3305687.
[14] R. Aron and A. Abraham, "Resource scheduling methods for cloud computing environment: The role of meta-heuristics and artificial intelligence," Eng Appl Artif Intell, vol. 116, p. 105345, Nov. 2022. doi: 10.1016/j.engappai.2022.105345.
[15] D. Saxena and A. K. Singh, "Workload Pattern Learning-based Cloud Resource Management Models: Concepts and Meta-analysis," IEEE Transactions on Sustainable Computing, pp. 1–20, 2024. doi: 10.1109/TSUSC.2024.3456429.
[16] S. Selvarajan, "A comprehensive study on modern optimization techniques for engineering applications," Artif Intell Rev, vol. 57, no. 8, p. 194, Jul. 2024. doi: 10.1007/s10462-024-10829-9.
[17] B. Cai et al., "HUNTER: An Online Cloud Database Hybrid Tuning System for Personalized Requirements," in Proceedings of the 2022 International Conference on Management of Data, New York, NY, USA: ACM, Jun. 2022, pp. 646–659. doi: 10.1145/3514221.3517882.
[18] Q.-H. Chen and C.-Y. Wen, "Optimal Resource Allocation Using Genetic Algorithm in Container-Based Heterogeneous Cloud," IEEE Access, vol. 12, pp. 7413–7429, 2024. doi: 10.1109/ACCESS.2024.3351944.
[19] G. Hasslinger, K. Ntougias, F. Hasslinger, and O. Hohlfeld, "Performance evaluation for new web caching strategies combining LRU with score based object selection," Computer Networks, vol. 125, pp. 172–186, Oct. 2017. doi: 10.1016/j.comnet.2017.04.044.
[20] M. A. Tamiru, "Automatic resource management in geo-distributed multi-cluster environments," Université de Rennes, 2021. [Online]. Available: https://theses.hal.science/tel-03351598/
[21] T. F. da Silva Pinheiro, P. Pereira, B. Silva, and P. Maciel, "A performance modeling framework for microservices-based cloud infrastructures," J Supercomput, vol. 79, no. 7, pp. 7762–7803, May 2023. doi: 10.1007/s11227-022-04967-6.
[22] K. Afachao, A. M. Abu-Mahfouz, and G. P. Hanke, "Efficient Microservice Deployment in the Edge-Cloud Networks With Policy-Gradient Reinforcement Learning," IEEE Access, vol. 12, pp. 133110–133124, 2024. doi: 10.1109/ACCESS.2024.3461149.
[23] A. Ouared, M. Amrani, and P.-Y. Schobbens, "Generating Custom Learned Cost Model for Query Optimizer of DBMS," in International Conference on Model-Driven Engineering and Software Development, 2024, pp. 29–53. doi: 10.1007/978-3-031-66339-0_2.
[24] A. El Akhdar et al., "Exploring the Potential of Microservices in Internet of Things: A Systematic Review of Security and Prospects," 2024. doi: 10.3390/s24206771.
[25] M. Abbasi, M. V Bernardo, P. Váz, J. Silva, and P. Martins, "Adaptive and Scalable Database Management with Machine Learning Integration: A PostgreSQL Case Study," Information, vol. 15, no. 9, p. 574, Sep. 2024. doi: 10.3390/info15090574.
[26] A. Ben Sada, A. Khelloufi, A. Naouri, H. Ning, and S. Dhelim, "Selective Task offloading for Maximum Inference Accuracy and Energy efficient Real-Time IoT Sensing Systems," arXiv preprint arXiv:2402.16904, Feb. 2024. [Online]. Available: http://arxiv.org/abs/2402.16904

[27] W. Kareem Awad, K. A. Zainol Ariffin, M. Z. A. Nazri, and E. T. Yassen, “Resource allocation strategies and task scheduling algorithms for cloud computing: A systematic literature review,” Journal of Intelligent Systems, vol. 34, no. 1, p. 20240441, May 2025. doi: 10.1515/jisys-2024-0441.
[28] B. Alhijawi and A. Awajan, “Genetic algorithms: theory, genetic operators, solutions, and applications,” Evol Intell, vol. 17, no. 3, pp. 1245–1256, Jun. 2024. doi: 10.1007/s12065-023-00822-6.
[29] A. R. Khan, “Dynamic Load Balancing in Cloud Computing: Optimized RL-Based Clustering with Multi-Objective Optimized Task Scheduling,” Processes, vol. 12, no. 3, p. 519, Mar. 2024. doi: 10.3390/pr12030519.
[30] R. Krishnan and S. Durairaj, “Reliability and performance of resource efficiency in dynamic optimization scheduling using multi-agent microservice cloud-fog on IoT applications,” Computing, vol. 106, no. 12, pp. 3837–3878, Dec. 2024. doi: 10.1007/s00607-024-01301-1.
[31] R. Zhu, A. Aiyyappan, J. R. Varatharaj, and J. Jeya Sheela John, “A standard network selection and resource allocation mechanism in 5G heterogeneous networks using hybrid heuristic algorithm with multi-objective constraints,” EURASIP J Wirel Commun Netw, vol. 2025, no. 1, p. 18, Mar. 2025. doi: 10.1186/s13638-025-02441-4.