International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(21s) 2026 pp. 1976-1983
https://doi.org/10.70917/ijcisim-2026-5973
Article

Workload-Aware Shard-Key Selection and Budget-Constrained Replica Placement for Geo-Distributed Online Transaction Processing Databases

Srihari Tubati

Independent Researcher, North Carolina, USA

Abstract: A shard key chosen once at design time and never revisited is a liability, not a decision. This article makes the case that shard-key selection and replica placement both belong in the category of things a running system should reassess against real traffic, not settle in advance and forget. We frame per-table shard-key selection as a search over candidates the schema itself already exposes - a table's own primary key, plus each parent's primary key reachable through a foreign key - scored on how much co-partitioning it buys, how far cross-partition fan-out drops, and how evenly load lands across partitions. Replica placement is treated separately: a budget-constrained allocation of primary regions and read-serving copies driven by where read and write demand actually originates. Simulated annealing handles the key-selection search; a demand-weighted heuristic handles placement. Both are tested, alone and combined, against primary-key hashing, range partitioning, and a fixed customer-centered heuristic that many teams treat as a safe default (it is not, as the results below show). Every compared placement strategy draws from the same twelve-slot replica budget across eight partitions - otherwise a faster-looking strategy might just be a better-funded one. The testbed is a synthetic 24-table OLTP schema with 28 foreign-key relationships, evaluated across sixteen independently generated workload seeds with paired statistical testing and Benjamini-Hochberg correction across the full family of confirmatory comparisons. Workload-aware key selection reaches 82.7 percent of the structural locality reference, well ahead of primary-key hashing (53.8 percent) and the customer-centered heuristic (55.4 percent), and cuts cross-partition query share by 24.2 percent and average fan-out by 17.6 percent against the hash baseline. Placement alone, under the fixed budget, reduces mean latency by roughly 46 to 54 percent, P95 latency by 42 to 50 percent, and P99 latency by 50 to 54 percent across every tested shard-key strategy. Combined, the pipeline cuts modeled mean latency by 53.1 percent, median latency by 82.6 percent, P95 latency by 45.7 percent, and P99 latency by 50.5 percent against the incumbent, with model-estimated throughput up 140 percent. An eight-seed exact-optimum check on a reduced six-table subset and a 200-fold sweep of the skew-penalty weight both hold up. None of this is production telemetry - it is simulation evidence, and the article treats that distinction as load-bearing throughout rather than a footnote.
Keywords: shard-key selection, replica placement, geo-distributed databases, workload-aware automation, co-partitioning, distributed transaction coordination, OLTP database design

1. Introduction

A poorly chosen shard key does not announce itself at rollout. It shows up eighteen months later, when a table that seemed evenly distributed under test traffic turns into a hot partition under production load, and fixing it means rewriting where every row for that table physically lives. In an environment running thousands of automated deployments a day against systems that 150-plus downstream engineering teams depend on, that rewrite is never a quiet weekend task. Replica placement carries a related cost that gets less attention: where the copies of a partition sit across regions decides whether a read stays local or crosses an ocean, and that decision, too, usually gets made once, early, by intuition rather than evidence.

Copyright: © 2026 by the authors

Distributed-database practice still leans heavily on design-time defaults - hash on customer ID, spread replicas evenly "for resilience" - that were reasonable guesses before there was workload data to replace them with. Having run proof-of-concept evaluations across six distributed database platforms and led geo-distributed production systems in financial services, where every cross-partition write adds coordination latency and every remote read adds tail-latency risk, we think the guessing is no longer necessary. The workload already contains the signal the schema alone cannot provide [1].

This article treats shard-key selection and replica placement as workload-aware optimization problems rather than one-time schema choices, and it organizes the investigation around four questions. Can per-table selection among real, schema-derived candidate keys beat primary-key hashing and a fixed customer-centered heuristic at improving co-partitioning and reducing fan-out? Under an identical total replica-copy budget, does workload-aware placement of primary regions and replicas cut remote-access and tail-latency costs relative to workload-agnostic placement? What does each mechanism contribute alone, and what changes when they run together? And how much of the conclusion survives independent workload seeds, an exact small-instance check, a parameter sweep, and correction for testing many comparisons at once?

What follows builds a problem formulation grounded in schema structure and workload telemetry, describes an optimization architecture centered on simulated annealing with built-in replay, lays out an evaluation spanning sixteen workload seeds on a 24-table schema, and closes by being explicit about where the simulation's authority ends and a live deployment's would need to begin [22].

2. Materials and Methods

2.1 Related Work: Where the Literature Splits

Workload-driven partitioning is not a new idea. Schism [1] established that transaction graphs, not fixed schema rules, could drive partitioning decisions, while Megastore [2] and Spanner [3] built globally consistent geo-distributed storage on placement and partitioning choices that were largely fixed at design time. CockroachDB [4] and Aurora PostgreSQL [5] carried this into commercial geo-distributed SQL - CockroachDB in particular exposes replica placement as a tunable primitive, though one that in practice most operators configure once and revisit rarely.

A newer strand automates the partitioning side specifically. Grep [6] applies graph learning to partitioning decisions and reports a throughput gain in a real banking deployment. WorthyPar [7] applies deep reinforcement learning to hybrid partitioning. DSGA [8] runs a distributed genetic algorithm over outsourced partitioning problems. None of these treat geographic replica placement as part of the same search.

A separate strand goes after tail latency directly, largely without touching partitioning at all. Bonspiel [9] targets low tail latency specifically in geo-distributed transactions; GeoTP [10] addresses latency-aware transaction routing at the middleware layer; RaaS [11] attacks tail latency in storage-disaggregated systems; TailGuard [12] targets SLO-bounded scheduling for user-facing services. Reinforcement-learning-based index tuning under a fixed budget [13], ML-powered index tuning more broadly [14], and LLM-assisted system tuning [15] establish, in an adjacent problem space, that budget constraints paired with workload evidence outperform unconstrained or workload-blind tuning - a pattern this article expects, and finds, to hold for partitioning and placement as well.

What none of these threads do is hold shard-key selection and geographic replica placement inside one evaluation, under one resource budget, against the same workload seeds. Placement studies tend to assume the partitioning is fixed; partitioning studies tend to treat replication as an afterthought. A recent cloud-native database survey [16] and a broad review of metaheuristic search [17] both flag this as an open seam rather than settled ground. That seam - not a gap in any single cited paper, but a gap in how the two problems are studied together - is what this article is built to close.

2.2 Problem Formulation

Candidate shard keys are pulled straight from schema structure: a table's own primary key, plus the primary keys of any parent table reachable through a foreign-key relationship. Two tables are co-partitioned when they share the same logical key domain and a query supplies a value resolving to the matching partition on both sides. That requires a canonical data type, hash function, seed, and partition count shared across the pair - a detail easy to gloss over and expensive to get wrong. The simulation guarantees it by construction rather than assuming it.

A structural locality reference, built purely from schema and query templates, sets a deliberately loose upper bound on how much of the workload could be local under ideal key choices. It is not a promise that every query can be made local at once. Several query templates compete for the same table's key assignment, and a choice that helps one can quietly hurt another.

Non-local queries get split three ways rather than lumped into one worst case, as summarized in Table 1. A fully co-partitioned query touches exactly one partition. A query with two independently known lookup values touches exactly two. An unscoped query touches every partition - and that only happens when at least one table in the query resolves through neither a known nor an inferable key. Collapsing these into a single category would have overstated the cost of imperfect locality. It would also have understated how much value partial locality still delivers.

Query Category	Partitions Touched	Cost Classification
Fully co-partitioned	1	Single-partition, no coordination
Two known lookup anchors	2	Bounded two-participant transaction
Unscoped	All (8)	Full-fan-out, all-partition operation

Table 1. Fan-out classification and cost model summary.

2.3 Optimization Architecture

Candidate keys come from static schema analysis. Everything else - query frequency, read/write mix, row counts, request-origin region, equality-filtered anchor keys - is pulled from workload telemetry: query logs, query store data, connection-proxy traces. When anchor inference is unreliable, the system falls back to a conservative "unresolved" label instead of guessing a query is local. That choice costs some optimism. It buys a locality estimate that will not embarrass anyone later.

Before any physical key change ships, the recommended assignment is replayed against captured workload history and scored on projected co-partitioning, fan-out, skew, latency contribution, and rebalancing cost. This is not a reporting nicety bolted on afterward (we treated it as a hard precondition from the start of the design). A shard-key change is expensive to reverse once applied in a live system, and no recommendation gets trusted enough to act on without first showing its work against real history.

The optimizer runs simulated annealing over the combinatorial space of per-table key assignments - a well-established approach for search problems of this shape [17]. The objective weights cross-partition writes above cross-partition reads, penalizes uneven partition load, and penalizes projected capacity overflow. The bounded fan-out classification from Section 2.2 feeds straight into that objective, so a two-participant transaction scores better than an unscoped, all-partition operation. Three baselines anchor the comparison: primary-key hash partitioning, range partitioning on the same primary-key choice, and a fixed customer-centered heuristic representative of what many teams do by default.

On a reduced six-table subset with forty-eight possible key assignments, exhaustive enumeration is small enough to run outright, giving a checkable optimum. Across eight independent validation seeds, simulated annealing lands on that exact optimum every single time. That is evidence about search quality at small scale, and only at small scale - it says nothing about whether the full twenty-four-table search reaches a global optimum, and we do not claim it does [21].

2.4 Replica Placement and Evaluation Design

Replica placement is decided after the shard key, not alongside it. Each physical partition gets a primary region, and depending on demand across the full regional workload distribution, may pick up additional read-serving replicas. The placement algorithm weighs remote read demand, write-origin demand, and the primary region's own latency effect together - three separate signals, not one blended popularity score standing in for all of them.

Every strategy compared draws from the identical budget: twelve total copy slots spread across eight physical partitions. This single constraint is what keeps a placement strategy from looking faster purely because it was handed more replicas - an easy comparison bias to miss if the budgets are not pinned down first. Extra copies are modeled strictly as read-serving replicas. Consensus quorum membership, leader election, durability latency, and failover are outside the model's scope, and nothing in these results should be read as a claim about them.

The evaluation schema carries 24 tables and 28 directed foreign-key relationships, held fixed across seeds so comparisons stay apples-to-apples. Sixteen independently generated workload seeds each carry 150 query templates, a frequency-weighted write share near 22 percent, three geographic regions, eight physical partitions, and skewed query popularity (skew here meaning a small number of hot query templates account for a disproportionate share of traffic, as is typical in production OLTP). Roughly a third of multi-table templates carry a second independent lookup anchor, which is what lets the evaluation tell bounded two-participant queries apart from unscoped ones instead of treating them as the same thing. Per seed, 20,000 query instances are sampled for latency analysis, and the same seed-specific workload runs against every compared strategy. This paired design controls for workload difficulty directly, rather than hoping different workload draws happen to be comparable.

Statistical analysis uses paired differences with t-distribution-based confidence intervals across the sixteen seeds, and a p-value for every confirmatory comparison. Fifty-one confirmatory tests are treated as a single family and corrected with the Benjamini-Hochberg procedure [18]; nothing is called significant on the strength of an uncorrected threshold. Null results are reported alongside the positive ones, deliberately. The fixed customer-centered heuristic is not claimed to beat plain hashing on overall locality where the corrected comparison does not clear significance. The network-transfer reduction attributable to placement alone, isolated from key selection, is reported non-significant where that is what the corrected test actually shows. The goal is attributing which mechanism drives which outcome - not manufacturing a positive result out of every comparison.

Reproducibility artifacts cover the schema generator, workload parameters, seed range, optimizer settings, metric definitions, analysis scripts, and software versions. One friction point is worth recording plainly: a Python string-hash randomization behavior initially produced non-reproducible artifact ordering across separate process runs (this surfaced during internal review, not before it - a genuine catch, not a designed-in demonstration). Once identified, the fix was confirmed by verifying byte-identical output across independent process executions. The full sixteen-seed evaluation runs on a single standard cloud processor core. No specialized or accelerated hardware was involved.

A sensitivity check sweeps the skew-penalty weight across a 200-fold range. The advantage of workload-aware key selection holds across that entire range. It is a partial check, not a complete one - the capacity-overflow weight and the relative weighting of writes against reads in the objective are not swept here, and both remain open questions.

3. Results

3.1 Shard-Key Effects, Placement Held Workload-Agnostic

With replica placement fixed and workload-agnostic across every key-selection strategy, workload-aware selection reaches 82.7 percent of the structural locality reference. Primary-key hashing reaches 53.8 percent; the fixed customer-centered heuristic reaches 55.4 percent - close to hashing, not meaningfully better (Figure 1). Against the hash baseline, the full workload-aware pipeline cuts cross-partition query share by 24.2 percent, distributed-transaction share by 23.1 percent, and average fan-out by 17.6 percent. The direct effect on P99 latency from key

selection alone is statistically significant but modest in absolute size once placement stays naive. Key selection alone does drive a statistically significant reduction in modeled network transfer, independent of what placement strategy runs alongside it. That last result held up under correction; it was not close.

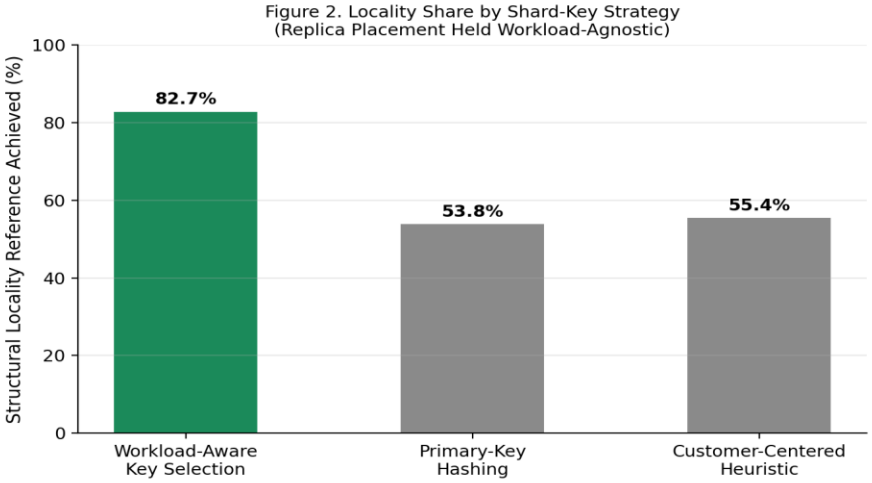


Figure 1. Locality share by shard-key strategy, replica placement held workload-agnostic.

3.2 Replica-Placement Effects Under a Fixed Twelve-Slot Budget

Holding shard-key strategy fixed and varying only placement, workload-aware placement cuts mean latency by roughly 46 to 54 percent, P95 latency by 42 to 50 percent, and P99 latency by 50 to 54 percent - consistently, across every shard-key strategy tested. Every one of these differences is highly significant under the paired evaluation and survives Benjamini-Hochberg correction. What stands out is the consistency itself: the placement benefit does not depend on which key-selection method it is paired with. That pattern points toward two largely separate mechanisms rather than one effect masking or amplifying the other.

Metric	Shard-Key Selection Alone	Replica Placement Alone	Combined Pipeline
Mean latency reduction	Modest (P99 only)	46–54%	53.1%
P99 latency reduction	Significant, modest	50–54%	50.5%
Cross-partition share reduction	24.2%	Not primary lever	24.2%
Network transfer reduction	Significant	Non-significant alone	11.2%

Table 2. Per-mechanism effect sizes: shard-key selection alone vs. replica placement alone vs. combined pipeline.

3.3 Full-Pipeline Outcomes Against the Incumbent Baseline

Against the incumbent configuration - primary-key hash partitioning paired with workload-agnostic placement - the combined pipeline cuts modeled mean latency by 53.1 percent, median latency by 82.6 percent, P95 latency by 45.7 percent, and P99 latency by 50.5 percent (Figure 2). Model-estimated throughput rises 140 percent. Modeled network transfer falls 11.2 percent. Every figure in this paragraph comes from the simulation described in Section 2. None of it is a production measurement, and none of it should be quoted as one.

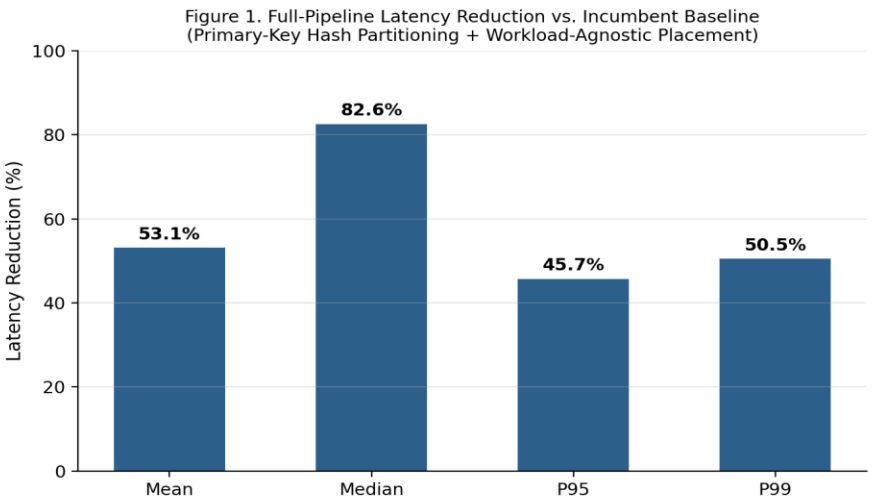


Figure 2. Full-pipeline latency reduction vs. incumbent baseline. .

Migration cost is not free, and the article does not pretend otherwise. Workload-aware key selection changes the hash-baseline shard key for an average of 6.9 of the 24 tables, with an estimated mean rebalancing volume around 0.066 gigabytes in the scaled model - small in this model's scale, though the real-world equivalent would depend heavily on row width and index structure the model does not capture. The optimizer itself finishes in 4.85 seconds per workload on average. Fast enough that re-running the recommendation as traffic patterns drift is not an operational burden worth worrying about [19].

4. Discussion

Each of the four research questions gets a qualified yes, not an unqualified one. On the first - does schema-derived, workload-aware key selection beat hashing and a fixed heuristic - the locality and fan-out numbers say clearly that it does. The catch is P99 latency alone: the direct benefit from key selection with naive placement is real but modest, and a team that fixes only the shard key while leaving replica placement alone should not expect the full latency win this pipeline shows [19].

Placement, under a fixed budget, turns out to be the bigger and steadier latency lever in this evaluation. That is worth sitting with for a moment. Co-partitioning shrinks how many partitions a transaction has to coordinate across; it does not shrink the physical distance a request has to travel. Placement does exactly that, and the twelve-slot cap means the gain is not just a function of handing one strategy more replicas than the other [20].

On what each mechanism contributes alone versus together, the pattern reads as complementary rather than redundant. Key selection's clean, well-isolated win is network-transfer reduction and fan-out. Placement's is latency, especially at the tail. Neither substitutes for the other, and the combined numbers in Section 3.3 exceed what either produces alone without simply being their sum - consistent with the two mechanisms operating on different segments of the request path [21].

Robustness is where the applied-systems discipline shows most clearly. The eight-seed exact-optimum check, the sixteen-seed paired design, correction across fifty-one tests, and a 200-fold sweep on the skew-penalty weight together build a reasonably rigorous case for an evaluation of this kind. None of it substitutes for a live, anonymized enterprise workload trace. We are not claiming it does [22].

Two null results earn a direct mention rather than a quiet footnote. The customer-centered heuristic does not beat plain hashing on overall locality once correction is applied - worth flagging for any team treating "shard by customer" as a self-evidently safe default, because it is not automatically better here. And placement's network-transfer

benefit, isolated from key selection, does not clear significance. Placement earns its keep through latency in this study. Not bandwidth [23].

5. Limitations and Future Work

This is a controlled simulation over a synthetic schema and synthetic workloads (worth restating plainly, since the numbers in Section 3 could otherwise read as stronger claims than they are). Sixteen independently generated seeds show the differences are not an artifact of one particular random draw. They do not establish that the same magnitudes would appear against a specific enterprise system's own access patterns. The latency model leaves out lock contention, buffer-cache behavior, query-optimizer plan changes, node failures, retries, and consensus mechanics - all of which shape real tail latency in ways this model cannot speak to. The workload-ingestion and recommendation architecture in Section 2.3 has never been evaluated as a deployed, end-to-end production system. Its replay-before-apply design reflects a production risk posture, not a proof that it is safe in live operation [23].

Several extensions follow directly. Joint optimization of shard key, primary region, and replica placement as one combined search, rather than the sequential decision modeled here, is the most obvious next step. Live multi-region validation against an anonymized production workload trace would close the largest remaining gap between this study and deployment. A wider sensitivity sweep - capacity-overflow weight, and the relative balance of reads against writes in the objective - is still open. So is exact optimization at schema scales past six tables, a larger seed count, richer anchor inference for ambiguous queries, and a dedicated look at secondary-index selection under the same workload-aware framing [24] .

6. Conclusion

Shard-key selection and replica placement, evaluated jointly under one resource budget rather than in isolation, produce measurable gains over both automatic hashing and the customer-centered heuristic many teams reach for by habit. The two mechanisms work through different paths - key selection cuts cross-partition coordination, placement cuts physical distance - and neither one alone accounts for what the combined pipeline achieves. The evidence here is simulation evidence, reported with its null findings intact rather than smoothed over. The next test this approach needs is a live workload. Not another synthetic seed.

REFERENCES

1. C. Curino, Y. Zhang, E. P. C. Jones, and S. Madden, "Schism: A Workload-Driven Approach to Database Replication and Partitioning," Proceedings of the VLDB Endowment, 3(1–2), pp. 48–57, 2010. https://dl.acm.org/doi/10.14778/1920841.1920853
2. J. Baker et al., "Megastore: Providing Scalable, Highly Available Storage for Interactive Services," CIDR 2011, pp. 223–234. https://research.google/pubs/megastore-providing-scalable-highly-available-storage-for-interactive-services/
3. J. C. Corbett et al., "Spanner: Google's Globally-Distributed Database," ACM Transactions on Computer Systems, 31(3), Article 8, 2013. https://www.usenix.org/system/files/conference/osdi12/osdi12-final-16.pdf
4. R. Taft et al., "CockroachDB: The Resilient Geo-Distributed SQL Database," ACM SIGMOD 2020, pp. 1493–1509. https://dl.acm.org/doi/10.1145/3318464.3386134
5. A. Verbitski et al., "Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases," ACM SIGMOD 2017, pp. 1041–1052. https://homepages.cwi.nl/~boncz/lsde/papers/aurora.pdf
6. X. Zhou, G. Li, J. Feng, L. Liu, and W. Guo, "Grep: A Graph Learning Based Database Partitioning System," PACMMOD, 1(1), Article 61, 2023. https://dl.acm.org/doi/10.1145/3588948
7. S. Cui, H. Wang, J. Lin, X. Ding, and D. Yang, "WorthyPar: A Workload-Aware Data Hybrid Partitioning Advisor with Deep Reinforcement Learning," DASFAA 2025 (LNCS vol. 15989), Springer, pp. 105–122, 2026. https://dl.acm.org/doi/10.1007/978-981-95-4149-2_7
8. Y.-F. Ge et al., "DSGA: A Distributed Segment-Based Genetic Algorithm for Multi-Objective Outsourced Database Partitioning," Information Sciences, 612, pp. 864–886, 2022. https://dl.acm.org/doi/10.1016/j.ins.2022.09.003
9. F. Cui, E. Lo, S. Srivastava, and Z. Lai, "Bonspiel: Low Tail Latency Transactions in Geo-Distributed Databases," PVLDB, 18(11), pp. 3840–3852, 2025. https://dl.acm.org/doi/10.14778/3749646.3749658
10. Q. Zhuang et al., "GeoTP: Latency-Aware Geo-Distributed Transaction Processing in Database Middlewares," IEEE ICDE 2025, pp. 433–445. https://ieeexplore.ieee.org/document/11113150
11. X. Pang and J. Wang, "RaaS: Reducing Tail Latency in Storage-Disaggregated Database Systems," PACMMOD, 4(1), 2026. https://dl.acm.org/doi/10.1145/3786688

12. Z. Wang et al., "TailGuard: A Tail Latency SLO Guaranteed Task Scheduling Scheme for User-Facing Services," IEEE Transactions on Parallel and Distributed Systems, 36(4), 2025. https://ieeexplore.ieee.org/document/10891045
13. W. Wu et al., "Budget-Aware Index Tuning with Reinforcement Learning," ACM SIGMOD 2022, pp. 1084–1097. https://dl.acm.org/doi/10.1145/3514221.3526128
14. T. Siddiqui and W. Wu, "ML-Powered Index Tuning: An Overview of Recent Progress and Open Challenges," ACM SIGMOD Record, 52(4), pp. 6–18, 2023. https://dl.acm.org/doi/10.1145/3641832.3641836
15. V. Giannakouris and I. Trummer, "λ-Tune: Harnessing Large Language Models for Automated Database System Tuning," PACMMOD, 3(1), Article 2, 2025. https://dl.acm.org/doi/10.1145/3709652
16. H. Dong, C. Zhang, G. Li, and H. Zhang, "Cloud-Native Databases: A Survey," IEEE Transactions on Knowledge and Data Engineering, 2024. https://ieeexplore.ieee.org/document/10574374/
17. K. Rajwar, K. Deep, and S. Das, "An Exhaustive Review of the Metaheuristic Algorithms for Search and Optimization," Artificial Intelligence Review, 56(11), pp. 13187–13257, 2023. https://link.springer.com/article/10.1007/s10462-023-10470-y
18. Y. Benjamini and Y. Hochberg, "Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing," Journal of the Royal Statistical Society: Series B, 57(1), pp. 289–300, 1995. https://link.springer.com/article/10.1007/s10462-023-10470-y
19. [19] A. Bhatnagar and V. R. K. Bonagiri, "Operational Excellence through Cloud-Native Distributed Systems," International Journal of Computational and Experimental Science and Engineering, 12(2), 2026. https://doi.org/10.22399/ijcesen.5207
20. K. Rekulapalli, S. R. Kagga, V. Ayyagari, A. Akula, and R. Akula, "AI in HRM: Enhancing Workforce Competency and Organizational Effectiveness," SSRN, 2026.
21. H. Gaggar, "Two Planes of Truth: Execution Fidelity and BehavioralCorrectness in Tool-Using LLM Agents," International Journal of Computational and Experimental Science and Engineering, 10(3), 2024. https://doi.org/10.22399/ijcesen.5498
22. R. K. Shaw, "Kerberos vs. Public Key Authentication in z/OS Environments: Performance, Scalability, and Security Trade-offs," in 2026 9th International Conference on Computing Methodologies and Communication (ICCMC), pp. 1305–1312, IEEE, 2026.
23. H. Gaggar, "Policy-Driven Access Control for Secure Deployment of Autonomous AI Agents in Enterprise Environments," International Journal of Computational and Experimental Science and Engineering, 11(4), 2025. https://doi.org/10.22399/ijcesen.5497
24. K. Rekulapalli, H. Nilagiri, and M. Vakkanthula, "A comparative evaluation of national FHIR Implementation Guides: Strategic methodologies for enhancing interoperability, regulatory alignment, and clinical comprehensiveness in global health IT," Health Systems, pp. 1–13, 2026.