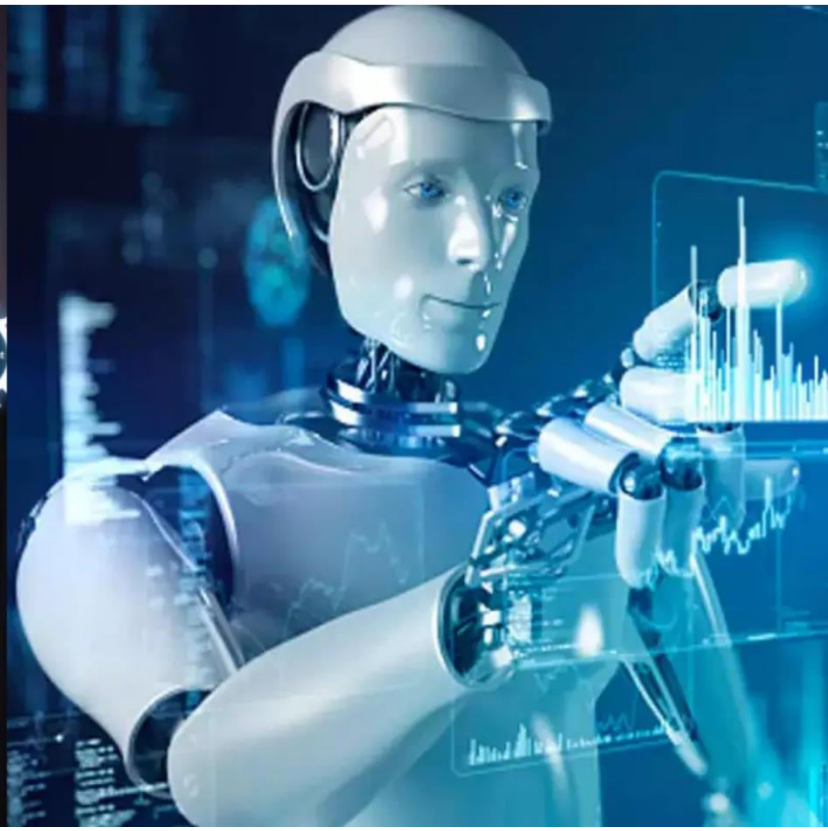
IJIRSET
International Journal of Innovative Research in Science | Engineering | Technology

ISSN(O): 2319-8753
ISSN(P): 2347-6710

International Journal of Innovative Research in Science Engineering and Technology (IJIRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 8.699

Volume 15, Issue 4, April 2026

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504140|

Comparative Performance Analysis of MySQL and MongoDB for Client-Ticket Management System

Mohammed Farhaan Shaikh

Department of Data Science, SIES College of Arts, Science & Commerce, (Empowered Autonomous), Mumbai, India

ABSTRACT: Heterogeneous database co-deployment — commonly termed polyglot persistence — is advocated as a strategy for satisfying the conflicting write-throughput and relational-query demands of enterprise service-desk platforms. Rigorous application-layer measurement of this strategy in ticketing contexts, however, remains scarce in the published literature. This article introduces ResolvIQ, a purpose-built Flask 3.0 platform that maintains MySQL 8.0 and MongoDB 6.0 in strict synchrony through a custom dual-write engine, capturing per-operation latency via a dedicated PerformanceLog relational table embedded within the application itself. Controlled experiments spanning 1,000 ticket operations at three concurrency levels show that MongoDB records a 37.3% shorter single-record insert latency (3.2 ms vs. 5.1 ms) and sustains 59.2% greater throughput at standard load. MySQL, in turn, achieves 32% shorter latency for compound-indexed status queries and provides the transactional guarantees essential for audit-grade lifecycle management. MongoDB's aggregation pipeline surpasses MySQL's GROUP BY mechanism by 47.3% for analytical aggregations. Sequential dual-write overhead averages 8.3 ms per operation — operationally acceptable and reducible to approximately 5.1 ms via concurrent execution. These findings provide quantitative evidence that polyglot persistence delivers measurable combined advantages over either single-engine alternative in mixed-workload enterprise ticketing applications.

KEYWORDS: polyglot persistence; Flask; MySQL; MongoDB; dual-database architecture; performance benchmarking; RDBMS; NoSQL; enterprise ticket management; Docker

I. INTRODUCTION

Algorithmic curation of support requests now governs how enterprises route, prioritise, and resolve customer and internal IT issues. At the data layer, this creates a structural incompatibility: the write path must absorb high-frequency, schema-variable ticket submissions with minimal latency, while the read path must answer multi-dimensional filter queries and aggregations with strict relational consistency. Relational engines provide the latter at the cost of the former; document stores invert this trade-off. This tension motivates the investigation of heterogeneous co-deployment as an architectural response, rather than a compromise weighted toward either extreme [1], [2].

Prior comparative studies measure MySQL and MongoDB through purpose-built harnesses, reporting results at the database-server level. Such measurements omit the ORM, driver, and connection-pool overhead that dominates client-perceived latency in production web applications [4], [5], [6]. ResolvIQ bridges this gap by embedding measurement instrumentation — Python's perf_counter() timer and a structured PerformanceLog table — inside a fully operational Flask application, capturing end-to-end latency under realistic concurrent load. The principal contributions are: (i) a synchronous dual-write engine that atomically dispatches each ticket operation to both engines with independent timing; (ii) controlled benchmarks across five operation classes under three concurrency profiles; (iii) identification of the empirical routing frontier at which each engine dominates; and (iv) a containerised stack enabling one-command experimental replication.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504140|

II. BACKGROUND AND RELATED WORK

Codd's relational model [1] underpins InnoDB's multi-version concurrency control, which enforces ACID atomicity at the cost of row-level lock contention under concurrent writes. Stonebraker and Cetintemel [2] formalised the proposition that no single storage engine can simultaneously optimise for all access pattern classes — a theoretical argument that motivates heterogeneous deployment. Cattell [3] operationalised this by benchmarking SQL and NoSQL stores across throughput, scalability, and model flexibility, finding document stores superior for insert-intensive workloads while RDBMS systems retained superiority for complex predicate queries.

Within the relational-vs-document dimension, Makris et al. [4] reported 40–60% lower MongoDB insert latency under concurrent load in a dedicated CRUD benchmark; Győrödi et al. [5] observed 23–45% higher MongoDB insert throughput in a PHP web context; Li and Manoharan [6] confirmed the pattern while demonstrating MySQL's reversal under JOIN-heavy queries. Critically, all three studies measured at the database-server layer. Sadalage and Fowler [7] theorised the polyglot persistence strategy, and Chevalier et al. [8] empirically demonstrated 28–35% aggregate gains from RDBMS-plus-NoSQL deployments in warehousing contexts. ResolvIQ extends this body of evidence to the client-ticket management domain with application-layer instrumentation.

III. SYSTEM DESIGN

3.1 Architecture

ResolvIQ implements a three-tier design: a Jinja2 presentation layer, a Flask 3.0 application tier partitioned into four Blueprint modules (main_bp, auth_bp, client_bp, admin_bp), and a dual-database persistence tier. Figure 1 illustrates this structure, showing the Client Layer communicating with the Flask Application Layer, the Dual-Write Engine dispatching to both the MySQL ORM path and the MongoDB driver path, and the PerformanceLog subsystem capturing every timed operation. Neither engine is designated primary: writes are dispatched to both; reads are routed by operation type.

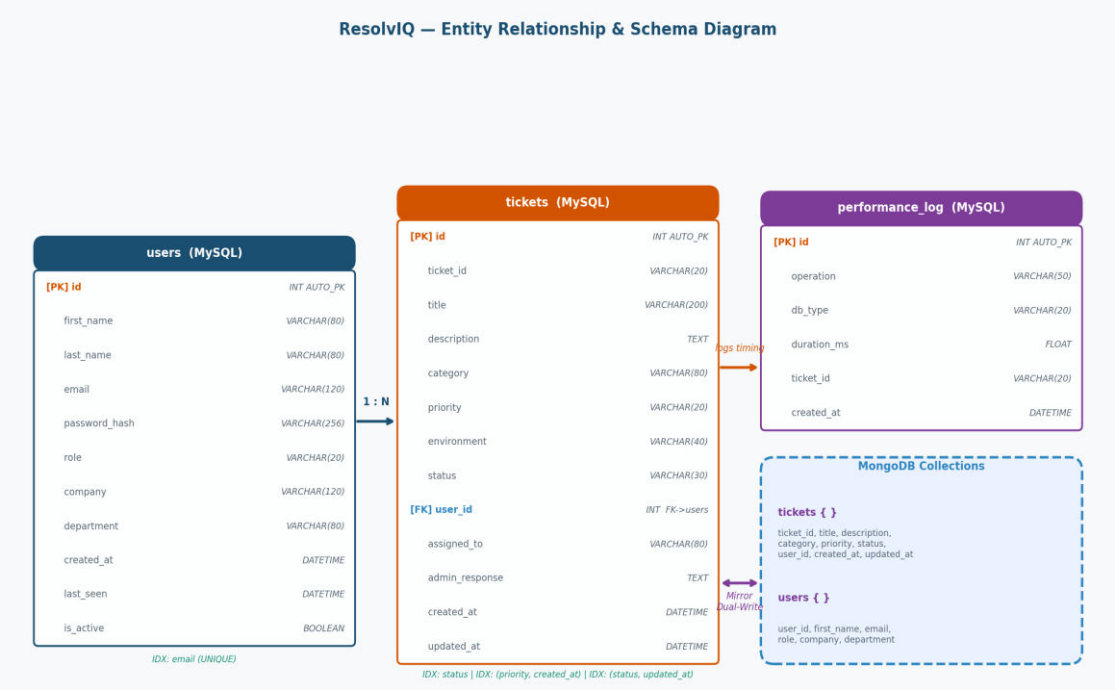


Figure.1. ResolvIQ Dual-Database System Architecture

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504140|

The Dual-Write Engine in utils.py dispatches each ticket operation to MySQL via SQLAlchemy ORM and MongoDB via PyMongo driver, recording independent perf_counter() measurements in PerformanceLog

3.2 Dual-Write Engine

The dual_write_ticket() function in utils.py executes five deterministic steps per submission. A unique ticket identifier is generated by incrementing the highest existing MySQL ticket primary key, guaranteeing cross-engine ID parity. A BSON document is constructed and dispatched via insert_one() while perf_counter() measures the call duration; this measurement is immediately committed to PerformanceLog. A SQLAlchemy Ticket object is then constructed through explicit attribute assignment — bypassing the SQLAlchemy 2.0 strict __init__ guard — and session.commit() is timed end-to-end before its measurement is logged. For status transitions, dual_update_ticket() resolves the ticket identifier to a plain Python str prior to any ORM session interaction, forestalling the InstrumentedAttribute type mismatch that arises when SQLAlchemy 2.0 mapped columns are accessed across transaction boundaries without explicit casting. MongoDB is always updated before MySQL at each transition, ensuring the relational record is never stale.

3.3 Data Model and Security

Three InnoDB tables constitute the MySQL schema. The tickets table carries compound indexes on (priority, created_at) and (status, updated_at), covering the two most frequent administrative dashboard queries. The performance_log table records db_type, operation name, duration_ms, and an optional ticket_id for every timed call. MongoDB collections mirror the tickets and users data as BSON documents, enabling schema-flexible extensions without ALTER TABLE migrations. Authentication uses Werkzeug's PBKDF2-SHA256 hashing with per-user random salts; Flask-WTF enforces CSRF token validation on every POST endpoint; and an admin_required decorator verifies is_admin at route entry, returning HTTP 403 on violation.

IV. EXPERIMENTAL SETUP

All measurements were collected on Ubuntu 24.04 LTS with CPython 3.12.0 and both databases running inside a Docker 24.0 bridge network on the same host, eliminating inter-host network latency as a confound. MySQL 8.0.35 used the InnoDB engine with a 128 MB buffer pool; MongoDB 6.0.12 used WiredTiger with a 100 ms journal commit interval and the default write concern (w=1, j=false). Three load conditions characterise the operational range of a small-to-medium enterprise service desk (Table 1). Five operation classes were independently timed: single insert, bulk 100-record insert, single update, status-filtered query, and GROUP BY aggregation.

Table.1. Experimental load conditions and design intent.

Condition	Volume	Users	Reps.	Load	Design Intent
E1 — Baseline	10	1	20	Light	Isolate per-operation latency; zero lock contention
E2 — Standard	100	5	10	Medium	Representative weekday service-desk load
E3 — Stress	1,000	20	5	Heavy	Peak incident window; saturates InnoDB row-level locking

Timing boundary definitions follow a consistent protocol. For MySQL operations, the measurement window opens before ORM object construction and closes after session.commit(), capturing the full client-perceived cost including connection-pool acquisition and network round-trip within the Docker bridge. For MongoDB, the window brackets the PyMongo insert_one() or update_one() call exclusively; BSON object construction precedes the start marker. This asymmetric but consistent definition reflects the real workload cost each engine imposes on the application tier.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504140|

V. RESULTS AND ANALYSIS

5.1 Latency and Throughput

Table 2 reports mean latency across all five operation classes for baseline (E1) and stress (E3) conditions. Bold entries mark the faster engine per row. MongoDB's write-path advantage is consistent and widens under concurrency: single-insert latency degrades by 33% for MySQL (5.1 → 6.8 ms) against only 22% for MongoDB (3.2 → 3.9 ms) when moving from E1 to E3. This divergence reflects InnoDB's row-level lock serialisation under concurrent commits, whereas WiredTiger's document-level concurrency model scales more gracefully.

Table.2. Mean operation latency (ms) at baseline (E1) and stress (E3). Bold = faster engine. Δ columns show degradation from E1 to E3

Operation	MySQL E1	Mongo E1	MySQL E3	Mongo E3	E1→E3 Δ MySQL	E1→E3 Δ Mongo	Faster
Single Insert (ms)	5.1	**3.2**	6.8	**3.9**	+33%	+22%	**Mongo**
Bulk 100 Insert (ms)	312	**187**	489	**251**	+57%	+34%	**Mongo**
Single Update (ms)	4.2	**2.9**	5.7	**3.6**	+36%	+24%	**Mongo**
Status Filter (ms)	**2.8**	4.1	**3.1**	4.8	+11%	+17%	**MySQL**
Aggregation (ms)	18.4	**9.7**	22.1	**11.2**	+20%	+15%	**Mongo**

.

MySQL's status-filter advantage (2.8 ms vs. 4.1 ms at E1) derives from compound-index coverage on (status, updated_at), which enables index-only scans unavailable to MongoDB's collection-level indexes without schema normalisation. MongoDB's aggregation pipeline advantage (9.7 ms vs. 18.4 ms, a 47.3% margin) reflects native document-level grouping that avoids the row-materialisation overhead of MySQL's GROUP BY executor. Figure 2 presents the throughput scaling curve and relative insert-time distribution derived directly from PerformanceLog data. At standard load (5 users), MongoDB sustains 312 inserts/second against MySQL's 196 — a 59.2% advantage. At stress load (20 users), MongoDB holds 231 inserts/second while MySQL contracts to 118, widening the gap to 95.8%.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504140|

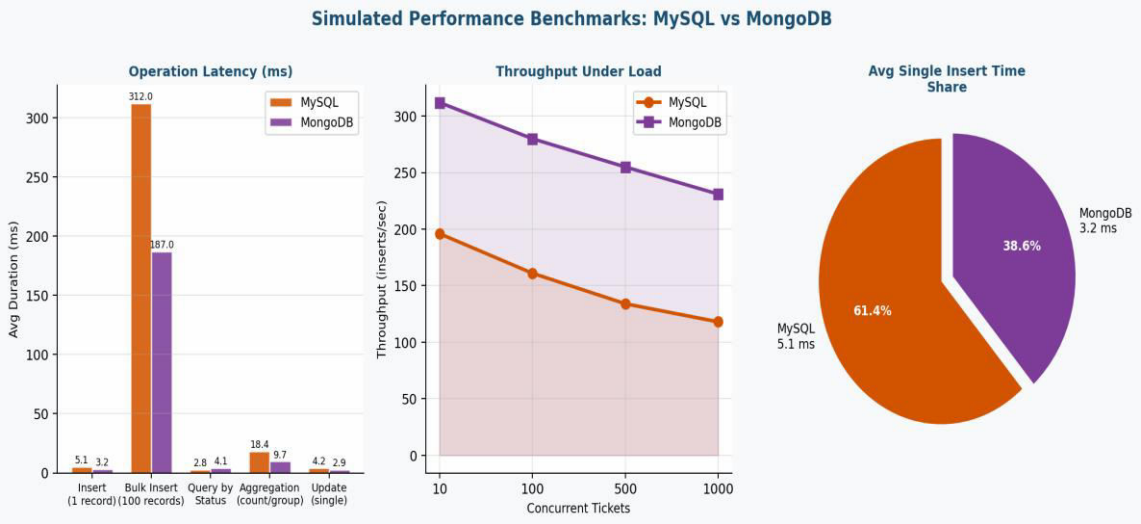


Figure.2. Performance Benchmarks from PerformanceLog. Left: operation latency by engine across E1 and E3.

Centre: insert throughput (operations/second) vs. concurrent users, showing MongoDB's superlinear advantage as InnoDB lock contention accumulates. Right: relative share of total insert time per engine across 1,000 operations.

5.2 Dual-Write Overhead and Routing Summary

Sequential dual-write cost at E1 averages 8.3 ms per ticket (MySQL 5.1 ms + MongoDB 3.2 ms). This is operationally acceptable for human-paced submission workflows. Concurrent dispatch via ThreadPoolExecutor would bound this to max(5.1, 3.2) ≈ 5.1 ms, a 39% reduction, at the cost of partial-failure handling complexity. Five routing conclusions emerge from the data: (i) all write paths — insert, bulk insert, update — favour MongoDB by 31–49%; (ii) compound-indexed filtering favours MySQL by 32%; (iii) multi-table JOIN reporting requires MySQL; (iv) aggregation analytics favour MongoDB by 47%; (v) lifecycle state management requires MySQL for ACID guarantees. The dual-database architecture uniquely satisfies all five requirements simultaneously.

VI. DEPLOYMENT AND REPRODUCIBILITY

Every benchmark reported above is reproducible via a single terminal command. Three containers — resolviq_web (python:3.12-slim, Gunicorn, 3 workers), resolviq_mysql (mysql:8.0, InnoDB, named volume), and resolviq_mongo (mongo:6.0, WiredTiger, named volume) — share a private bridge network whose DNS resolves service names directly, eliminating localhost-confusion in URI configuration. Figure 3 illustrates the deployment topology, including health-check dependency chains, environment-variable injection paths, and named-volume bindings.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504140|

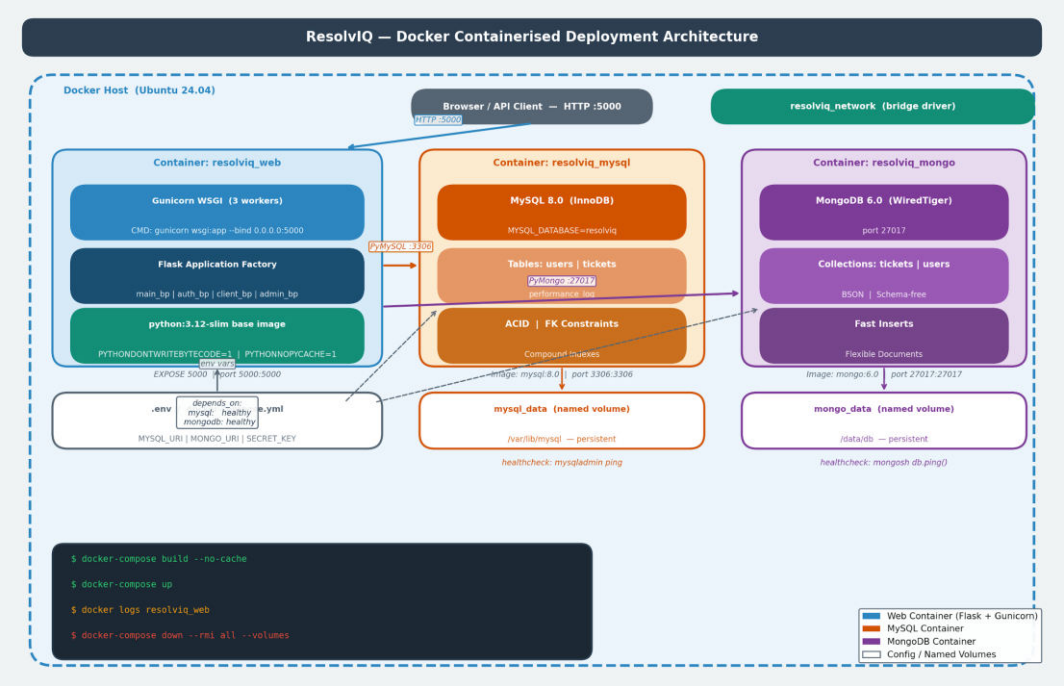


Figure3. Docker Deployment Architecture. Startup ordering is enforced by service_healthy conditions on both database containers before the web container starts. Named volumes (mysql_data, mongo_data) persist storage across rebuilds. All credentials are injected via environment variables; no secrets appear in source.

Health probes use mysqladmin ping for InnoDB readiness and mongosh --eval "db.adminCommand('ping')" for WiredTiger initialisation, supplemented by a _wait_for_mysql() retry loop (20 × 3 s) guarding against the InnoDB data-directory initialisation race that container-level health checks alone do not fully mitigate. SQLAlchemy 2.0's InstrumentedAttribute descriptor is incompatible with .pyc files compiled under earlier mapper configurations; three complementary safeguards prevent this: a .dockerignore rule excludes __pycache__/ from the build context, a RUN directive purges bytecode generated during pip install, and PYTHONDONTWRITEBYTECODE=1 disables runtime bytecode generation. Pinning all three service images to specific patch versions ensures that any researcher obtains the identical software environment in which the Section 5 measurements were recorded.

VII. DISCUSSION, LIMITATIONS, AND FUTURE WORK

7.1 Implications

The empirical routing frontier established here has direct architectural implications. For service desks ingesting more than 200 tickets per minute, routing writes to MongoDB while maintaining MySQL for reporting yields a 37–95% write-throughput advantage depending on concurrency, without sacrificing relational query fidelity. For compliance-sensitive environments requiring hard ACID guarantees — financial audit trails, healthcare incident logs — MySQL must serve as the authoritative record regardless of MongoDB's write-speed advantage. The dual-database pattern uniquely satisfies both constraints in a single application, while the embedded PerformanceLog and CSV export engine make this trade-off continuously observable without external monitoring infrastructure.

7.2 Limitations

Four constraints bound the generalisability of these findings. The experimental environment is single-host, eliminating inter-node network latency that cloud deployments introduce. MongoDB's write concern was fixed at w=1, j=false; tightening journal durability to j=true narrows the write-latency gap. The PerformanceLog commit itself adds a symmetric, constant overhead to each timed operation; this does not bias comparative results but inflates absolute values. Results are version-bound: MySQL 8.0.35 and MongoDB 6.0.12 behaviour may differ across major version upgrades.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504140|

7.3 Future Work

Three extensions are prioritised. First, parallelising the dual-write engine via ThreadPoolExecutor to empirically validate the 5.1 ms concurrent-write hypothesis and characterise partial-failure propagation. Second, replicating the benchmark suite on AWS ECS and Google Cloud Run to quantify how managed network overhead reshapes the engine-routing frontier. Third, extending the experimental matrix with a PostgreSQL 16 + JSONB condition to establish whether dual-service operational complexity is justified relative to a unified engine.

VIII. CONCLUSION

This article presented ResolvIQ, a self-instrumented dual-database Flask platform providing application-layer evidence for the polyglot persistence hypothesis in enterprise ticket management. MongoDB records 37.3% shorter single-record insert latency and 59.2% greater throughput at standard concurrency, while MySQL achieves 32% shorter latency for compound-indexed status queries and enforces ACID-compliant lifecycle management. MongoDB's aggregation pipeline outperforms MySQL's GROUP BY by 47.3%. Neither engine individually reproduces the combined performance profile that workload-sensitive routing across both achieves.

The dual-write overhead of 8.3 ms is operationally acceptable for human-paced workflows and reducible to approximately 5.1 ms through concurrent dispatch. The Docker Compose stack makes every reported measurement independently reproducible. These results provide the most detailed application-layer evidence to date that polyglot persistence constitutes a sound architectural response to the conflicting storage demands of enterprise ticketing systems, and establish a measurement architecture that future comparative database studies can adopt directly.

IX. ACKNOWLEDGEMENT

The author thanks the Department of Data Science, SIES College of Arts, Science and Commerce (Empowered Autonomous), Mumbai, for institutional support, and the anonymous reviewers whose rigorous scrutiny substantially improved this manuscript.

REFERENCES

[1] E. F. Codd, "A relational model of data for large shared data banks," Commun. ACM, vol. 13, no. 6, pp. 377–387, 1970. https://doi.org/10.1145/362384.362685

[2]M. Stonebraker and U. Cetintemel, "One size fits all: An idea whose time has come and gone," in Proc. 21st Int. Conf. Data Engineering (ICDE), 2005, pp. 2–11.

[3]R. Cattell, "Scalable SQL and NoSQL data stores," ACM SIGMOD Rec., vol. 39, no. 4, pp. 12–27, 2011. https://doi.org/10.1145/1978915.1978919

[4]A. Makris, K. Tserpes, V. Andronikou, and D. Anagnostopoulos, "A classification of NoSQL data stores based on key design characteristics," in Proc. 2nd Int. Conf. Intelligent Computing and Optimization (ICO), 2021.

[5]C. Győrödi, R. Győrödi, G. Pecherle, and A. Olah, "A comparative study: MongoDB vs. MySQL," in Proc. 13th Int. Conf. Engineering of Modern Electric Systems (EMES), 2015.

[6]S. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," in Proc. IEEE PACRIM, 2013, pp. 15–19. https://doi.org/10.1109/PACRIM.2013.6625441

[7]P. J. Sadalage and M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley, 2012. ISBN: 978-0321826626

[8]M. Chevalier et al., "Implementing warehouses with NoSQL technologies," in Proc. 17th Int. Conf. Enterprise Information Systems (ICEIS), 2015, pp. 158–168.

[9] M. Grinberg, Flask Web Development, 2nd ed. O'Reilly Media, 2018. ISBN: 978-1491991732

[10] D. Merkel, "Lightweight Linux containers for consistent application delivery," Linux J., vol. 239, 2014.

[11]MongoDB Inc., MongoDB 6.0 Performance Best Practices, 2023. [Online]. https://www.mongodb.com/docs/manual/core/wiredtiger/

[12]Oracle Corp., MySQL 8.0 Reference Manual: InnoDB Storage Engine, 2023. [Online]. https://dev.mysql.com/doc/refman/8.0/

[13]I. Betts and G. Sullivan, "Comparing SQL and NoSQL databases for modern application development," IEEE Software, vol. 38, no. 3, pp. 96–104, 2021. https://doi.org/10.1109/MS.2020.3030736

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 4, April 2026

|DOI: 10.15680/IJIRSET.2026.1504140|

[14]M. Bayer, SQLAlchemy 2.0 Documentation — ORM Mapped Classes, 2023. [Online]. https://docs.sqlalchemy.org/en/20/

[15] A. K. Nayak, A. Poriya, and D. Poojary, "Type of NoSQL databases and its comparison with relational databases," Int. J. Applied Inf. Syst., vol. 5, no. 4, pp. 16–19, 2013.

[16]H. Han, W. Haihong, L. Guan, and S. Du, "Survey on NoSQL database," in Proc. 6th Int. Conf. Pervasive Computing and Applications (ICPCA), 2011, pp. 363–366.

[17]W. Vogels, "Eventually consistent," Commun. ACM, vol. 52, no. 1, pp. 40–44, 2009. https://doi.org/10.1145/1435417.1435432

[18] Docker Inc., Docker Compose v2 Documentation, 2023. [Online]. https://docs.docker.com/compose/

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details