

Benchmarking PostgreSQL and MongoDB for Image-Based Time-Series Data

1st Alfa Yohannis
Department of Informatics
Universitas Pradita
Tangerang, Indonesia
alfa.ryano@gmail.com

2nd Alexander Waworuntu
Department of Informatics
Universitas Multimedia Nusantara
Tangerang, Indonesia

3rd Master Edison Siregar
Department of Informatics
Universitas Pradita
Tangerang, Indonesia

Abstract—This paper presents a performance comparison between PostgreSQL with TimescaleDB and MongoDB for image-based time-series workloads in consumer electronics and IoT systems. Using 320x240 JPEG frames, the study evaluates insertion throughput, aggregation latency, driver overhead, and storage utilization under controlled conditions. Experimental results show that MongoDB achieves higher ingestion throughput and significantly smaller storage footprint due to native compression, while PostgreSQL provides substantially lower aggregation latency and more stable analytical performance. These findings highlight clear trade-offs between write efficiency, query performance, and storage efficiency for vision-oriented time-series applications.

Index Terms—time-series databases, image-based IoT, PostgreSQL, MongoDB, consumer electronics

I. INTRODUCTION

The rapid proliferation of consumer electronics (CE) and Internet of Things (IoT) devices equipped with embedded vision capabilities has generated unprecedented volumes of image-based time-series data, ranging from security cameras and smart doorbells to industrial inspection systems and wearable health monitors [1], [2]. These systems produce continuous streams of timestamped image frames alongside metadata such as device identifiers and sensor parameters, placing unique demands on database management systems for high-throughput ingestion, efficient storage of binary payloads, and low-latency temporal analytics [3], [4]. Traditional relational databases struggle with the scale and velocity of such workloads, while specialized time-series databases (TSDBMSs) and document stores have emerged as viable alternatives, yet their comparative performance for image-centric time-series remains underexplored [5], [6].

PostgreSQL extended with TimescaleDB and MongoDB represent two prominent architectural approaches for managing time-series data in production environments [7], [8]. TimescaleDB augments PostgreSQL’s relational foundation with hypertables, automatic time-based partitioning, and chunked storage optimized for temporal queries, while MongoDB leverages its document model and native time-series collections with WiredTiger compression for flexible schema handling and high-ingestion workloads [9], [10]. Existing benchmarks primarily focus on scalar sensor metrics (e.g., temperature, CPU utilization) using tools such as the Time

Series Benchmark Suite (TSBS), leaving a critical gap in understanding how these systems perform when handling realistic image payloads representative of CE vision streams [11]–[13].

This study addresses this gap through a systematic performance comparison of PostgreSQL with TimescaleDB versus MongoDB for image-based time-series workloads modeled after consumer electronics applications. The primary aim is to quantify trade-offs across key metrics—insertion throughput, aggregation latency, driver overhead, and storage efficiency—under controlled conditions using standardized 320×240 JPEG frames with accompanying metadata on commodity hardware. This work introduces a reproducible benchmarking methodology for image-based time-series data, provides empirical evidence of complementary strengths between relational time-series extensions and document stores, and offers actionable guidance for CE system architects in selecting databases based on ingestion rate requirements, analytical latency needs, and storage constraints in multi-camera deployments. These findings extend prior TSDBMS surveys by addressing multimedia time-series use cases that are increasingly prevalent in next-generation vision-enabled IoT systems [3], [5].

II. RELATED WORK

Time-series database management systems (TSDBMSs) have been extensively studied for workloads dominated by temporally ordered sensor and event data, particularly in IoT and DevOps contexts [3], [5]. Prior surveys report that specialized engines and extensions, such as TimescaleDB on PostgreSQL, introduce time-based partitioning, chunking, and compression mechanisms to overcome the scalability and performance limitations of traditional row-oriented relational databases [4], [5], [13]. These studies consistently identify ingestion throughput, query latency, and storage footprint as the dominant performance metrics, which directly align with the dimensions evaluated in this work.

Several benchmarks and industrial reports directly compare TimescaleDB with general-purpose document stores such as MongoDB for time-series workloads [7], [8], [11]. Results commonly indicate that TimescaleDB’s hypertable abstraction and chunk-based storage provide competitive or superior ingestion performance and substantially faster time-based aggrega-

gations than MongoDB for metric-style data, especially for large device fleets and long time ranges [9]. At the same time, MongoDB’s native time-series collections and document-oriented schema offer operational simplicity and flexible schema evolution, which remain attractive for heterogeneous IoT deployments despite higher aggregation overheads in some benchmarks [10].

In the consumer electronics and IoT domain, most prior studies focus on scalar sensor signals such as temperature, power, or motion rather than image-based time-series streams [1], [2]. Database selection guidelines often emphasize high ingest rates, retention policies, and compression, and frequently recommend TimescaleDB and MongoDB for time-stamped data, yet without addressing large binary payloads such as JPEG frames [14]. Existing benchmarking tools, including TSBS, rely primarily on synthetic numeric workloads [11], [12], leaving a clear gap in understanding database behavior under image-centric workloads. The present study addresses this gap by evaluating PostgreSQL with TimescaleDB and MongoDB under identical binary-image ingestion, aggregation, and storage conditions, thereby extending TSDBMS evaluations to vision-oriented consumer electronics and embedded IoT systems [3]–[5].

III. METHODOLOGY

This study evaluates PostgreSQL with TimescaleDB and MongoDB for image-based time-series workloads representative of consumer electronics (CE) and IoT systems. The dataset consists of fixed-size image frames generated from a single source image resized to 320×240 pixels and encoded in JPEG format, reflecting the resolution of low-power embedded vision devices and entry-level camera sensors. Each frame is stored together with a device identifier, timestamp, image dimensions, and MIME type. To ensure controlled and repeatable experimental conditions, the same image is reused for all insert operations, eliminating variability caused by heterogeneous content.

In the PostgreSQL configuration, image frames are stored in a relational time-series table configured as a TimescaleDB hypertable. Each record includes a unique frame identifier, device identifier, high-resolution timestamp, raw image bytes stored as a binary attribute, image width and height, MIME type, and a creation timestamp. To satisfy TimescaleDB requirements, a composite primary key consisting of the timestamp and unique frame identifier is used, enabling correct partitioning and time-based chunk management for efficient temporal indexing and scalable storage.

In MongoDB, the same logical information is stored as documents in a native time-series collection. Each document contains a device identifier, timestamp, binary image payload stored as `BinData`, image width and height fields, and a MIME type field. MongoDB relies on the WiredTiger storage engine with native block-level compression applied to both data and indexes. In contrast, PostgreSQL uses uncompressed relational storage with TOAST mechanisms for handling large binary objects. Although the physical storage layouts differ,

TABLE I: Performance and Storage Comparison between PostgreSQL and MongoDB

Metric	PostgreSQL	MongoDB
Insert Throughput (rows/sec)		
PostgreSQL vs MongoDB	1785.05 ± 274.34	3783.46 ± 551.40
Aggregation Latency (ms)		
PostgreSQL vs MongoDB	23.97 ± 0.82	353.13 ± 16.53
Driver Roundtrip Latency (ms)		
PostgreSQL vs MongoDB	0.0415 ± 0.0360	0.0858 ± 0.0030
Storage Size (MB)		
Table / Collection	4777.72	835.41
Database Total	4787.29	835.45

both systems store identical logical content to ensure a fair comparison of storage and performance behavior.

The benchmarking process consists of three phases: insertion throughput, aggregation performance, and driver roundtrip overhead. For insertion benchmarking, each measured run inserts 200,000 image frames using a fixed batch size of 1,000 records per transaction, following a warm-up phase to stabilize internal buffers and storage allocation. Each measured run starts from an empty table or collection to ensure identical initial conditions. Aggregation benchmarking uses fixed one-minute time-bucket queries to compute total frame counts and average image sizes, while driver overhead is measured using minimal roundtrip queries to isolate client–server communication latency independent of query complexity.

Four primary metrics are evaluated: insertion throughput (rows/sec), aggregation latency (ms), driver roundtrip latency (ms), and storage utilization (MB). PostgreSQL storage is measured using relation size functions and TimescaleDB hypertable size functions, while MongoDB storage is obtained using `collStats` and `dbStats`. All experiments are conducted on Ubuntu 22.04 with an Intel Core i7 processor, 16 GB memory, and 4 physical cores (8 logical threads). The database stack consists of PostgreSQL 18.1 with TimescaleDB 2.24.0 and MongoDB 7.0.26. The benchmarking framework is implemented in Python using `psycopg2` 2.9.10 and `PyMongo` 4.15.4 with synchronous single-connection clients to isolate intrinsic database engine performance.

A. Insert Throughput Performance

Insert throughput is a key performance indicator for image-based time-series workloads, as it reflects the system’s ability to ingest continuous high-volume sensor data. As shown in Table I, MongoDB achieves a substantially higher average insertion rate of 3783.46 ± 551.40 rows/sec, while PostgreSQL records 1785.05 ± 274.34 rows/sec. This result indicates that MongoDB sustains approximately twice the ingestion throughput of PostgreSQL under identical experimental conditions.

The distribution of throughput values is further illustrated in Fig. 1a. MongoDB exhibits higher peak throughput but also larger variability, as reflected by its wider interquartile range and standard deviation. In contrast, PostgreSQL shows a more

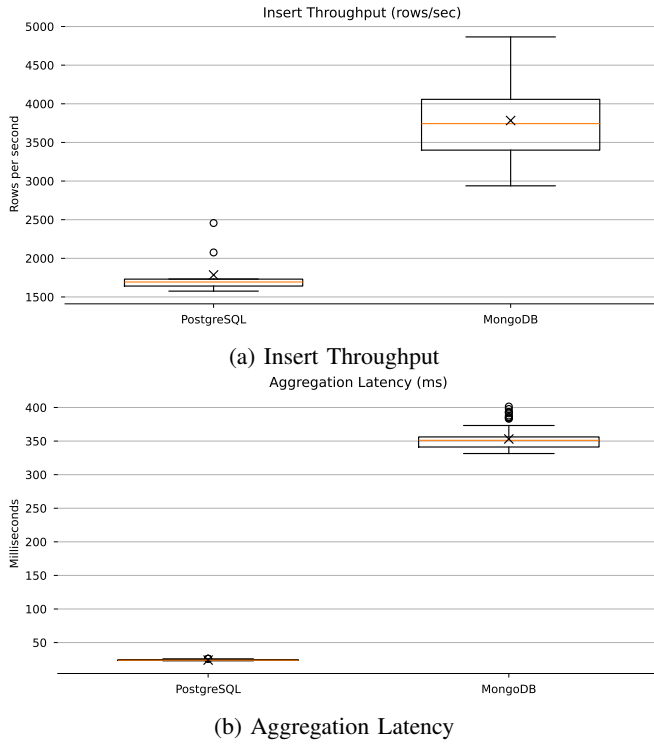


Fig. 1: Performance comparison of PostgreSQL and MongoDB across insertion throughput and aggregation latency.

compact distribution, indicating more stable and predictable insertion behavior across runs.

From a system design perspective, MongoDB benefits from its document-oriented storage engine and optimized bulk insert pipeline, which reduces per-record transaction overhead. PostgreSQL with TimescaleDB, while offering stronger transactional guarantees and durability through write-ahead logging, incurs higher synchronization and write amplification costs. For consumer electronics and IoT image streaming applications, these results suggest that MongoDB is better suited for high-speed data ingestion, whereas PostgreSQL offers superior stability and consistency.

B. Aggregation Query Performance

Aggregation latency reflects the efficiency of analytical processing over stored time-series image data. As reported in Table I, PostgreSQL significantly outperforms MongoDB in aggregation performance, achieving an average latency of 23.97 ± 0.82 ms, while MongoDB records a much higher latency of 353.13 ± 16.53 ms. This result demonstrates an order-of-magnitude advantage for PostgreSQL in executing time-bucketed aggregation queries.

The performance distribution is visualized in Fig. 1b. PostgreSQL exhibits both low median latency and a narrow interquartile range, indicating highly consistent analytical performance across repeated runs. In contrast, MongoDB shows substantially higher latency with wider dispersion, reflecting

greater execution variability caused by its document-scanning and pipeline-based aggregation mechanism.

This performance gap is primarily attributed to PostgreSQL’s query optimizer, columnar-style access patterns in TimescaleDB, and its native support for time-based partitioning via hypertables. These features enable efficient chunk pruning and index utilization during aggregation. MongoDB, while flexible in schema design, processes aggregations through its document pipeline engine, which introduces higher computational overhead for large-scale time-bucket operations. For consumer electronics and IoT workloads requiring real-time or near-real-time analytics, PostgreSQL therefore provides a clear advantage in aggregation performance.

C. Driver Roundtrip Overhead

Driver roundtrip overhead represents the baseline communication and parsing cost between the application and the database engine, independent of query complexity. As shown in Table I, PostgreSQL exhibits a very low average roundtrip latency of 0.0415 ± 0.0360 ms, while MongoDB records 0.0858 ± 0.0030 ms. Both values are in the sub-millisecond range, indicating that driver communication overhead is negligible when compared to insertion and aggregation costs.

Although MongoDB shows approximately twice the roundtrip latency of PostgreSQL in relative terms, the absolute difference remains extremely small and does not materially affect end-to-end system performance for bulk ingestion or analytical workloads. The dominant performance factors therefore remain disk I/O, write amplification, indexing, and query execution rather than client–server communication latency. This confirms that the throughput and aggregation benchmarks primarily reflect backend engine behavior rather than client library inefficiencies.

D. Storage Utilization Analysis

Storage efficiency is a critical factor for large-scale image-based time-series systems, particularly in consumer electronics environments with limited storage capacity. As reported in Table I, PostgreSQL requires 4777.72 MB for the table and 4787.29 MB at the database level, while MongoDB consumes only 835.41 MB for the collection and 835.45 MB for the database. This indicates that MongoDB uses approximately six times less storage than PostgreSQL for the same dataset.

This substantial difference is primarily attributed to MongoDB’s built-in compression mechanisms in the WiredTiger storage engine, which apply block-level compression to both data and indexes. In contrast, PostgreSQL stores large binary objects as uncompressed BYTEA values within table and TOAST storage structures. As a result, although both systems store identical image content at the logical level, MongoDB achieves a significantly more compact physical representation on disk.

From a system design perspective, these results suggest that MongoDB is better suited for storage-constrained IoT and consumer electronics deployments where minimizing on-device storage footprint is essential. PostgreSQL, while incurring higher storage overhead, offers stronger transactional

guarantees and mature relational indexing support. The choice between the two systems therefore involves a clear trade-off between storage efficiency and relational robustness in large-scale image time-series applications.

E. Overall System Trade-offs

The experimental results reveal clear and complementary trade-offs between PostgreSQL and MongoDB across the three primary performance dimensions of this study: insertion throughput, aggregation latency, and storage utilization. MongoDB consistently demonstrates superior insertion throughput and significantly lower storage footprint, making it highly efficient for high-rate image ingestion and storage-constrained environments. In contrast, PostgreSQL exhibits substantially lower aggregation latency and more stable analytical performance, indicating its strength in query-intensive and real-time analytical workloads.

When interpreted in terms of frame rate requirements for consumer electronics (CE) devices, the insertion throughput directly determines the maximum sustainable frames per second (FPS). At a resolution of 320×240 , MongoDB achieves an average throughput of approximately 3783 rows/sec, indicating that it can theoretically sustain several thousand image frames per second from distributed sources, making it suitable for multi-camera and high-speed sensing scenarios. PostgreSQL, sustaining approximately 1785 rows/sec, remains fully adequate for typical CE frame rates of 10–60 FPS per device, but offers a smaller margin for high-density, multi-stream deployments. Conversely, the significantly lower aggregation latency of PostgreSQL enables near-real-time frame-level analytics, which is critical for applications such as video surveillance, quality inspection, and anomaly detection. From a system design perspective, MongoDB is therefore well suited for edge-level frame ingestion where high FPS and compact storage dominate, whereas PostgreSQL is better positioned for backend analytics where fast temporal aggregation and stable query performance are required.

IV. CONCLUSIONS

This study presented a comprehensive experimental comparison between PostgreSQL with TimescaleDB and MongoDB for image-based time-series workloads representative of consumer electronics and IoT systems. The results demonstrated that MongoDB provides higher insertion throughput and significantly better storage efficiency due to its compressed document-oriented storage model, while PostgreSQL achieves substantially lower aggregation latency and more stable analytical performance through time-based partitioning and mature query optimization. When interpreted in terms of frame rate and image resolution, both systems are capable of supporting real-time ingestion for low-resolution streams, while exhibiting different scalability characteristics at higher resolutions. These findings confirm that database selection should be guided by application priorities across ingestion speed, analytical latency, and storage constraints. As future work, this study will be extended to evaluate performance

under multiple concurrent client connections, parallel ingestion and query processing, and higher-resolution image streams (e.g., VGA, HD, and 720p) to better reflect real-world multi-camera consumer electronics deployments.

ACKNOWLEDGMENT

This research was supported by Universitas Pradita and Universitas Multimedia Nusantara.

REFERENCES

- [1] Intuz, "Top databases in the market for iot applications," <https://www.intuz.com/guide-on-top-iot-databases>, 2023, accessed: 2025-12-05.
- [2] E. Cholakova *et al.*, "Communication interfaces, protocols, and specialized time series databases for temporal information in iot," <https://epluse.ceec.bg/wp-content/uploads/2025/07/20250102-04.pdf>, 2025, accessed: 2025-12-05.
- [3] C. S. Jensen, T. B. Pedersen, and C. Thomsen, "Time series management systems: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 11, pp. 2581–2600, 2017. [Online]. Available: <https://www.semanticscholar.org/paper/Time-Series-Management-Systems-A-Survey-Jensen-Pedersen/el1f2932ac11adb90b3c628e041e28bac2d9a9f5e>
- [4] K. Haines *et al.*, "Survey of time series database technology," UK Natural Environment Research Council, Tech. Rep., 2019, accessed: 2025-12-05. [Online]. Available: <https://nora.nerc.ac.uk/id/eprint/527832/7/N527832CR.pdf>
- [5] N. de Waal, "Literature study: Timeseries databases," Atlarge Research, TU Delft, Tech. Rep., 2022, accessed: 2025-12-05. [Online]. Available: <https://atlarge-research.com/pdfs/2022-dewaal-litsurvey.pdf>
- [6] Tiger Data, "The best time-series databases compared," <https://www.tigerdata.com/learn/the-best-time-series-databases-compared>, 2024, accessed: 2025-12-05.
- [7] InfluxData, "Compare mongodb vs timescaledb," <https://www.influxdata.com/comparison/mongodb-vs-timescaledb/>, 2021, accessed: 2025-12-05.
- [8] Timescale, "How to store time-series data in mongodb and why that's a bad idea," <https://www.tigerdata.com/blog/how-to-store-time-series-data-mongodb-vs-timescaledb-postgresql-a73939734016>, nov 2025, accessed: 2025-12-05.
- [9] Mad Devs, "Managing time-series data: Why timescaledb beats postgresql," <https://maddevs.io/writeups/time-series-data-management-with-timescaledb/>, 2025, accessed: 2025-12-05.
- [10] QueryLeaf, "Mongodb time series collections for iot sensor data management: Real-time analytics and high-performance time-based data processing," <https://www.queryleaf.com/blog/2025/11/23/mongodb-time-series-collections-for-iot-sensor-data-management-real-time-analytics-nov-2025>, accessed: 2025-12-05.
- [11] Timescale, "Time series benchmark suite (tsbs)," <https://github.com/timescale/tsbs>, 2018, accessed: 2025-12-05.
- [12] CrateDB, "Independent time series benchmark confirms cratedb top-tier performance," <https://cratedb.com/blog/independent-time-series-benchmark-confirms-cratedb-top-tier-performance>, 2024, accessed: 2025-12-05.
- [13] Anonymous, "Assessing query execution time and implementational complexity of timescaledb and mongodb for time series," Master's thesis, Dalarna University, 2024, accessed: 2025-12-05. [Online]. Available: <https://www.diva-portal.org/smash/get/diva2:1865472/FULLTEXT01.pdf>
- [14] EMQX, "Mqtt performance benchmark testing: Emqx-mongodb integration," <https://www.emqx.com/en/blog/mqtt-performance-benchmark-testing-emqx-mongodb-integration>, 2023, accessed: 2025-12-05.