SQL vs NoSQL in Cloud OLTP: A Case Study in Systematic Performance Evaluation

Carl Camilleri, Joseph G. Vella, and Vitezslav Nezval
Computer Information Systems, Faculty of ICT, University of Malta, Malta
e-mail: {ccami14 | joseph.g.vella | vitezslav.nezval}@um.edu.mt

***Abstract*—The selection of appropriate DBMS for document-oriented applications significantly impacts system performance and operational efficiency. While prior studies have compared SQL and NoSQL technologies, many focus on narrow use cases or outdated system builds, offering limited guidance for today's multifaceted application requirements. This paper presents a systematic case study comparing PostgreSQL and MongoDB in a cloud-based environment, considering the needs of Cloud OLTP applications. By using an extended YCSB benchmark across 80 scenarios that vary dataset size, workload composition, and concurrency, the study highlights how systematic evaluation reveals nuanced performance trade-offs. PostgreSQL exhibited consistent strengths in mixed and high-concurrency workloads, while MongoDB demonstrated advantages in low-concurrency read-heavy and write-intensive scenarios. The findings underscore the importance of aligning workload characteristics with DBMS capabilities, and illustrate how structured, reproducible evaluation can inform more balanced database selection for Cloud OLTP applications.**

Keywords-Database Management Systems; Performance Evaluation; Cloud OLTP; Relational Databases; Document Databases.

I. Introduction

Information systems (ISs) have become so prevalent that they have become crucial, if not critical, in facilitating the day-to-day human activity.

Systems must meet users' stringent Quality of Service (QoS) expectations. For example, studies indicate that response times of e-Commerce ISs exceeding two seconds can decrease user satisfaction and result in lost business [1], [2].

In tandem, users of transactional ISs also expect a high level of Quality of Data (QoD). For example, the customer experience on an e-Commerce IS will degrade if an order is placed online, only to be subsequently cancelled because it cannot be fulfilled due to depleted stocks. Business can be lost if the e-Commerce retailer cannot publish the latest prices or the newest products: prospective customers will look elsewhere to find the newest and cheapest alternatives.

Effectively, one of the core functions of a transactional IS is in line with the adages "Data is the new oil" [3] and "Data is the soul of the real world" [4]. Prospective customers demand fast access to the most up-to-date version of the dataset.

Data must be stored in some location, typically referred to as a database, and managed by a dedicated database management system (DBMS). Here, QoS and QoD can become competing objectives. Updating the data as fast as possible improves QoD, but this can have a negative impact on QoS: users' operations that read data and which must complete as fast as possible to guarantee the necessary QoS, start competing for hardware resources with write operations that are required for effective QoD.

The proliferation of data-intensive applications has fundamentally transformed the landscape of DBMSs, creating new challenges for system architects in selecting appropriate technologies for workloads sustained by ISs. Modern applications increasingly require systems capable of handling diverse data models, high transaction volumes, and stringent performance requirements while maintaining data consistency and availability. Furthermore, a new class of applications has been defined. "Cloud Online Transaction Processing" ("Cloud OLTP") applications [5] prioritise QoS, without necessarily requiring complex query capabilities and sophisticated transaction models (e.g., ACID).

This evolution has sparked considerable debate between traditional relational database systems and Not Only SQL (NoSQL) alternatives, particularly in scenarios involving document-oriented data structures. Modern applications increasingly demand DBMSs that can efficiently handle semi-structured data while maintaining the reliability and consistency guarantees traditionally found in relational systems. This creates a technology selection dilemma: system architects must choose between mature relational DBMSs with document capabilities and purpose-built document-oriented NoSQL systems, often without sufficient realistic evidence to guide their decisions. Despite the growing body of literature comparing database technologies [6], [7], systematic performance evaluations focusing on document-oriented Cloud OLTP workloads remain scarce. Existing studies often emphasise specific use cases [7] or fail to account for the multi-faceted nature of modern application requirements, including the need for both transactional integrity and query performance across varying load conditions [8]. Furthermore, the rapid evolution of both PostgreSQL and MongoDB technologies means that studies that compare DBMSs have a short shelf-life.

The fundamental challenge addressed in this research centres on conducting a systematic and empirical comparison of DBMS performance for document-oriented transactional workloads. We then use this methodology to specifically investigate the comparative performance characteristics of PostgreSQL and MongoDB when handling applications that require simultaneous support for transactional integrity, high availability, and scalable document management capabilities.

Our research contributes to the existing body of knowledge by presenting a case study that applies established systematic benchmarking practices to evaluating SQL and NoSQL in Cloud OLTP contexts, using PostgreSQL and MongoDB as

examples for SQL and NoSQL technologies, respectively. Our empirical experiments evaluate both systems systematically by: (1) using identical hardware and network conditions in a production-like cloud environment, (2) examining performance across multiple workload compositions and scale factors, (3) analysing both throughput and latency characteristics under varying concurrent load conditions, and (4) providing practical guidance for system architects facing similar technology selection decisions.

We also aim to address some of the limitations identified in prior database comparison studies, including:

- Limited Scenario Coverage: Some studies [7] focus on narrow use cases without comprehensive workload variation, whilst in our work we perform tests along an 80-scenario matrix, hence providing broader coverage.
- Configuration Bias: Other studies [9] acknowledge that database comparison exercises do not ensure equivalent transactional and durability guarantees. We address this systematically, for example by ensuring that our workloads require a majority write concern for MongoDB.

In this context, this research provides:

1) **Benchmark Methodology**: A reproducible, cloud-based experimental approach for evaluating document-oriented database performance in cloud environments.
2) **Empirical Performance Comparison**: Evidence-based guidance for system architects and developers facing similar technology selection decisions.
3) **Empirical Performance Data**: Comprehensive performance measurements comparing PostgreSQL and MongoDB under comparable, realistic conditions.
4) **Workload-Specific Insights**: Detailed analysis of how different workload characteristics affect the relative performance of each system.

Our aim is therefore to advance beyond "which is faster" comparisons toward establishing systematic evaluation approaches that future database management comparison studies can adopt. We identify metrics, such as response time, to allow us to compare very different DBMSs, each using a vanilla configuration that is readily available in realistic setups.

The remainder of this paper is structured as follows: Section II reviews relevant literature on database management performance comparisons and OLTP workload characteristics. Section III describes our experimental methodology, including infrastructure setup and benchmark configuration. Section IV presents detailed results and analysis across different workload scenarios, and discusses the implications of our findings and provides recommendations for practical applications. Finally, Section V concludes with a summary of key contributions and directions for future research.

II. Literature Review

A. Choice of DBMS technology

The choice of DBMS technology significantly impacts application performance, scalability, and operational complexity. This remains a critical technical choice for the success of an IS application.

Choosing the right DBMS technology, or even the right mix of DBMS technologies, from the wide range of available options, is not a trivial exercise and it typically follows a prescribed method.

1) Type of Workload: First, one must identify the type of workload that the DBMS will sustain. Online transactional processing (OLTP) workloads consist of WRITE operations that modify small amounts of data, and READ queries that process a few records and retrieve the majority of the attributes available [10]. Cloud OLTP [5] is similar to OLTP, but each operation affects a single record. In contrast, Online analytical processing (OLAP) workloads typically consist of read-only queries that traverse a large amount of records, performing aggregations and retrieving a small set of attributes [10]. Workloads consisting of both transactional and analytical queries are referred to as Hybrid Transactional and Analytical Processing (HTAP) [11].

In this study, we tackle specifically the case for Cloud OLTP workloads.

2) Data Integrity and Modeling: Second, we must identify the DBMS features that are most pertinent to the application at hand. These may include:

1) **Data integrity**: does the DBMS need to enforce any application-specific operation pre-conditions or rules that determine whether an operation on a data element is accepted?
2) **Data modeling**: which data structures lend themselves best to both store the dataset, and satisfy the data management operations (e.g., READ and WRITE queries) in a manner that the correctness, QoS and QoD requirements of the application are optimisable?

Two of the most popular data models are the relational data model and the document data model. Codd's relational data model [12] popularised DBMSs, and stores data in tables which are matrices of rows and columns. Conversely, the document data model stores data as a series of documents identified by some unique key [13]. Relational DBMSs (RDBMSs) support the former data model, whilst Document-Oriented DBMSs support the latter. Some DBMSs have the capability to support more than one data model, and a popular data format in DBMSs that support the document data model is the JavaScript Object Notation (JSON). Relational DBMSs are typically managed via the Structured Query Language (SQL), a domain-specific language first standardised in the late eighties [14]. Conversely, non-relational DBMSs rely on other languages and are collectively referred to as No-SQL DBMSs [15]

3) Scalability: Another aspect of consideration is the need for horizontal scalability. Systems that require the DBMS to scale horizontally need to opt for a distributed DBMS (DDBMS) [16]. This need can be driven by several objectives, as illustrated in Figure 1. The way data is distributed across several machines for horizontal scalability can also differ, as shown in Figure 2, Figure 3, and Figure 4. The choice of data

distribution strategy affects a DBMS's capability in achieving the requirements for horizontal scalability.

Traditional RDBMSs, exemplified by PostgreSQL[1], have long dominated transactional workloads due to their adoption of ACID[2] guarantees, mature optimisation techniques, and standardised SQL. However, the rise of document-oriented NoSQL databases, such as MongoDB [17], has challenged this dominance by offering flexible schema designs, horizontal scaling capabilities, and optimisations specifically tailored for document-based operations.

The technological dichotomy of RDBMSs and NoSQL databases presents particular challenges for applications managing semi-structured data that can be effectively modeled using either relational or document data models. Such applications often exhibit mixed workload characteristics, combining high-frequency read operations with periodic write-intensive tasks, demanding both transactional consistency and query performance optimisation. The decision between relational and document-oriented approaches becomes further complicated when considering operational factors such as horizontal scalability requirements, data consistency guarantees, and infrastructure complexity.

Recent advances in multi-model database capabilities have blurred traditional boundaries between relational and NoSQL systems. PostgreSQL's native JSON support and document querying capabilities enable it to handle document-oriented workloads effectively, while MongoDB has introduced features to strengthen data consistency guarantees and transactional support. This convergence necessitates empirical evaluation to understand the performance implications of each approach under realistic workload conditions.

B. Functional DBMS Requirements for OLTP

OLTP and Cloud OLTP applications, including those handling document-based data, benefit from several features that a DBMS can offer, namely:

- **Transactional Features** [18], [19]
  - **OLTP Workload Suitability**: The system must handle workloads where data retrieval and modification operations are typically restricted to small sets of records.
  - **Data Integrity Constraints**: The system must provide mechanisms to enforce business rules and maintain data consistency, further ensuring that concurrent operations do not violate application-specific constraints.
  - **Strong Consistency Guarantees**: For critical operations, the DBMS must provide a view where concurrent data changes appear to be applied in a total order across all clients, preventing anomalies in multi-user environments.

[1]https://www.postgresql.org/, Dec 15, 2025
[2]Atomicity, Consistency, Isolation, and Durability

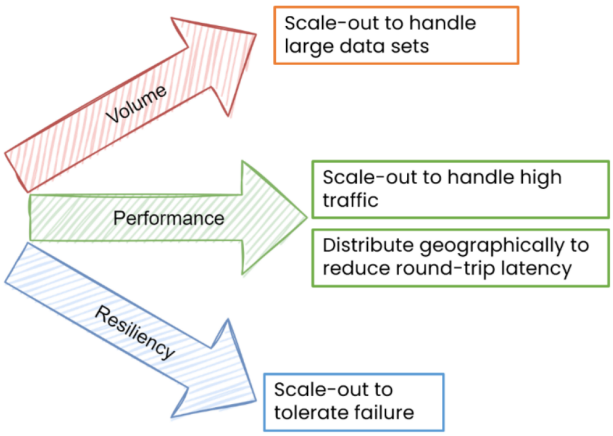

Figure 1: Objectives of DBMS scalability

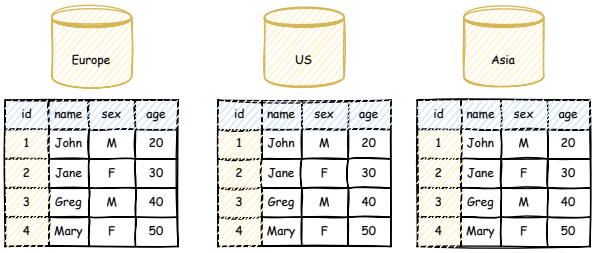

Figure 2: Data Distribution via full Replication (Mirroring)

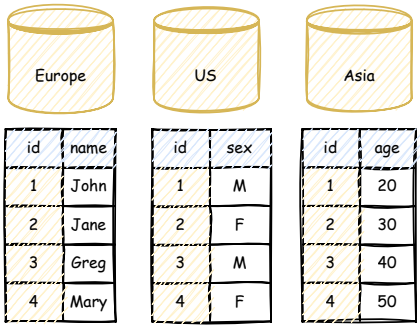

Figure 3: Data Distribution via Vertical Partitioning

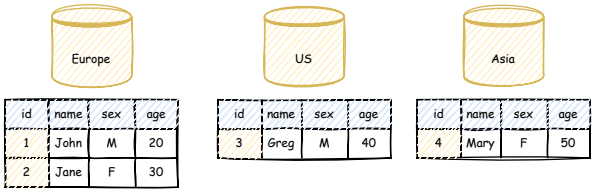

Figure 4: Data Distribution via Sharding (Horizontal Partitioning)

- **Operational Features** [20]–[23]
  - **High Availability**: The system must minimise downtime and provide continuous service availability.
  - **Horizontal Scalability for Performance**: The system should support replication-based scaling to distribute read workloads across multiple nodes, particularly important for applications with varying traffic patterns.
  - **Data Model Flexibility**: The system must efficiently support either relational or document data models, allowing applications to choose the most appropriate representation for their specific use cases.
- **Performance Features** [5]
  - **Fast Read Operations**: Given that many OLTP applications exhibit read-heavy characteristics, the system must optimise for rapid data retrieval operations.
  - **Fast Write Operations**: The system must efficiently handle data modification operations without significantly impacting concurrent read performance.
  - **Mixed Workload Support**: The system should maintain consistent performance across varying ratios of read and write operations.

C. Systematic Assessment Methodology

Although we do not consider our efforts as an exclusive exercise in benchmarking, in our methodology we adopt a strict approach and follow several objectives that are typical of benchmarking. Benchmark design is characterised by several key objectives [24], including:

1) **Relevance**: the assessment should interact with the system under test (SUT) in a manner that is realistic, and thus in-line with the typical interaction that the SUT should expect in a real-world deployment.
2) **Repeatability**: the assessment process should aim for a level of confidence that running the same assessment multiple times would yield similar results.
3) **Fairness**: the assessment should be specific to the domain at hand. An example of an unfair assessment is one that measures the performance of complex queries in RDBMS to elicit a conclusion that a distributed file system has poor performance. Fair assessments also look at multiple qualities of a system under test: for example, focusing solely on performance quality when comparing systems that are functionally different is not considered a fair assessment.
4) **Portability**: it should be possible to execute the assessment against different systems, which implies that one must carefully select the system features to examine, and find a balance between using a small subset of system features (which would render the benchmark design obsolete), and using system-specific or cutting-edge features, which may only be offered by a limited number of SUTs (reducing portability). This quality also emphasises affordability, in that a portable assessment does not necessarily require complex logic or expensive infrastructures.
5) **Understandability**: the assessment should seek to elicit meaningful metrics and the workload should be meaningful to the domain of the SUT.

Benchmarks depend on workload generators to raise requests to SUTs, and different types of workloads exist. Trace-based workloads are configured with a precise set of activities (typically extracted from monitors deployed on a live installation) that are replayed at runtime. In contrast, synthetic workloads generate artificial requests to the SUT, in a manner that the workload mix follows pre-defined probability distributions.

In general, trace-based workloads align better to the repeatability objective of benchmarking however, because they rely on execution traces from live system installations, they are exposed to several challenges including: a) traces may not be long enough to generalise benchmark results; b) it is difficult to obtain and share traces, especially due to security considerations; and c) traces may not include corner cases, preventing benchmarking from testing an SUT's behaviour in atypical situations.

Synthetic workloads can be used to overcome the challenges of trace-based workloads, and several techniques are used to improve their alignment to the benchmarking objectives. For example, a synthetic workload can be designed based on observations from live applications, from which realistic probability distributions that fit live workloads are elicited, hence aligning to the relevance benchmarking objective. Furthermore, one can increase the probability that the actual workload generated aligns to the workload distribution required by running a synthetic workload over a longer period, hence improving the aspect of repeatability. Furthermore, benchmark results are assessed based on repeated iterations to improve repeatability by, for example, minimising the impact of transient external events (e.g., transient events in a cloud infrastructure). Consequently, several studies report average values based on the throughput of three benchmark workload executions [9], [25].

Synthetic workloads are a popular choice in diverse benchmarking studies, including benchmarks for runtime verification [26], for serverless cloud computing [27], and programming frameworks [28]. Synthetic workloads are also prevalent in DBMS benchmarks. The Transaction Processing Performance Council (TPC) puts forward the specifications of several synthetic benchmarks, each modelled differently to remain relevant and representative for different domains. For example, the TPC-C benchmark [29] is widely used to assess the performance of transactional DBMSs [30]. The Yahoo! Cloud Serving Benchmark [5], or YCSB, is a synthetic workload generator, built on the basis of observations of typical web serving use cases at Yahoo!, and is used to study the performance of cloud data serving systems, such as DDBMSs [31]–[34].

III. Implementation

A. Requirements and Systems Analysis

This experiment addresses the following key questions:

1) How do PostgreSQL and MongoDB compare in terms of transaction throughput for document-oriented Cloud OLTP workloads across different read/write ratios?
2) What are the latency[3] characteristics of read and write operations for both systems under varying concurrent load conditions?
3) How do system metrics scale with increasing dataset sizes and concurrent user loads for both PostgreSQL and MongoDB?
4) Under what specific workload conditions does each system demonstrate superior throughput, and what factors contribute to these differences?
5) What practical considerations should guide the selection between PostgreSQL and MongoDB for applications with similar functional requirements?

B. Scope and Limitations

This exercise focuses specifically on document-oriented workloads using the document data model capabilities of both systems. The evaluation encompasses varying workload compositions (read/write ratios), dataset sizes, and concurrent user loads using standardised benchmarking methodologies.

The experiment is constrained to specific versions of PostgreSQL and MongoDB and to a single configuration of each DBMS deployed in cloud-based Database-as-a-Service (DBaaS) environments. Various other valid configurations for each DBMS were beyond the scope of this study. The evaluation uses synthetic workloads generated by the YCSB framework [5], which may not capture all nuances of real-world application patterns. Additionally, the study focuses on performance metrics and does not extensively evaluate factors such as administrative complexity, development productivity, or long-term maintenance costs.

C. Assessment Objectives

This study aims to address specific research objectives. The primary objective is to conduct a comprehensive empirical performance comparison between PostgreSQL and MongoDB for document-oriented Cloud OLTP workloads, evaluating their relative strengths and weaknesses across different operational scenarios. Other secondary objectives include:

1) **Throughput Analysis**: Quantify the transaction processing capabilities of both systems under varying workload compositions and concurrent user loads.
2) **Latency Characterisation**: Measure and compare the response time characteristics for both read and write operations across different system configurations.
3) **Scalability Assessment**: Evaluate how both systems perform as dataset sizes and concurrent user loads increase.
4) **Workload Sensitivity Analysis**: Determine how changes in read/write ratios affect the relative performance of each system.
5) **Practical Guidance Development**: Provide evidence-based recommendations for system architects selecting between these technologies for specific application contexts. We thus aim to carry out this exercise in a way that could prove useful and familiar to system architects faced with the question "Which DBMS should I use?".

[3]We define *latency* as the time elapsed between when a client submits an operation to when the result of that operation arrives back at the client

D. Methodology Overview

This empirical analysis adopts a rigorous experimental approach to evaluate the comparative performance of PostgreSQL and MongoDB for document-oriented Cloud OLTP workloads, and is carried out in the following context:

- The analysis focuses on the Document data model. This is the only data model that is natively supported by both SUTs and thus, in the spirit of fairness, is the only one considered.
- Analysis is performed using the YCSB tool, which has been used in other DBMS empirical performance analyses [35], [36]. YCSB already supports PostgreSQL and MongoDB. However, for PostgreSQL, it provided support for simulating a workload by a single client (i.e., thread). For our use case, we extended YCSB with a client-side connection pooler, and in doing so enabled the simulation of workloads via multiple clients. Our extension was also proposed on the official YCSB GitHub repository [4].
- The SUTs were deployed in DBaaS mode using Aiven for PostgreSQL and Atlas for MongoDB.
- MongoDB clients connected with the majority write concern, in an effort to approximate the durability guarantees of PostgreSQL.

E. System Under Test Configuration

Figure 5 illustrates the SUT infrastructure setup used for this empirical performance evaluation. The configuration of the two DBMSs chosen for evaluation is given in Table I.

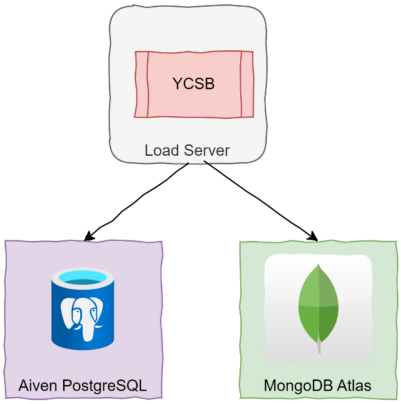

Figure 5: SUT Infrastructure Setup

Both systems were deployed in Microsoft Azure's France Central region to ensure consistent network latency and infrastructure characteristics. The DBaaS approach was chosen to reflect realistic production deployment scenarios, aligned with our *practical guidance development* objective, while

[4]https://github.com/brianfrankcooper/YCSB/pull/1709, Oct 18, 2025

TABLE I: Database Management System Specifications

Specification	PostgreSQL	MongoDB
Version	PostgreSQL v16.2	MongoDB v7.0
Deployment	Aiven PostgreSQL (DBaaS)	MongoDB Atlas (DBaaS)
Service Plan	Business-64	M60 (low CPU optimisation)
Configuration	2 servers (primary-secondary)	3 servers (1 primary, 2 replicas)
Hardware	8 vCPUs, 64 GiB RAM/server	8 CPU cores, 64GB RAM/server
Storage	Premium SSD v1, 1000GB	1024GB, 5000 IOPS (192 MB/s)
Connection Limit	1000	32000
Network Performance	–	Extremely High
Backup	–	Continuous Cloud Backup

eliminating infrastructure management variables that could affect performance measurements.

The synthetic workload generation was performed using a dedicated load server with the following specifications:

- Cloud Platform: Microsoft Azure
- Machine Type: D32s_v5
- CPU: 32 vCPUs
- Memory: 128GB RAM
- Storage: 80GB SSD
- Operating System: Linux Ubuntu
- Network: Co-located in France Central region

This configuration provided sufficient resources to generate high-concurrency workloads without causing client-side bottlenecks that could skew performance measurements.

F. Empirical Performance Evaluation Framework

The characteristics of the dataset generated by YCSB were:

- Data Model: Document-based (JSON format)
- Record Count: 1,000,000 documents for database seeding
- Document Structure: Semi-structured data typical of document-oriented applications, generated using the default YCSB data load generator (i.e., JSON documents with ten properties of 100 bytes each).
- Key Distribution: Zipfian distribution to simulate realistic access patterns

To ensure fair evaluation, the standard YCSB framework was extended with the following enhancements:

- Client-side connection pooling for PostgreSQL to enable multi-threaded workload simulation
- Optimised database drivers for both systems
- Enhanced metrics collection for latency analysis

The PostgreSQL extension addressing single-client limitations was contributed to the official YCSB repository[5], ensuring reproducibility and community benefit.

G. Experimental Design

The experimental design evaluated system performance across multiple workload dimensions to capture realistic application scenarios:

Read/Write Ratios: Four distinct workload compositions were tested to represent different application characteristics:

- 100% READ / 0% WRITE: Read-only analytical queries
- 95% READ / 5% WRITE: Read-heavy with occasional updates
- 80% READ / 20% WRITE: Balanced read-write workload
- 70% READ / 30% WRITE: Write-intensive transactional workload

Dataset Scaling: Performance evaluation was conducted across four dataset sizes to assess scalability characteristics:

- 10,000 records: Small dataset (baseline)
- 100,000 records: Medium dataset
- 500,000 records: Large dataset
- 1,000,000 records: Maximum dataset size

Concurrency Levels: Five concurrent user load levels were tested to evaluate system behaviour under varying contention, defined by virtual users (vUsers):

- 60 vUsers: Low concurrency (baseline)
- 120 vUsers: Moderate concurrency
- 240 vUsers: High concurrency
- 480 vUsers: Very high concurrency
- 960 vUsers: Maximum concurrency stress test

The complete experimental design resulted in 80 unique scenarios per database system (i.e., 4 workload types × 4 dataset sizes × 5 concurrency levels). Each scenario was executed three times to ensure measurement consistency and reduce the impact of transient system variations, resulting in 240 individual test runs per system and 480 total benchmark executions.

Each benchmark run was conducted for a duration of 60 seconds to allow for system stabilisation and meaningful performance measurement collection. The total experimental execution required approximately eight hours of continuous benchmarking.

H. Database-Specific Configurations

PostgreSQL was configured to optimise document-oriented workload performance while maintaining ACID compliance. **Data storage** uses native JSONB format for document storage and indexing. **Connection Management** uses built-in connection pooling with optimised parameters. The **Consistency Level** retains full ACID compliance with default Read Committed isolation level. **Replication** to read replicas is asynchronous.

MongoDB was also configured with production-recommended settings for transactional workloads. **Write Concern** was configured as Majority write for durability

[5]https://github.com/brianfrankcooper/YCSB/pull/1709, Oct 18 2025

guarantees. **Read Preference** is given to the Primary for consistency. **Connection Management** uses native connection pooling and multiplexing. **Replication** to the replica set requires majority acknowledgment. The majority write concern configuration was specifically chosen to approximate the durability and consistency guarantees provided by PostgreSQL, ensuring fair and meaningful comparison between systems.

We acknowledge that both DBMSs chosen for this exercise are intrinsically very different technologies, and as such it is not trivial to achieve a precise like-for-like configuration. However, we chose a configuration for each DBMS that is considered familiar to system architects and application developers, in line with our objectives.

I. Metrics Collection

Performance evaluation focused on two primary metrics categories. **Throughput Metrics** include Operations per second (overall system throughput), Read operations per second, Write operations per second and Transaction completion rates. **Latency Metrics** include Average response time for read operations, Average response time for write operations, 95th percentile latency measurements and Maximum observed latencies under load. All metrics were collected using YCSB's built-in measurement framework, with additional custom instrumentation for detailed latency analysis across different concurrency levels and workload compositions.

J. Data Analysis Approach

Performance analysis employed comparative statistical methods to identify significant performance differences between systems. Results were aggregated across multiple test runs to ensure statistical reliability, with percentage-based comparisons used to normalise performance differences across varying absolute throughput levels.

The analysis framework generated comprehensive performance profiles for each system across all tested scenarios, enabling identification of workload-specific performance characteristics and optimal use case recommendations.

IV. Results

The empirical performance evaluation across 80 distinct scenarios per database system reveals significant performance variations between PostgreSQL and MongoDB under different workload conditions. Figure 6 presents a summary of results as percentage differences in average throughput between MongoDB and PostgreSQL, where positive values indicate MongoDB superiority and negative values indicate PostgreSQL superiority. Figures 7, 8, 9, 10, and 11 illustrate in detail the performance profile of both SUTs under each scenario.

The empirical results demonstrate that PostgreSQL consistently outperforms MongoDB in the majority of tested scenarios, particularly excelling in mixed read-write workloads. However, MongoDB exhibits competitive advantages in specific operational contexts, notably under write-heavy workloads with maximum server saturation and read-heavy workloads under low concurrent user loads.

A. Throughput Analysis

1) Overall Performance Characteristics: Across all tested scenarios, PostgreSQL demonstrates superior throughput performance in approximately 75% of cases, with performance advantages ranging from 20% to 40% compared to MongoDB. This performance superiority is most pronounced in scenarios involving mixed workload compositions, where PostgreSQL's optimised query execution and transaction management provide significant advantages.

The performance difference between systems varies substantially based on workload characteristics:

- **Read-Only Workloads (100% READ)**: PostgreSQL shows 15-35% better performance under medium to high concurrency levels (i.e., 240-960 virtual users)
- **Read-Heavy Workloads (95% READ/5% WRITE)**: PostgreSQL maintains 10-30% performance advantage across most scenarios
- **Balanced Workloads (80% READ/20% WRITE)**: PostgreSQL demonstrates 20-40% superior performance consistently
- **Write-Intensive Workloads (70% READ/30% WRITE)**: PostgreSQL shows 15-25% better performance, with exceptions under maximum load conditions

2) MongoDB Performance Advantages: MongoDB demonstrates competitive advantages in specific scenarios:

Write-Heavy, High-Concurrency Scenarios: MongoDB exhibits approximately 20% better throughput performance compared to PostgreSQL under write-intensive workloads (i.e., 70% READ/30% WRITE) when operating at maximum concurrent user loads (960 virtual users). This advantage likely stems from MongoDB's optimised write handling and connection management under high contention scenarios.

Read-Heavy, Low-Concurrency Scenarios: Under read-dominated workloads (i.e., 100% READ) with low concurrent user loads (i.e., 60-120 virtual users), MongoDB demonstrates 30-40% superior performance compared to PostgreSQL. This advantage diminishes as concurrency increases, suggesting that MongoDB's document-oriented query processing provides benefits primarily under low-contention conditions.

B. Latency Analysis

1) Read Operation Latency: The latency analysis reveals distinct performance characteristics for read operations between the two SUTs:

MongoDB Read Performance: MongoDB consistently demonstrates superior read operation latency across all tested scenarios, with average improvements of approximately 5ms compared to PostgreSQL. This advantage remains relatively stable across different concurrency levels and dataset sizes, suggesting fundamental differences in document retrieval optimisation.

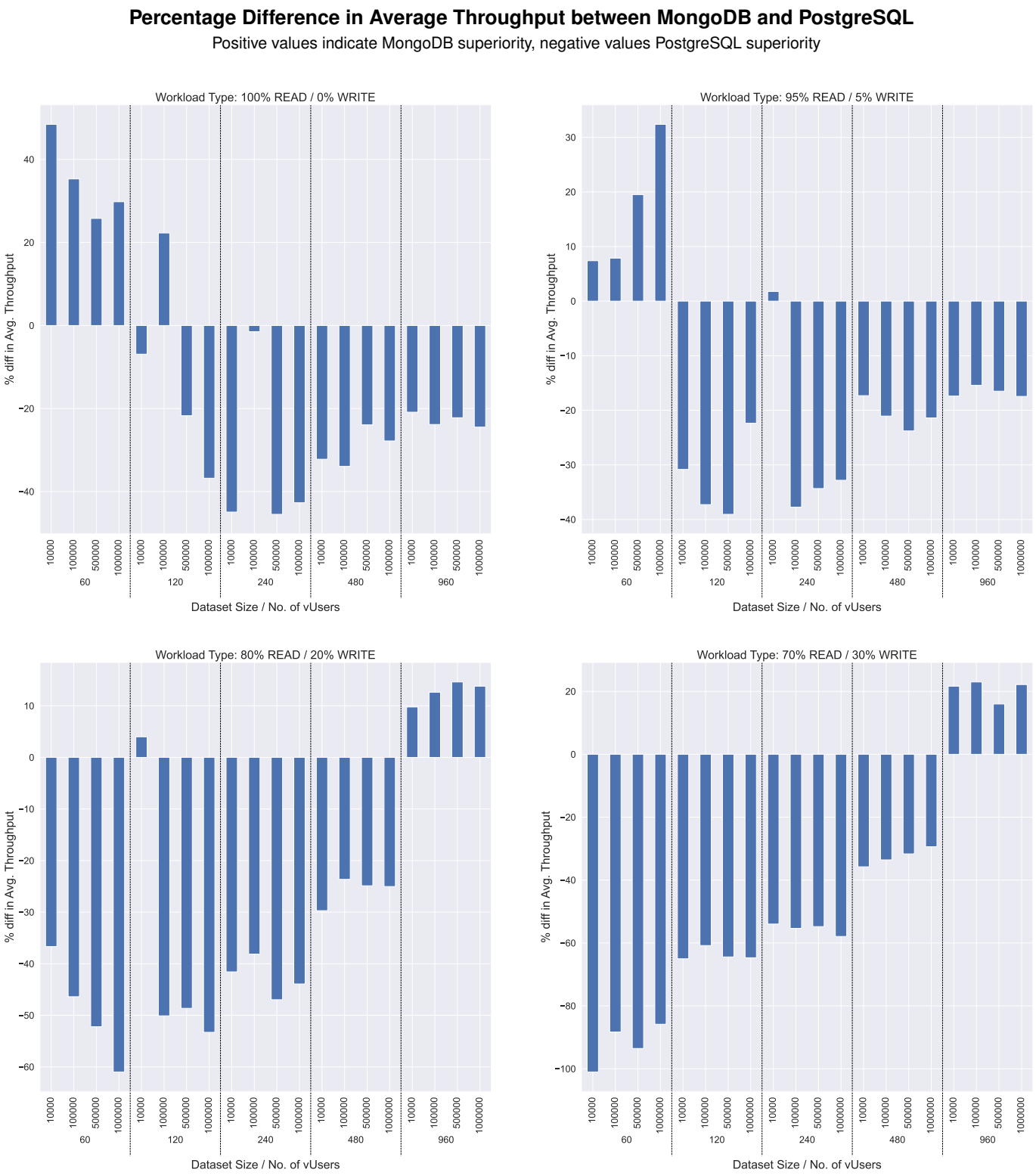

Figure 6: Summary of results as a Percentage Difference in Average Throughput between MongoDB and PostgreSQL with different workloads, dataset sizes and number of virtual users

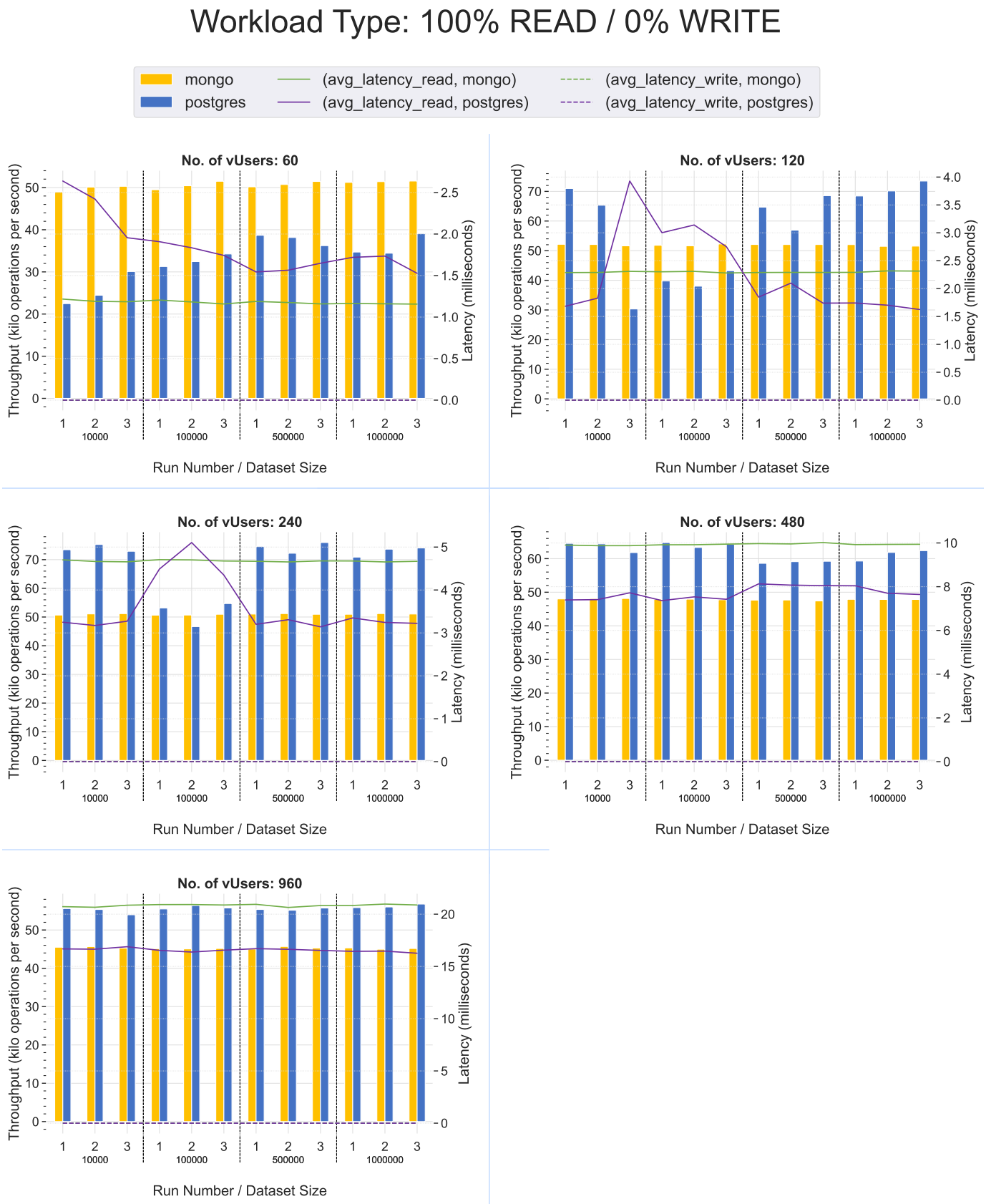

Figure 7: Detailed Throughput and Latency Results for the scenario 100% READ/ 0% WRITE

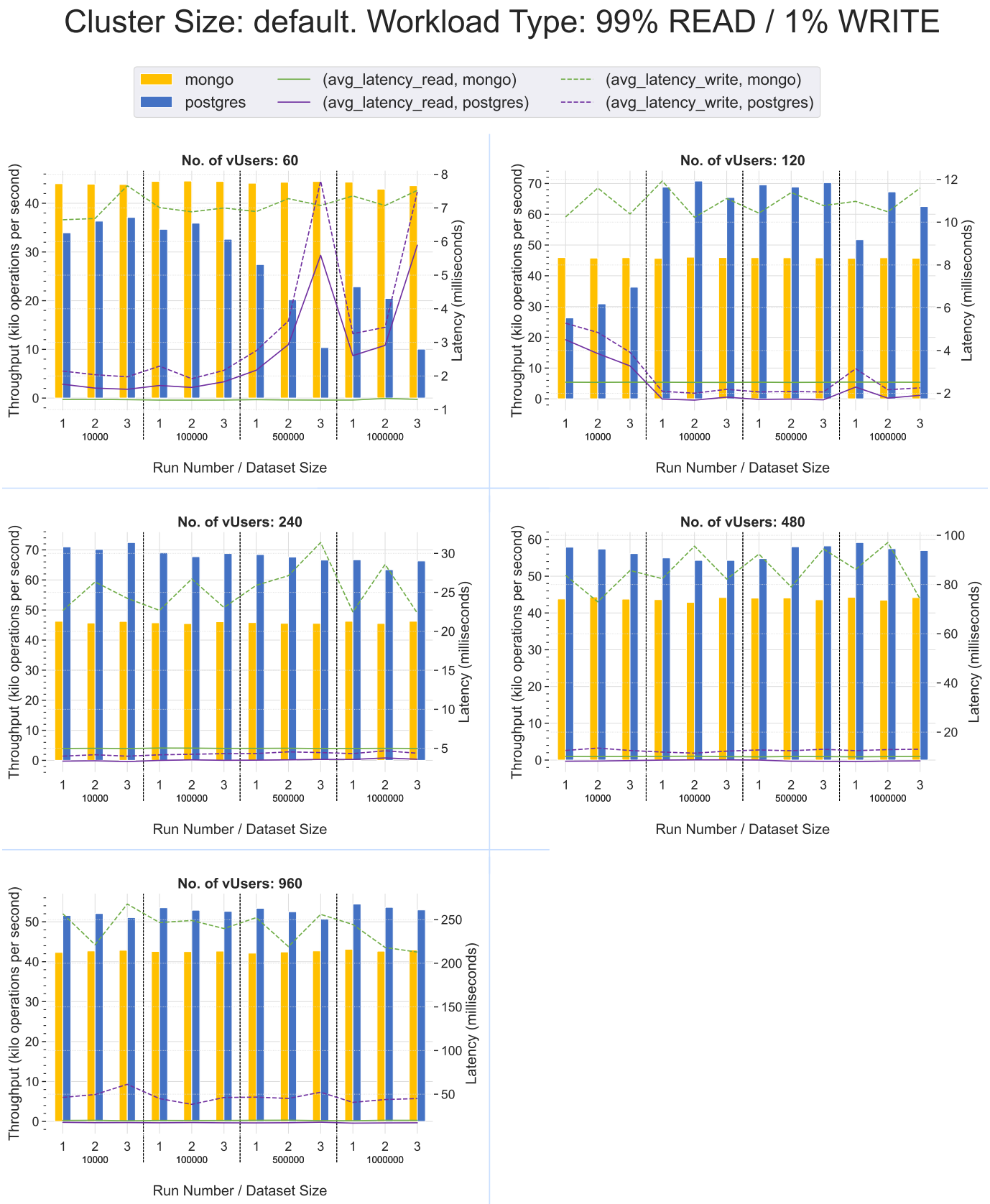

Figure 8: Detailed Throughput and Latency Results for the scenario 99% READ/ 1% WRITE

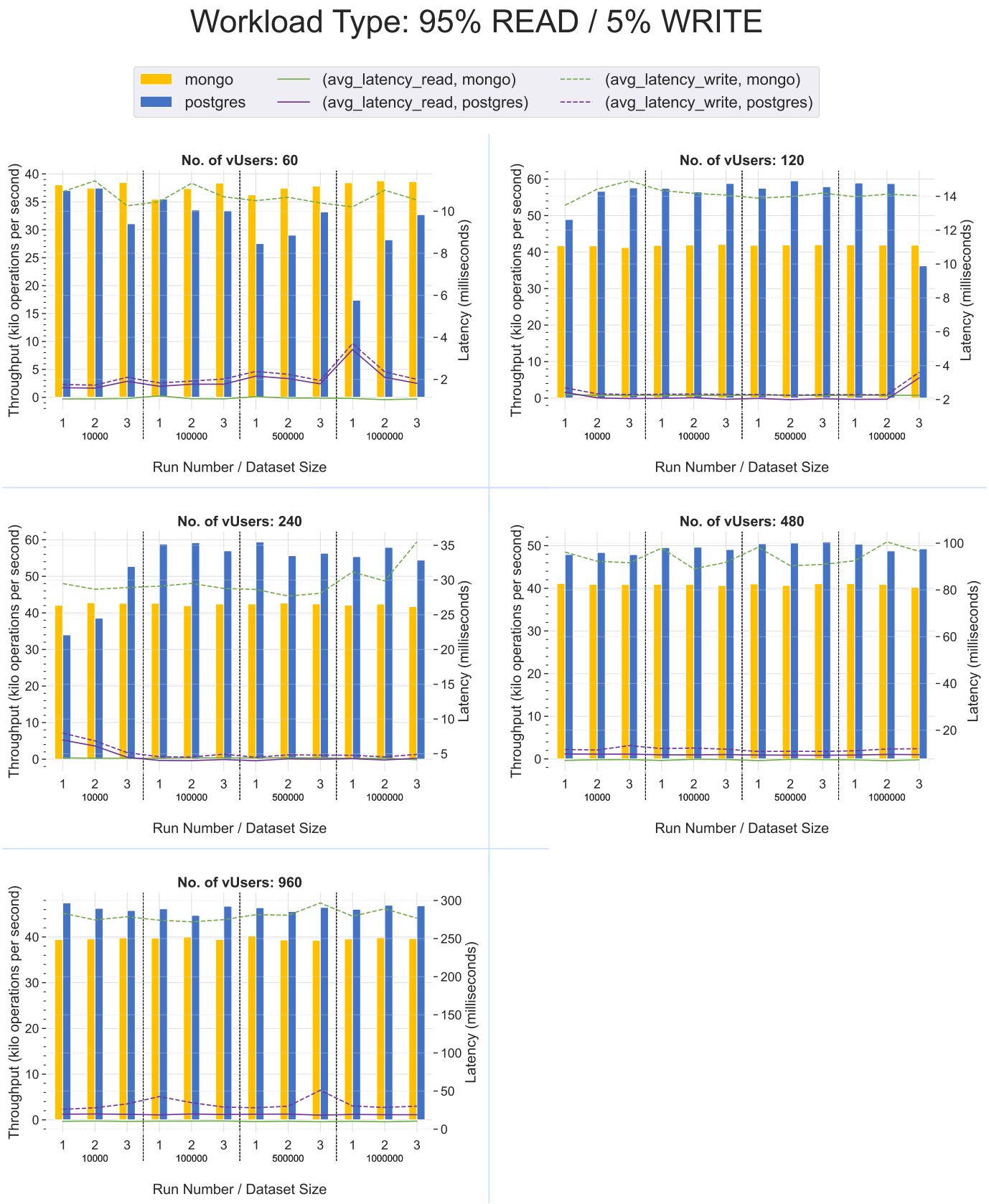

Figure 9: Detailed Throughput and Latency Results for the scenario 95% READ/ 5% WRITE

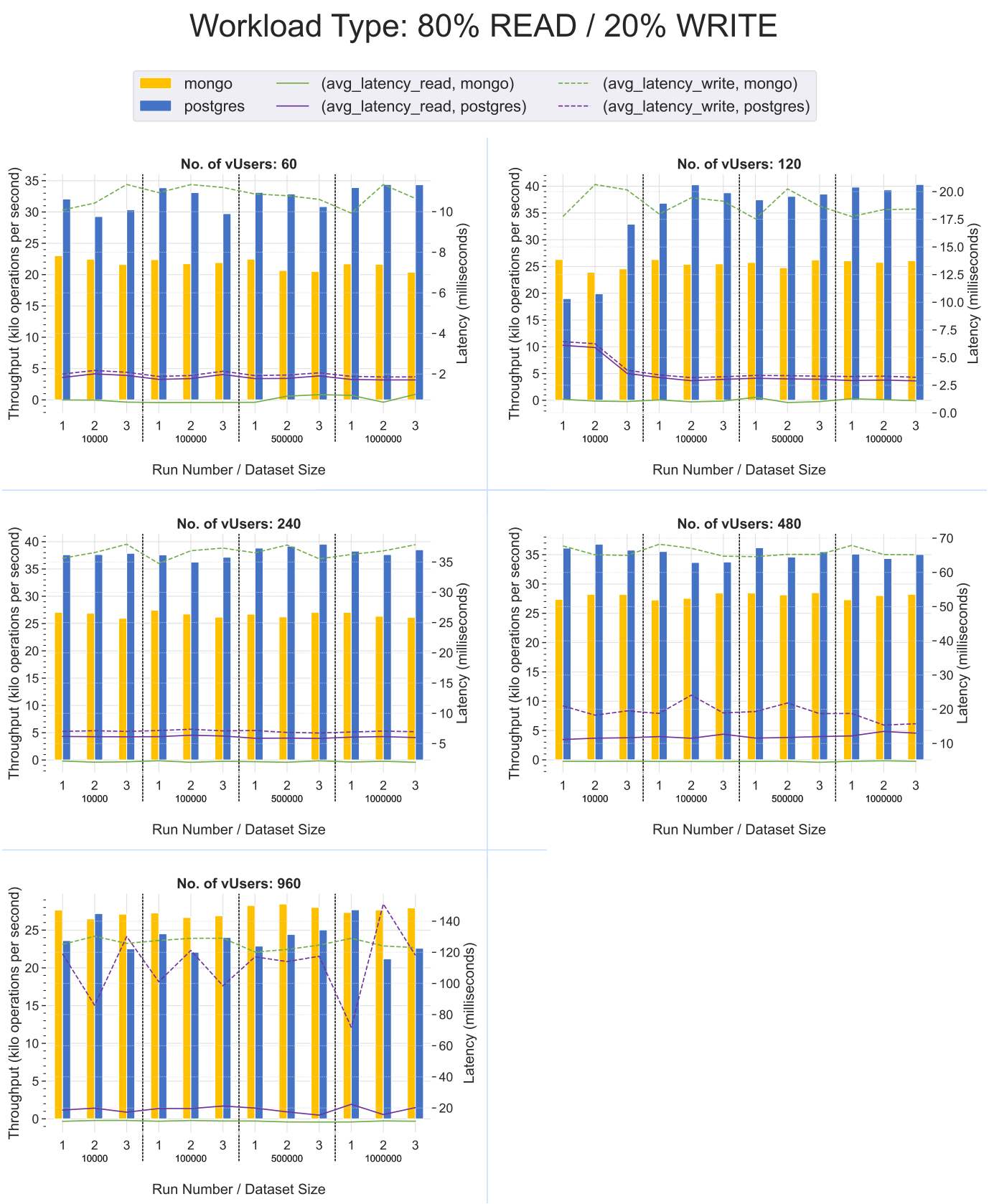

Figure 10: Detailed Throughput and Latency Results for the scenario 80% READ/ 20% WRITE

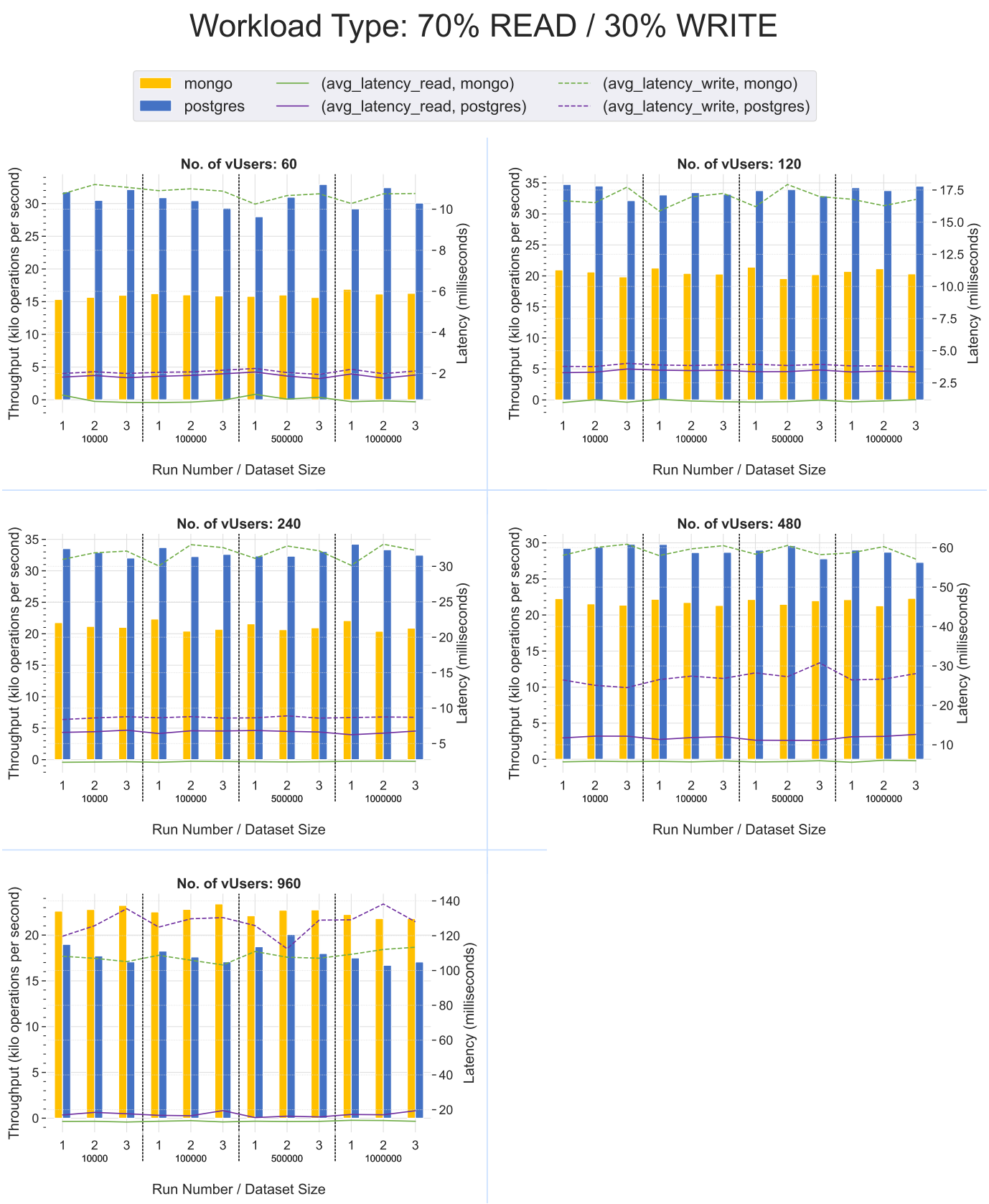

Figure 11: Detailed Throughput and Latency Results for the scenario 70% READ/ 30% WRITE

PostgreSQL Read Performance: Although PostgreSQL exhibits slightly higher read latencies than MongoDB, the difference remains minimal (5ms average) and does not significantly impact overall application responsiveness in most practical scenarios.

2) Write Operation Latency: Write operation performance reveals the most significant performance difference between the systems.

PostgreSQL Write Performance: PostgreSQL demonstrates substantially superior write operation performance across all tested scenarios. Write latencies remain consistently low even under high concurrent loads, maintaining sub-50ms response times in most configurations.

MongoDB Write Performance: MongoDB exhibits higher write operation latencies, particularly under high concurrency conditions. The most severe performance degradation occurs in write-intensive scenarios with maximum concurrent users, where MongoDB write latencies can exceed PostgreSQL by up to 250ms. For example, in the 95% READ/5% WRITE scenario with 960 virtual users, MongoDB write operations average 250ms longer than equivalent PostgreSQL operations.

C. Scalability Analysis

1) Dataset Size Scaling: Both systems demonstrate reasonable scalability characteristics as dataset sizes increase from 10,000 to 1,000,000 records, whilst using the same infrastructure and DBMS configuration.

PostgreSQL Scaling: PostgreSQL maintains consistent performance across dataset size variations, with throughput degradation remaining below 15% as dataset size increases by two orders of magnitude. This consistent performance suggests effective query optimisation and indexing strategies for document-based operations.

MongoDB Scaling: MongoDB exhibits similar dataset scaling characteristics, with performance degradation comparable to PostgreSQL across different dataset sizes. However, the absolute performance remains lower than PostgreSQL in most scenarios.

2) Concurrency Scaling: The concurrent user load analysis reveals different scaling patterns for each system:

PostgreSQL Concurrency Handling: PostgreSQL demonstrates consistent performance scaling as concurrent user loads increase, maintaining stable throughput levels even at maximum concurrency (i.e., 960 virtual users). The system exhibits graceful degradation without performance cliffs or severe bottlenecks.

MongoDB Concurrency Handling: MongoDB shows more variable performance under increasing concurrency loads. While the system handles low to moderate concurrency effectively, performance becomes inconsistent at high concurrency levels (i.e., throughput does not increase when concurrency increases), particularly for write-intensive operations.

D. Workload Composition Sensitivity

1) Read/Write Ratio Impact: The analysis reveals significant sensitivity to workload composition for both systems:

Read-Dominant Workloads: As read operations dominate the workload (moving from 70% to 100% read operations), MongoDB's relative performance improves, particularly under low concurrency conditions. However, PostgreSQL maintains superior absolute performance in most scenarios.

Write-Intensive Workloads: Increasing write operation percentages generally favour PostgreSQL due to its superior write latency characteristics. The performance gap between systems widens as write operations become more prevalent, except under maximum concurrency conditions where MongoDB shows some advantages.

E. Statistics Reporting

All reported performance differences are based on the mean of three independent benchmark runs per scenario, improving reporting reliability. The consistent patterns observed across multiple test runs and scenario variations provide confidence in the robustness of the observed performance characteristics.

The most significant findings, including PostgreSQL's 20-40% throughput advantages and MongoDB's 250ms write latency penalties under high concurrency, were consistently observed across all repetitions, indicating systematic rather than random performance differences.

F. Performance Thresholds

Both database systems demonstrated the capability to handle substantial transaction loads:

- **Maximum Observed Throughput**: Both systems achieved over 20,000 transactions per second under optimal conditions
- **Sustained Performance**: Both systems maintained over 15,000 transactions per second across most scenarios
- **Baseline Performance**: Even under adverse conditions, both systems sustained minimum throughput levels exceeding 10,000 transactions per second

These performance levels significantly exceed typical application requirements and demonstrate the suitability of both systems for high-throughput Cloud OLTP applications.

G. Summary of Key Findings

The comprehensive performance evaluation yields the following primary observations:

1) **PostgreSQL General Superiority**: PostgreSQL demonstrates superior performance in approximately 75% of tested scenarios, with advantages ranging from 20-40%
2) **MongoDB Niche Advantages**: MongoDB shows competitive performance in read-heavy, low-concurrency scenarios (30-40% advantage) and write-heavy, high-concurrency scenarios (20% advantage)
3) **Latency Trade-offs**: MongoDB offers superior read latency (5ms improvement) while PostgreSQL provides substantially better write latency (up to 250ms improvement under high load)
4) **Scalability Characteristics**: Both systems demonstrate reasonable scaling behaviour across dataset sizes and concurrency levels

5) **High Absolute Performance**: Both systems exceed 20,000 transactions per second, indicating suitability for demanding Cloud OLTP applications

V. Conclusion

This study identified the intricacies of delivering the QoS and QoD required by users of an IS sustaining a transactional workload, under the characteristics of Cloud OLTP [5].

We discussed several of the many characteristics of a DBMS that have an important bearing on the choice of product for a particular use-case. We then walked through the use-case considered in our problem domain, and briefly described how and why MongoDB and PostgreSQL are a good fit. Lastly, we detailed a thorough performance comparison of a representative setup of these two DBMSs, using a workload suitable for the document data model. Results for this comparison exercise show that PostgreSQL outperforms MongoDB in the general case, but MongoDB performs faster in other cases.

This exercise brings forth several observations, including that:

- An optimal choice of DBMS technology depends on a thorough understanding of the problem domain at hand, ensuring that the DBMS delivers the necessary functions to support the dataset, the operations that will be performed on it, and the system and business requirements that need to be satisfied.
- NoSQL DBMSs are not necessarily faster than SQL-based DBMSs in all cases. Our empirical analysis has shown that both MongoDB and PostgreSQL can meet stringent QoS and QoD requirements.
- The data model should not be the only deciding factor as to whether to use an SQL-based or NoSQL DBMS: many SQL-based DBMSs support multiple data models, including the document data model used by NoSQL DBMSs.
- Deploying a distributed DBMS requires careful thought as to which level of data consistency and durability guarantees are needed and expected from the DBMS. For example, MongoDB does not guarantee that cross-document changes occurring in the master node are replicated in the same order to read-only replicas[6], whilst PostgreSQL's replication methodology based on the write-ahead log (WAL) does provide these guarantees[7].

Nonetheless, the rigorous approach of this effort is not considered to constitute a thorough analysis of this non-trivial topic. Notably, the comparison was done on a single, representative setup of PostgreSQL and MongoDB available at the time of writing. Similar exercises can be carried out on several other setups, and we identify five areas that represent limitations of this study and where future research efforts can be directed:

1) *Configuration Scope:* This study examined one representative configuration per system. Future work should explore the impact of different consistency levels, storage engines, and optimisation parameters, e.g., different configurations of READ and WRITE concerns for MongoDB, the impact of READs handled by slave nodes and WRITEs handled by master nodes and weaker data consistency levels such as Eventual Consistency and Causal Consistency [37]–[39].
2) *Workload Patterns:* YCSB workloads, while standard, may not capture all real-world access patterns. Industry-specific benchmarks (e.g., e-Commerce, IoT telemetry) would introduce workload and dataset variety (e.g., varying amounts of data records, varying document sizes) and thus broaden applicability.
3) *Version Evolution:* DBMSs evolve rapidly, so performance comparisons should be conducted periodically to track performance evolution across versions.
4) *Hardware Diversity:* Cloud-based evaluation provides consistency but may not reflect performance on specialised hardware configurations.
5) *Operational Complexity:* Performance is only one dimension of database selection. Future framework extensions should incorporate operational metrics, development productivity, and total cost of ownership.

[6]https://www.mongodb.com/docs/manual/core/replica-set-sync/#multithreaded-replication, accessed Nov 9, 2025

[7]https://www.postgresql.org/docs/current/protocol-replication.html, accessed Nov 9, 2025

References

[1] Y. Lee, A. N. Chen, and V. Ilie, "Can online wait be managed? The effect of filler interfaces and presentation modes on perceived waiting time online," *MIS Quarterly*, pp. 365–394, 2012.

[2] S. Papastavrou, P. K. Chrysanthis, and G. Samaras, "Performance vs. freshness in web database applications," *World wide web*, vol. 17, no. 5, pp. 969–995, 2014.

[3] C. Humby, "Data is the new oil," *Proc. ANA Sr. Marketer's Summit. Evanston, IL, USA*, vol. 1, 2006.

[4] M. Xiaolei, L. Sen, and Y. Xiaofei, "Traffic Data Management Technology in ITS," *Intelligent Road Transport Systems*, p. 97, 2022.

[5] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ACM, 2010, pp. 143–154.

[6] M. Salahuddin, S. Majeed, S. Hira, and G. Mumtaz, "A Systematic Literature Review on Performance Evaluation of SQL and NoSQL Database Architectures," *Journal of Computing & Biomedical Informatics*, vol. 7, no. 02, 2024.

[7] A. Makris, K. Tserpes, G. Spiliopoulos, D. Zissis, and D. Anagnostopoulos, "MongoDB Vs PostgreSQL: A comparative study on performance aspects," *GeoInformatica*, vol. 25, no. 2, pp. 243–268, 2021.

[8] T. Taipalus, "Database management system performance comparisons: A systematic literature review," *Journal of Systems and Software*, vol. 208, p. 111 872, 2024.

[9] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, and C. Matser, "Performance evaluation of NoSQL databases: a case study," in *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*, 2015, pp. 5–10.

[10] M. Bach and A. Werner, "Hybrid column/row-oriented DBMS," in *Man–Machine Interactions 4: 4th International Conference on Man–Machine Interactions, ICMMI 2015 Kocierz Pass, Poland, October 6–9, 2015*, Springer, 2015, pp. 697–707.

[11] C. Camilleri, J. G. Vella, and V. Nezval, "HTAP With Reactive Streaming ETL," *Journal of Cases on Information Technology (JCIT)*, vol. 23, no. 4, pp. 1–19, 2021.

[12] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[13] H. Hashem and D. Ranc, "Evaluating NoSQL document oriented data model," in *IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, IEEE, 2016, pp. 51–56.

[14] *ISO 9075:1987 - information processing systems — database language — SQL*, https://www.iso.org/standard/16661.html, Accessed: 21-May-2024, 1987.

[15] C. Strozzi, "NoSQL: A relational database management system," *Lainattu*, vol. 5, p. 2014, 1998.

[16] C. Camilleri, J. G. Vella, and V. Nezval, "D-Thespis: A Distributed Actor-Based Causally Consistent DBMS," in *Transactions on Large-Scale Data-and Knowledge-Centered Systems LIII*, Springer, 2023, pp. 126–165.

[17] DoubleClick, ShopWiki, and GiltGroupe, *Mongodb*, https://www.mongodb.org, https://www.mongodb.org, 2007.

[18] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, 7th ed. Pearson/Addison Wesley, 2017.

[19] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi, "Fast distributed transactions and strongly consistent replication for OLTP database systems," *ACM Transactions on Database Systems (TODS)*, vol. 39, no. 2, pp. 1–39, 2014.

[20] K. Domdouzis, P. Lake, and P. Crowther, *Concise guide to databases: A practical introduction*. Springer, 2021.

[21] P. Lake and P. Crowther, "Database Availability," in *Concise Guide to Databases: A Practical Introduction*, Springer, 2013, pp. 221–239.

[22] Z. H. Liu and D. Gawlick, "Management of Flexible Schema Data in RDBMSs-Opportunities and Limitations for NoSQL," in *CIDR*, 2015.

[23] C. Camilleri, J. G. Vella, and V. Nezval, "Thespis: Causally-consistent OLTP," in *2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS)*, IEEE, 2021, pp. 261–269.

[24] D. Bermbach, E. Wittern, and S. Tai, *Cloud service benchmarking*. Springer, 2017.

[25] S. Chowdhury and W. Golab, "A scalable recoverable skip list for persistent memory," in *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, 2021, pp. 426–428.

[26] L. Aceto, D. P. Attard, A. Francalanza, and A. Ingólfsdóttir, "On benchmarking for concurrent runtime verification," in *International Conference on Fundamental Approaches to Software Engineering*, Springer International Publishing Cham, 2021, pp. 3–23.

[27] M. Copik, G. Kwasniewski, M. Besta, M. Podstawski, and T. Hoefler, "Sebs: A serverless benchmark suite for function-as-a-service computing," in *Proceedings of the 22nd International Middleware Conference*, 2021, pp. 64–78.

[28] C. Camilleri, J. G. Vella, and V. Nezval, "Actor model frameworks: An empirical performance analysis," in *International Conference on Information Systems and Management Science*, Springer, 2022, pp. 461–472.

[29] F. Raab, "TPC-C - The Standard Benchmark for Online Transaction Processing (OLTP)," in *The Benchmark Handbook for Database and Transaction Systems (2nd Edition)*, Morgan Kaufmann Publishers Inc, 1993.

[30] N. Crooks, M. Burke, E. Cecchetti, S. Harel, R. Agarwal, and L. Alvisi, "Obladi: Oblivious serializable transactions in the cloud," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 727–743.

[31] S. A. Mehdi, C. Littley, N. Crooks, L. Alvisi, N. Bronson, and W. Lloyd, "Not Causal! Scalable Causal Consistency with No Slowdown Cascades," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 453–468.

[32] K. Shudo and T. Yaguchi, "Causal consistency for distributed data stores and applications as they are," in *IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, IEEE, vol. 1, 2016, pp. 602–607.

[33] M. Zawirski, "Dependable eventual consistency with replicated data types," Ph.D. dissertation, Universite Pierre et Marie Curie, 2015.

[34] C. Camilleri, J. G. Vella, and V. Nezval, "Horizontally Scalable Implementation of a Distributed DBMS Delivering Causal Consistency via the Actor Model," *Electronics*, vol. 13, no. 17, p. 3367, 2024.

[35] S. Ferreira, J. Mendonça, B. Nogueira, W. Tiengo, and E. Andrade, "Benchmarking Consistency Levels of Cloud-Distributed NoSQL Databases Using YCSB," *IEEE Access*, 2025.

[36] N. B. Seghier and O. Kazar, "Performance benchmarking and comparison of NoSQL databases: Redis vs MongoDB vs Cassandra using YCSB tool," in *International Conference on Recent Advances in Mathematics and Informatics (ICRAMI)*, IEEE, 2021, pp. 1–6.

[37] C. Camilleri, J. G. Vella, and V. Nezval, "Thespis: Actor-Based Causal Consistency," in *2017 28th International Workshop on Database and Expert Systems Applications (DEXA)*, IEEE, Aug. 2017, pp. 42–46. DOI: 10.1109/DEXA.2017.25.

[38] C. Camilleri, J. G. Vella, and V. Nezval, "ThespisTRX: Causally-Consistent Read Transactions," *International Journal of Information Technology and Web Engineering (IJITWE)*, vol. 15, no. 1, pp. 1–16, 2020.

[39] C. Camilleri, J. G. Vella, and V. Nezval, "ThespisDIIP: Distributed Integrity Invariant Preservation," in *Database and Expert Systems Applications*, M. Elloumi, M. Granitzer, A. Hameurlain, C. Seifert, B. Stein, A. M. Tjoa, and R. Wagner, Eds., Cham: Springer International Publishing, 2018, pp. 21–37, ISBN: 978-3-319-99133-7.