Benchmarking and Comparative Performance Analysis of PostgreSQL and MySQL on AWS RDS

Kristin Henderson and Jaren Shead, *M.S. Data Science, Southern Methodist University*

***Abstract*—This paper compares the performance of two widely used relational database management systems (RDBMS)—PostgreSQL and MySQL—on Amazon Web Services (AWS) Relational Database Service (RDS). The objective of this study is to evaluate the performance of these systems under identical configurations with various load scenarios. We used *HammerDB-4.12* to simulate workloads and measure performance metrics such as transactions per minute (TPM), query execution time, and latency. Our results show significant performance differences between the two systems. MySQL demonstrated higher transaction throughput and lower latency, while PostgreSQL exhibited higher latency under concurrent read/write operations. These results highlight important considerations for database selection in cloud environments.**

***Index Terms*—Benchmarking, Cloud Computing, Database Systems, Latency, MySQL, *HammerDB*, Performance Evaluation, PostgreSQL, Relational Database Management System (RDBMS), Transactions per Minute (TPM)**

I. Introduction

As more organizations move their data storage and operations to the cloud, selecting the right relational database management system (RDBMS) is critical to optimizing performance and cost-efficiency. PostgreSQL and MySQL are two of the most widely adopted open-source RDBMS solutions [1], [2]. Despite their similarities, each system has distinct performance characteristics that make it better suited for specific workloads. Our goal is to provide a comparative performance analysis of PostgreSQL and MySQL within Amazon Web Service's (AWS) Relational Database Service (RDS).

The objective of this study is to measure and compare the performance of these databases under identical configurations and load conditions using benchmarking tools. This analysis could be used to help guide database administrators and decision-makers in selecting the appropriate database for their specific needs, especially in cloud-based environments. The performance metrics evaluated include transactions per minute (TPM), total transactions, and latency.

K. Henderson can be contacted via e-mail: hendersonk@smu.edu. J. Shead can be contacted via email: jshead@smu.edu.

II. Methodology

A. Database Setup

To ensure a fair and cost-effective comparison, PostgreSQL and MySQL were both deployed as Amazon RDS instances using identical configurations within the AWS Free Tier [3]. Although initially intended to be compared on AWS Aurora, we used standard PostgreSQL and MySQL instances to remain within the Free Tier's resources, which also influenced our choice of instance type and storage.

Both databases were configured with the same instance type, memory allocation and storage setup (Table 1). Each instance was configured with a db.t4g.micro instance class, which provides 2 vCPUs and 1 GB of RAM. Storage was set to 20 GiB of General Purpose SSD (gp3) with 3000 provisioned Input/Output Operations per Second (IOPS) and a storage throughput of 125 MiBps. We enabled autoscaling to accommodate potential workload expansions.

TABLE I
AWS RDS Configuration Details

Configuration	PostgreSQL / MySQL RDS
Instance Class	Db.t4g.micro
vCPU	2
RAM	1 GB
Storage Type	General Purpose SSD (gp3)
Storage	20 GiB
Provisioned IOPS	3000 IOPS
Storage Throughput	125 MiBps
Storage Autoscaling	Enabled
Max Storage Threshold	1000 GiB

The benchmarking tests were conducted on a t2.micro EC2 instance running Amazon Linux (Linux/UNIX platform). This instance hosted *HammerDB-4.12* and was equipped with MySQL 8.0.40 (MySQL Community Server - GPL) and PostgreSQL 16.4 client utilities to ensure compatibility with the respective RDS. Resource utilization metrics, including CPU, memory, and disk I/O, were monitored using Amazon CloudWatch to capture detailed performance data. Guidance on benchmark setup and results interpretation was provided by OpenAI's ChatGPT [4].

This setup ensured identical specifications across both databases, facilitating a controlled and reproducible comparison of PostgreSQL and MySQL performance.

B. Benchmarking Tools

To ensure a fair comparison, we selected a benchmarking tool compatible with both databases, enabling a single workload to be run on both platforms. The results reflect each database's performance with identical tasks. We used *HammerDB- 4.12* [5], a cross-platform benchmarking tool, with a TPC-C dataset to simulate an identical mixed transactional workload across PostgreSQL and MySQL. Identical configuration parameters were applied to evaluate the performance of each database (Table 2).

TABLE 2
BENCHMARK CONFIGURATION PARAMETERS

Parameter	MySQL Benchmark	PostgreSQL Benchmark	Description
Benchmark Tool	HammerDB 4.12	HammerDB 4.12	Tool used for running the benchmark
Transaction Type	TPC-C: transactional	TPC-C: transactional	Type of workload simulated
Number of Warehouses (Scaling Factor)	10	10	Dataset size determined by warehouses; each contains tens to hundreds of thousands of rows
Number of Tables	9	9	TPCC schema in HammerDB includes 9 main tables
Active Virtual Users (Number of Clients)	10	10	Number of virtual users issuing requests in parallel
Parallel User Threads	10	10	Concurrency configuration (parallel execution)
Total Iterations (Transactions)	10 million per user	10 million per user	Transaction count limit per virtual user
Duration (minutes)	5	5	Length of time for which the benchmark ran
Ramp (minutes)	2	2	Length of time for ramp-up to benchmark

C. Data Collection and Metrics

The study focused on the following performance metrics:

- Transactions per minute (TPM): A measure of the database's throughput.
- Latency: The average time taken to complete a transaction.
- Total number of transactions: The total volume of activity processed.
- Resource Utilization: CPU, memory, and disk I/O metrics, collected through Amazon CloudWatch.

III. Results

A. PostgreSQL Results

The PostgreSQL RDS instance processed a total of 32,176 transactions, with an average aggregated TPM of 888. The average latency was recorded at 448.52 ms, highlighting PostgreSQL's limitations in efficiently handling concurrent write-heavy requests (Table 3).

B. MySQL Results

In contrast, the MySQL RDS instance processed 60,928 transactions over the same duration, achieving a significantly higher TPM of 4,058. The average transaction latency was much lower at 179.63 ms, indicating better performance under concurrent write-heavy operations (Table 3).

TABLE 3
BENCHMARK STATISTICS COMPARISON

Metric	MySQL Benchmark	PostgreSQL Benchmark	Description
Total Transactions (Activity)	60,928	32,176	Total number of procedure calls processed
New Orders Per Minute (NOPM)	1,759	340	Rate of processing new orders, which add new data to the system
Transactions per Minute (TPM)	4,058	888	Total number of aggregated transactions completed per minute
Elapsed Time (s)	424.32	462.06	Total time for ramp-up and benchmark duration (~7min)
New Orders Latency (ms)	179.63	448.52	Average latency for processing new orders
Weighted Average Latency (ms)	154.91	288.51	Weighted average latency across all transaction types

C. Operating System(OS)-Level Resource Usage

The benchmarking tests revealed differences in how PostgreSQL and MySQL used system resources during high-load conditions. Metrics include network throughput, disk IOPS, freeable memory, and CPU utilization. Each metric was monitored before, during, and after the benchmark to gain insight into each database's performance (Table 4).

- Network Throughput and Disk IOPS: Both databases showed significant changes in network and disk activity during the benchmark. MySQL's network transmit throughput decreased from 2.7 MB/s before the benchmark to 1.23 MB/s during and 695 kB/s after, with a similar decrease in network receive throughput. PostgreSQL showed increased network activity, with transmit throughput rising from 13.8 kB/s before to 26.1 kB/s during and receive throughput rising from 1.49 kB/s before to 8.82 kB/s during. While these trends differ, they reflect the intense data movement associated with transaction processing. PostgreSQL executed more read operations per second, whereas

MySQL excelled in write operations at the cost of efficiency, consuming nearly twice as much CPU.

- Memory Usage: Memory usage spiked for both databases during transaction processing. PostgreSQL required much more memory during and after the benchmark, suggesting higher resource demands for sustained workloads. In contrast, MySQL demonstrated lower and more balanced memory use throughout the benchmark.
- CPU Utilization: CPU utilization increased sharply for both databases under the workload. MySQL exhibited significantly higher CPU usage both before and during the benchmark, indicating a reliance on processing resources to optimize performance.

TABLE 4
OPERATING SYSTEM BENCHMARK STATISTICS (5 MIN AVERAGES)

	MySQL Benchmark			PostgreSQL Benchmark		
Metric	before	during	after	before	during	after
Network Transmit Throughput	2.7 MB/s	1.23 MB/s	695 kB/s	13.8 kB/s	26.1 kB/s	5.47 MB/s
Network Receive Throughput	3.19 MB/s	36.5 kB/s	15.3 kB/s	1.49 kB/s	8.82 kB/s	52.6 kB/s
Read IOPS (ops/s)	3.8	310	104	0.45	450	127
Write IOPS (ops/s)	2.67	507	279	2.58	50.8	60.2
Freeable Memory (MB)	98.2	77.5	87.5	209	70.4	99.9
CPU Utilization (%)	37.70	40.80	12.60	3.97	20.40	12.80

The resource usage metrics suggest that PostgreSQL is better suited for high-read environments with constrained CPU resources, while MySQL's more demanding resource consumption makes it well-suited for heavy, write-intensive workloads requiring peak performance.

IV. DISCUSSION

The benchmarking results revealed some interesting differences between PostgreSQL and MySQL, offering insights into each database's strengths and limitations.

Key Factors Affecting Performance

1. Workload Type and Balance: The TPC-C dataset we used simulates a mixed transactional workload. It mimics an online wholesale supplier that focuses more heavily on New Order write operations.
2. Transaction Complexity: The TPC-C workload includes transactions of varying complexity. For example, New Order transactions create multiple records, while Order Status transactions fetch and aggregate data from multiple tables. These differences in complexity can impact both latency and throughput, as shown in our results. PostgreSQL may have performed better in read transactions because of its sophisticated query planner and optimizer, which allows it to join and aggregate more efficiently [6]. However, MySQL's handling of concurrent transactions through its buffering and caching mechanisms, likely contributed to its superior write performance [7].
3. Disk I/O Requirements: Disk activity increased significantly for both databases during the benchmark, reflecting the I/O-heavy nature of the TPC-C workload. MySQL, with its InnoDB storage engine, appeared to optimize disk usage through in-memory caching and buffering, enabling faster write performance [7, 8]. PostgreSQL exhibited higher sustained write activity possibly due to its Write-Ahead Logging (WAL) mechanism, which prioritizes durability and consistency by logging all changes before writing them to disk [9]. These design choices highlight the difference between MySQL's focus on performance and PostgreSQL's emphasis on ensuring data integrity and reliability.
4. Concurrency and Resource Handling: MySQL used higher CPU during the benchmark to handle concurrent transactions efficiently, but its resource usage dropped significantly afterward. This suggests that MySQL is optimized for peak performance during high-load conditions. MySQL's row-level locking, a feature of its InnoDB engine, facilitates high concurrency by allowing multiple transactions to access different rows simultaneously [7, 8]. PostgreSQL showed lower CPU usage, reflecting its reliance on robust disk-based processes like WAL and background workers to handle concurrency [9]. Both databases managed resources effectively but prioritized different aspects: MySQL focused on speed, while PostgreSQL emphasized reliability.
5. Network Throughput and Resource Optimization: MySQL exhibited unexpected behavior in network throughput. Both transmit and receive throughput decreased during the benchmark. This suggests that MySQL may rely more heavily on in-memory caching and reduced network communication during intensive workloads. In contrast, PostgreSQL maintained more consistent network activity, reflecting its focus on ensuring durability and data integrity through frequent interactions with storage and network resources [9].

Future Work

This study provides a foundational comparison of PostgreSQL and MySQL under a TPC-C workload, but further testing is recommended to gain a more comprehensive understanding of their capabilities under varied conditions.

Future benchmarks should consider the following paths:

- Expanding Workload Diversity: While this study focused on a write-heavy transactional workload, additional benchmarks could explore read-heavy

workloads, mixed workloads balancing reads and writes, and analytical queries (e.g. TPC-H benchmarks). Including variation would allow us to examine each database's performance for different application requirements, including PostgreSQL's potential in handling complex queries and large datasets.

- Varying Benchmark Configurations: Adjusting parameters like the number of virtual users, transaction rates, and dataset sizes would provide a more thorough understanding of database performance under diverse conditions.
- Optimizing Configurations: Tuning database settings specific to MySQL and PostgreSQL could boost their capabilities under specific workloads, ensuring each database performs at its best while still maintaining a fair comparison.
- Scaling AWS Instance Types: Testing on larger or more diverse AWS instance types, such as memory-optimized or compute-optimized instances, could demonstrate how PostgreSQL and MySQL scale.
- Exploring Amazon Aurora Versions: Amazon Aurora, available for both MySQL and PostgreSQL, offers enhanced scalability, fault tolerance, and performance due to its distributed architecture and serverless options [10]. Comparing Aurora's performance with standard PostgreSQL and MySQL under identical workloads could provide valuable insights into its advantages for high-performance and cost-sensitive applications (e.g. Aurora's ability to autoscale for fluctuating workloads vs. standard configurations requiring manual tuning). This additional benchmarking data may benefit applications that are already using AWS services.

By expanding the scope of our comparison, future research can provide more information into the strengths and trade-offs of PostgreSQL and MySQL, offering practical recommendations for their use in diverse cloud-based applications.

V. Connection to Data Science

This project highlights a crucial intersection between data science and database performance optimization. Efficient handling of large datasets is critical to data science, where the ability to retrieve and process data quickly can greatly influence the effectiveness of data analysis and model-building workflows. Selecting the right RDBMS impacts the performance of data retrieval, storage, and computation, especially in cloud environments where resources are shared and often constrained.

Understanding how PostgreSQL and MySQL perform under various workloads provides insights that data scientists and engineers can apply to optimize data pipelines and exploratory data analysis (EDA) [11]. By comparing these RDBMSs, this study adds to our knowledge of how different databases handle varying scenarios, helping inform choices for database configuration in real-world applications.

Key Goals Achieved:

- Benchmarked two RDBMSs under controlled conditions on AWS.
- Measured essential performance metrics, including transactions per minute (TPM), latency, and total transactions.
- Provided insights into the performance trade-offs between PostgreSQL and MySQL, aiding in the selection of the optimal RDBMS for specific workload requirements.

The project made use of several tools, including *HammerDB-4.12*, and *Amazon CloudWatch*, to effectively simulate workloads and monitor resource utilization.

VI. Conclusion

This study offers a comparative analysis of PostgreSQL and MySQL performance on AWS RDS, focusing on transactions per minute (TPM) and latency under concurrent workloads. Our results indicate that MySQL significantly outperformed PostgreSQL in transaction throughput and response time, making it a strong candidate for applications requiring high write performance and low latency, such as e-commerce or online transactional systems. PostgreSQL, while exhibiting lower throughput and higher latency in this test, demonstrated efficient handling of read-intensive workloads and consistent resource usage. This suggests it may be more suitable for applications emphasizing data durability and complex read operations.

These results provide useful guidance for database administrators and data engineers when choosing an RDBMS for cloud environments, emphasizing the importance of aligning database choice with specific workload requirements. For write-heavy or latency-sensitive workloads, MySQL appears to be the more memory-efficient choice, while PostgreSQL may excel in scenarios requiring complex query execution, strict consistency guarantees, or limited compute resources.

Future research could enhance these insights by exploring a broader range of workloads, testing different AWS instance types, and varying benchmark configurations. This could provide a more comprehensive understanding of database performance under varying conditions. This initial benchmark serves as a foundation, encouraging additional studies to deepen our understanding of each database's capabilities, trade-offs, and potential optimizations for cloud-based deployments.

References

[1] PostgreSQL Documentation. [Online]. Available: https://www.postgresql.org/docs/.

[2] MySQL Documentation. [Online]. Available: https://dev.mysql.com/doc/.

[3] AWS Documentation: Amazon RDS. [Online]. Available: https://aws.amazon.com/rds/.

[4] OpenAI's ChatGPT. [Online]. Available: https://openai.com. Assistance in benchmarking setup and interpretation

[5] HammerDB Documentation. [Online]. Available: https://www.hammerdb.com/
[6] PostgreSQL Documentation: Query Planning. [Online]. Available: https://www.postgresql.org/docs/current/runtime-config-query.html.
[7] MySQL InnoDB Architecture. [Online]. Available: https://www.mysqltutorial.org/mysql-administration/mysql-innodb-architecture/.
[8] MySQL InnoDB Storage Engine. [Online]. Available: https://www.mysqltutorial.org/mysql-administration/mysql-innodb-storage-engine/.
[9] Beginner's Guide to Understanding WAL in PostgreSQL. [Online]. Available: https://stormatics.tech/blogs/beginners-guide-wal-in-postgresql.
[10] Amazon Aurora Documentation. [Online]. Available: https://aws.amazon.com/rds/aurora/
[11] Bechant, "Database Benchmarking: Performance, Cost, and Efficiency Considerations." [Online]. Available: https://benchant.com/blog/database-benchmarking.