Database Systems Project

Diogo Matos[1 [70252]] and José Pedreira[1 [61835]]

NOVA School of Science and Technology, Universidade NOVA de Lisboa, Campus da Caparica, 2829516, Caparica, Portugal

Abstract. This project benchmarks the performance of two open-source relational database management systems—PostgreSQL [3] and MariaDB [2]—using HammerDB [5] with the TPROC-C benchmark. By tuning configuration parameters such as buffer sizes, log buffers, and max connections, and by varying workloads (e.g., warehouses and virtual users), we evaluate both throughput (TPM) and efficiency (NOPM). We present an automated benchmarking pipeline and provide statistical comparisons between the two different databases, highlighting their stability, scalability, and optimal configurations.

Keywords: Database Benchmarking · HammerDB · PostgreSQL · MariaDB · TPROC-C · Performance Evaluation · Throughput · Scripting

1 Introduction

This project aims to perform a comparative evaluation of two relational database management systems: **MariaDB** and **PostgreSQL** [2] [3]. Our primary objectives are threefold:

1. To develop an automated benchmarking framework using HammerDB [5].
2. To analyze and compare performance and stability across different configurations and systems.
3. To derive insights into how configuration tuning affects throughput and responsiveness.

We selected MariaDB and PostgreSQL due to their widespread use and strong community support. MariaDB was chosen for its simplicity and compatibility with MySQL—it serves as a reliable baseline system with minimal setup complexity. PostgreSQL, on the other hand, is known for its advanced features, strong support for concurrency and indexing, and overall robustness in more demanding environments.

Originally, we also intended to include Oracle Database [1] in our benchmarks. However, due to our team's limited size (two members) and the substantial overhead of deploying and configuring Oracle—particularly due to its proprietary—we ultimately excluded it from full automation. Nevertheless, we prepared partial setup files and preliminary deployment attempts, which demonstrate our intent and effort to include it in the study.

1.1 Test Infrastructure and Environment

To ensure a thorough and reproducible evaluation, we ran the same benchmark tests on two distinct environments:

- A **macOS-based setup**, which required deploying database systems inside Docker containers, and running HammerDB in a dedicated Linux virtual machine. Due to the lack of native support for HammerDB on macOS, this setup required extensive scripting and environment orchestration to ensure tests were automated, reproducible, and isolated.
- A **Windows-based setup**, which also required some extensive scripting to perform and environment orchestration in order to reproduce the tests done on the macOS-based setup

Both systems executed an identical set of test with the same configuration matrix, allowing us to evaluate not only the performance of MariaDB and PostgreSQL, but also the influence of the host environment on test stability and throughput.

1.2 Benchmark: TPROC-C

The benchmark we used to compare the two databases is **TPROC-C**, an open source, transactional workload implemented in HammerDB that is derived from the TPC-C Benchmark Standard. As a derivative, TPROC-C is simpler and less costly to use, but is not meant to be a replacement for TPC-C.

A transactional, or OLTP (online transaction processing), workload test the ability of a database to process transactions from a number of users while adhering to the ACID properties.

TPC-C is a benchmark created by the TPC, which is the industry with the only published benchmarks that are recognized by all of the leading database vendors.

TPROC-C, which means "Transaction Processing Benchmark derived from the TPC "C" specification", is designed to allow users to easily perform tests without needing software other than the database and HammerDB.

TPROC-C is designed so that the results obtained from repeating the same test on the same hardware and software configuration will not vary much from each other. This allows the user to measure the impact caused by changes made to the configuration.

The TPC-C specification on which TPROC-C is based implements a computer system to fulfill orders from customers to supply products from a company. The company sells 100,000 items and keeps its stock in warehouses. Each warehouse has 10 sales districts and each district serves 3000 customers. The customers call the company whose operators take the order, each order containing a number of items. Orders are usually satisfied from the local warehouse however a small number of items are not in stock at a particular point in time and are supplied by an alternative warehouse. The TPROC-C schema is shown below, in particular note how the number of rows in all of the tables apart from the ITEM table which is fixed is dependent upon the number of warehouses you choose to create your schema. [4]

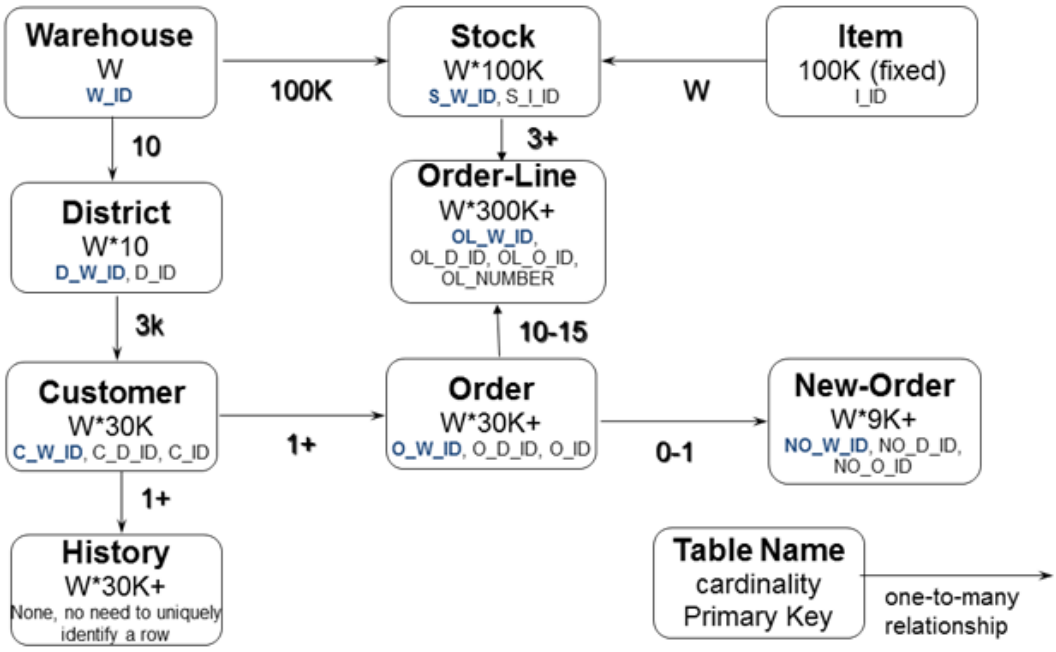


Fig. 1: TPROC-C Schema

HammerDB workloads provide two statistics to compare systems:

– **TPM**: Is the metric derived from the database engine of the number of commits and rollbacks it is processing. Due to this, it is not recommended to use TPM to compare between DBMSs.
– **NOPM (New Orders per minute)**: Measures the rate of New Order transactions, thus measuring the same metric as the tpmC of the TPC-C workload. Since NOPM is a performance metric independent of any particular database implementation, it is the recommended metric to use.

2 Methodology

This section outlines the methodology followed to benchmark the performance of MariaDB and PostgreSQL. Our goal was to automate testing in a way that allowed us to consistently measure throughput (TPM) and transaction efficiency (NOPM), while also observing the stability of the results across repeated runs.

2.1 MacOS-Based Benchmarking Setup (Author: José Pedreira)

System Architecture

– **Database Hosting**: Both MariaDB and PostgreSQL were deployed in Docker containers running on the host macOS system.
– **Benchmark Execution**: A Linux virtual machine (VM), running Ubuntu, was used to execute the HammerDB CLI with custom TCL scripts. This VM had network access to the database containers running on the macOS host.
– **Automation**: Bash scripts executed from within the VM orchestrated each test by:
  1. Starting the appropriate Docker container .
  2. Waiting for database readiness.
  3. Executing the schema deployment script.
  4. Running HammerDB.
  5. Parsing logs to extract TPM and NOPM results.

  This is will be further discussed in the future.

This architecture, while more complex than a single-node setup, ensured that each test was isolated, reproducible, and unaffected by GUI constraints or OS incompatibilities.

Test Parameters We varied the following parameters across multiple configurations:

– Number of virtual users: 5, 10, 20, 50, 100 and then in later testing 150 and 200
– Number of warehouses: 5, 10, 20, 50 and then in later testing 100 and 150
– Database settings:
  • `max_connections`: Between 100 and 500 then later to 1000
  • `innodb_buffer_pool_size` (MariaDB): Between 128MB to 2G then later to 4G
  • `shared_buffers`, `wal_buffers` (PostgreSQL): Between 8MB to 128MB then later to 256MB

Each configuration was labeled and grouped based on workload size (e.g., "Small", "Max CPU", "I/O Test", etc.) to simplify analysis.

Data Collection and Analysis At the end of each benchmark run, logs were parsed to extract the final TPM and NOPM values, along with detailed metrics per timestep. A Jupyter notebook was used to compute average values, standard deviations, and generate bar plots and boxplots to visualize performance and stability trends.

2.2 Windows-Based Benchmarking Setup (Author: Diogo)

For some of the tests, the PC had other applications open, for other tasks, which may have impacted some of the results that were obtained.

System Architecture

- **Database Hosting**: Both MariaDB and PostgreSQL tested on the local Windows 10 operating system.
- **Benchmark Execution**: Custom TCL scripts, based on the scripts written by José, were used to execute the HammerDB CLI. These scripts were used to automate the process executing tests with different values for the variables of the databases MariaDB and PostgreSQL.
- **Automation**: Bash scripts executed using Git Bash with admin privileges orchestrated each test by:
  1. Stopping the database.
  2. Changing its configuration values
  3. Waiting for database to restart.
  4. Executing the schema deployment script.
  5. Running HammerDB.
  6. Parsing logs to extract TPM and NOPM results.

  This is will be further discussed in the future.

This architecture, while more complex than using the GUI, sped up the process of executing tests with different database configurations and allowed the automatic extraction of test data.

Test Parameters We varied the following parameters across multiple configurations (Same tests as José):

- Number of virtual users: 5, 10, 20, 50 and 100
- Number of warehouses: 5, 10, 20 and 50
- Database settings:
  - `max_connections`: Between 100 and 500 then later to 1000
  - `innodb_buffer_pool_size` (MariaDB): Between 128MB to 2G then later to 4G
  - `shared_buffers`, `wal_buffers` (PostgreSQL): Between 8MB to 128MB then later to 256MB

Each configuration was labeled and grouped based on workload size (e.g., "Small", "Max CPU", "I/O Test", etc.) to simplify analysis.

Data Collection and Analysis Like in macOS, at the end of each benchmark run, logs were parsed to extract the final TPM and NOPM values, along with detailed metrics per timestep.

Automated Benchmarking Framework A crucial part of this project involved designing and implementing scripts that automated the benchmarking of database configurations under our test scenarios. Our approach aimed to provide repeatability, flexibility in test parameters, and clean data collection for both MariaDB and PostgreSQL systems.

This automation was beneficial, however the time it took to get the scripts ready, alongside a last minute complication with PostgreSQL, due to the Stress test being too much for the PC, made it so that the number of repetitions for the PostgreSQL tests with "all warehouses" disabled had to be reduced to have some data to analyze before the deadline

Changing the configuration parameters manually would have taken a lot of time, so implementing a script that handled that automatically allowed us to focus on other parts of the project while the tests were running, with minimum supervision necessary.

Script Structure The automation was built using a combination of Bash and TCL scripting. Each database system (MariaDB and PostgreSQL) was provided with dedicated versions of:

- `schema_only.sh`: Responsible for initializing the schema without populating it.
- `benchmark_run.tcl`: Executes the TPROC-C workload under defined test parameters.
- `run_benchmarks.sh`: Orchestrates test iterations, applies configuration settings, and triggers HammerDB workloads.
- `restart_*.sh`: Bash scripts used to restart containers with updated database configurations prior to each test.
- `stop_*.sh`: Bash scripts used to stop database services, waiting for them to completely shutdown before finishing execution.
- `stop_*.sh`: Bash scripts used to start database services.

Configuration Management Each script was designed to accept parameters that varied:

- Number of virtual users (`vu`)
- Number of warehouses (`wh`)
- Max connections
- Buffer pool sizes
- Log buffer sizes (or WAL buffer equivalents)

Due to time limitation we didn't include the parameter for all warehouses to change between true and false, but this was made manually in the script and will be further discussed in the future. We also weren't able to perform as many tests for PostgreSQL with "all warehouses" disabled as we originally planned, and we skipped doing the Stress test, since running that test for PostgreSQL with "all warehouses" enabled was too much for the PC, to the point that it was necessary to reinstall PostgreSQL so that it could be used again.

These combinations allowed us to explore a broad performance space and observe how system resources impact the functionality and result of the HammerDB workload.

Test Execution Flow For each configuration:

1. The database container was restarted with the specified settings via the corresponding `restart_.sh` script.
2. The schema was deployed using the `schema_only*.tcl` script.
3. The system waited until the DB was ready and populated.
4. The `benchmark_run*.tcl` script was executed 5 (2 for PostgreSQL with "all warehouses" disabled) times to gather statistics under repeatable load.
5. Results were parsed and saved into structured log files for analysis.

This flow was fully automated using the `run_benchmarks*.sh` scripts, which ensured consistency across test runs and minimized manual intervention.

Post-processing After each run, log parsing was done using a custom Python script (explained in Section 3) to extract TPM (Transactions Per Minute) and NOPM (New Orders Per Minute) metrics. All 5 "runs" for each test scenario were then written to CSV files and used to generate performance comparison plots.

3 Results

This section presents and analyzes the results obtained from our automated benchmarking framework. All tests were executed on both MariaDB and PostgreSQL under identical workloads and configurations, using HammerDB's TPC-C profile. To ensure a fair and structured

evaluation, we designed a benchmark suite composed of eight distinct test cases. Each test simulates a particular workload pattern, varying parameters such as connection limits, memory allocation, concurrency levels, and transaction volume. These configurations were consistently applied across both MariaDB and PostgreSQL deployments, allowing for direct comparison under identical test conditions.

Table 1 shows the settings used for each test case. These parameters were passed to the respective database systems through configuration files and benchmark scripts.

Table 1: Benchmark Configuration Matrix for MariaDB and PostgreSQL

Test Case	`MC`	`BPS`	`LBS`	`VU`	`WH`	**Test Nº**
Small	100	256MB	8MB	5	5	1
Medium	200	512MB	16MB	10	10	2
Large	500	1GB	32MB	20	20	3
Max CPU	500	2GB	64MB	50	20	4
I/O Test	200	256MB	8MB	20	5	5
Memory	200	2GB	64MB	10	10	6
Latency	100	128MB	8MB	5	20	7
Stress	1000	4GB	128MB	100	50	8

Legend: MC = Max Connections, **BPS** = Buffer Pool Size for MariaDB and Shared Buffers for PostgreSQL, **LBS** = Log Buffer Size for MariaDB and Wal Buffers for PostgreSQL, **VU** = Virtual Users and **WH** = Warehouses

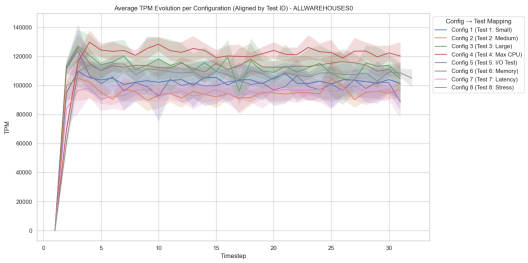
All this 8 tests where tested with the All warehouse setting set to true and false, which is an option that enables increased I/O to the database data area by assigning all of the warehouses in the schema to the Virtual Users in turn [4]

In addition to this set of tests both the MariaDB and PostgreSQL databases where tested with a set of test based on their best performance test and that will be explained in their respective subsections.

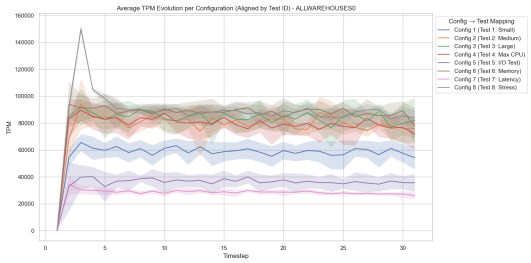
3.1 MariaDB Performance Analysis

This section presents the performance results obtained from MariaDB across the defined benchmark configurations. Each configuration was tested with two settings: `All-Warehouses = False` and `All-Warehouses = True`.

TPM Over Time (MariaDB) Figures2 and Figure3 illustrate the evolution of Transactions per Minute (TPM) over time for each configuration with `AllWarehouses = False` and `True`, respectively. The curves are averaged across the five benchmark runs, and the shaded areas represent the standard deviation.

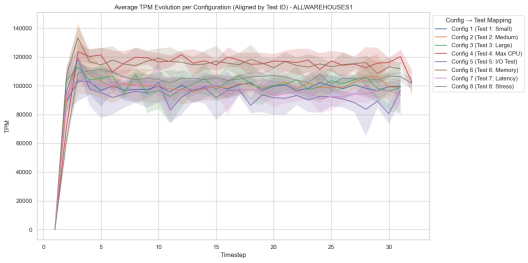


(a) José – All-Warehouses = False

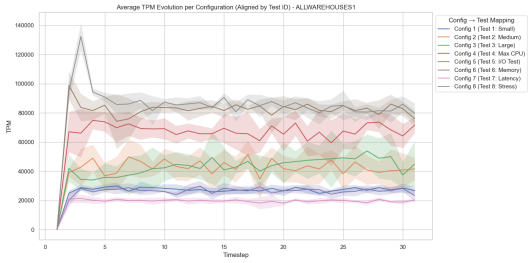


(b) Diogo – All-Warehouses = False

Fig. 2: Average TPM Evolution (All-Warehouses = False) – MariaDB



(a) José – All-Warehouses = True



(b) Diogo – All-Warehouses = True

Fig. 3: Average TPM Evolution (All-Warehouses = True) – MariaDB

Warehouse Setting Comparison (MariaDB) Figures 4 and 5 directly compare the TPM and NOPM for each configuration under both warehouse settings. While the effect of warehouse locality is not dramatic in José's case it does make a huge difference in Diogo's system.

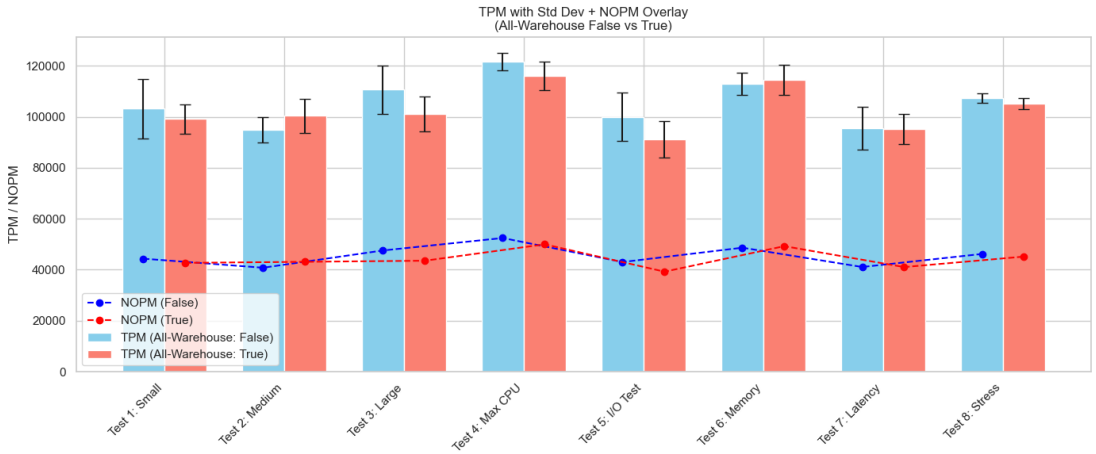


Fig. 4: Average TPM and NOPM per configuration – MariaDB (José)

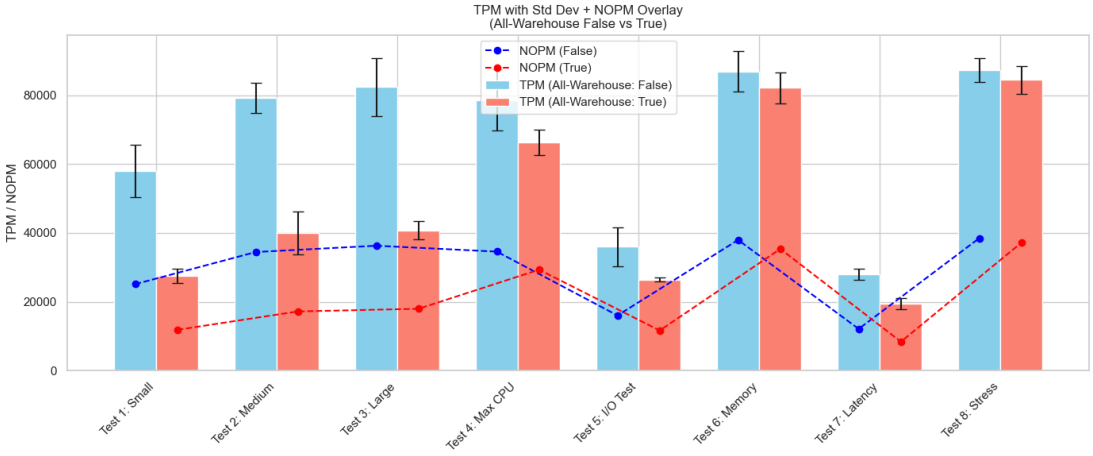


Fig. 5: Average TPM and NOPM per configuration – MariaDB (Diogo)

Speaking broadly we can say that the All warehouses in the case of the MariaDB does result in better performance. Notable the best performance test in José's system was: Max CPU by a large margin considering that even with the All warehouses to false that test came in secound place followed closely by the Memory test as displayed in the following table:

Table 2: Top MariaDB Configurations by Average TPM (José)

Test Name	AllWH	Avg TPM	Std	Min	Max	Avg NOPM	Std	Min	Max
Max CPU	False	121708.20	3361.48	117845	125544	52446.60	1458.15	50713	54138
Max CPU	True	115993.60	5678.21	106196	120932	49971.00	2428.00	45737	51913
Memory	True	114346.80	5893.96	107234	120905	49247.40	2486.11	46270	51993
Memory	False	112844.80	4288.99	106163	118100	48646.00	1773.27	45863	50771
Large	False	110600.40	9358.92	99266	124666	47555.40	4001.25	42694	53554

But in Diogo's system there is a huge difference in performance. The best performance test was: Stress with both the All warehouses false and true in the Top 3 performance followed by the Memory test. The following table can gives us a better insight into the results:

Table 3: Top MariaDB Configurations by Average TPM (Diogo)

Test Name	AllWH	Avg TPM	Std	Min	Max	Avg NOPM	Std	Min	Max
Stress	False	87341.60	3400.24	81846	90306	38486.40	1458.17	36144	39807
Memory	False	86985.20	5917.19	81056	95449	37960.60	2441.66	35439	41396
Stress	True	84502.20	4040.63	77740	88494	37189.20	1732.70	34315	38936
Large	False	82459.60	8454.26	70137	93228	36283.40	3761.79	30762	41057
Memory	True	82152.20	4600.80	77555	87918	35351.00	1955.25	33417	37768

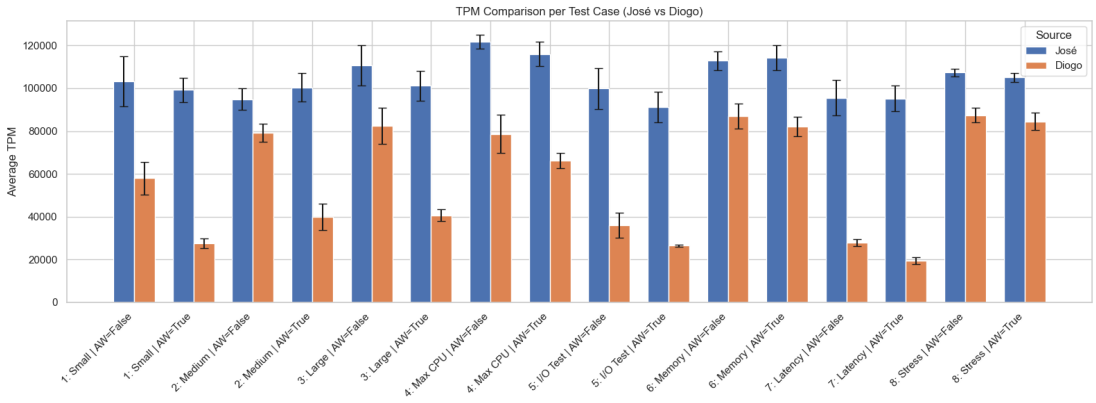


Fig. 6: TPM comparison between José and Diogo

The results show that MariaDB is highly sensitive to the broadly different hardware we presented.

On José's system (Apple M4 Pro, 12 cores (8 performance cores + 4 efficiency cores), 24 GB LPDDR5 3200 MHz, 512 SSD and macOS 15.5 (24F74) with Ubuntu 20.04 LTS running on a VM), the performance penalty from the all warehouse parameter was minimal. In fact, Max CPU remained the best performing configuration. This results suggest that José's system had ample CPU and memory to handle the increased complexity of distributed transactions across warehouses.

In contrast, Diogo's system (Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz, 1498 Mhz, 4 Cores, 8 Logical Processors, 8 GB DDR4 2667MHz, ???? SSD, Windows 10 Home Version 10.0.19045) experienced a significant performance drop when the "all warehouses" option was toggled to true. Tests involving high concurrency or high warehouse usage performed worse. This is likely due to increased memory and CPU usage.

These findings reflect what the documentation describes for All-Warehouses [4], which causes each virtual user to execute transactions across all warehouses, increasing the complexity and locking in the system.

In summary based on our testing the All-warehouses parameter should be carefully evaluated based on the target deployment environment. While it is more realistic, it can strain less powerful systems and obscure finer tuning efforts.

Best MariaDB Configuration Based on this analysis, the `Max CPU` configuration with `All-Warehouse=False` emerged as the best-performing setup for MariaDB run. To investigate whether this performance could be further optimized—or whether it represents a system-level saturation point—we developed six additional test configurations all of them ran on José's system. These exploratory runs aim to determine whether additional tuning could yield higher results or if the database system has already reached its optimal threshold under the current test environment.

Table 4: Benchmark Configuration Matrix for Best MariaDB

Test Case	MC	BPS	LBS	VU	WH	**Test Nº**
Max CPU	500	2GB	64MB	50	20	4
Max CPU1	500	2GB	64MB	50	50	41
Max CPU2	500	2GB	64MB	50	100	42
Max CPU3	500	2GB	64MB	60	20	43
Max CPU4	500	2GB	64MB	100	20	44
Max CPU5	500	4GB	64MB	50	20	45
Max CPU6	500	2GB	128MB	50	20	46

Legend: MC = Max Connections, **BPS** = Buffer Pool Size, **LBS** = Log Buffer Size, **VU** = Virtual Users and **WH** = Warehouses

The figure 7 illustrate the evolution of Transactions per Minute (TPM) over time for each configuration and figure 8 better showcases the performance achieved by the different configurations.

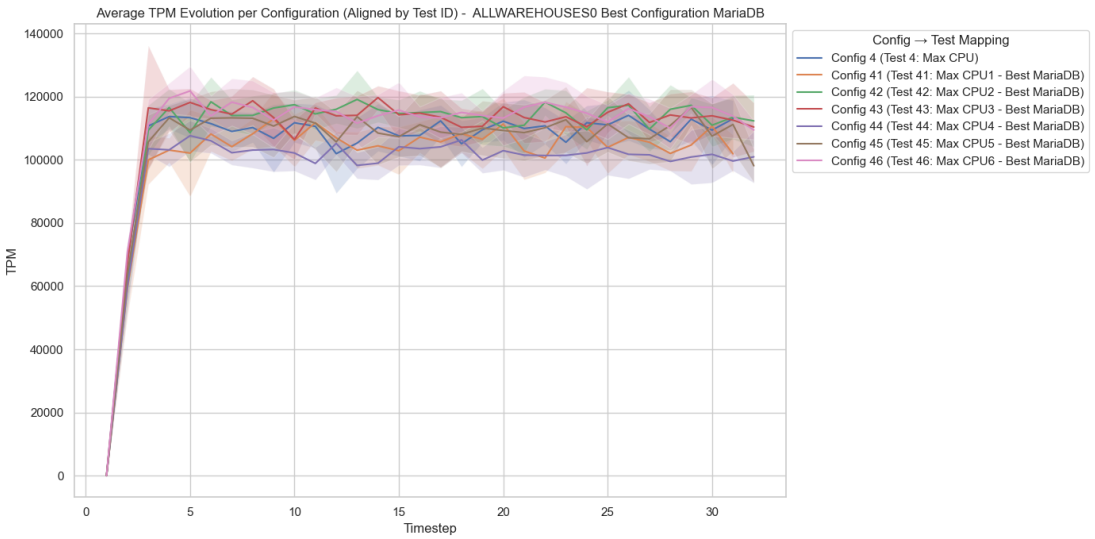


Fig. 7: TPM Evolution per Configuration – Best MariaDB Scenarios (AllWarehouses=False)

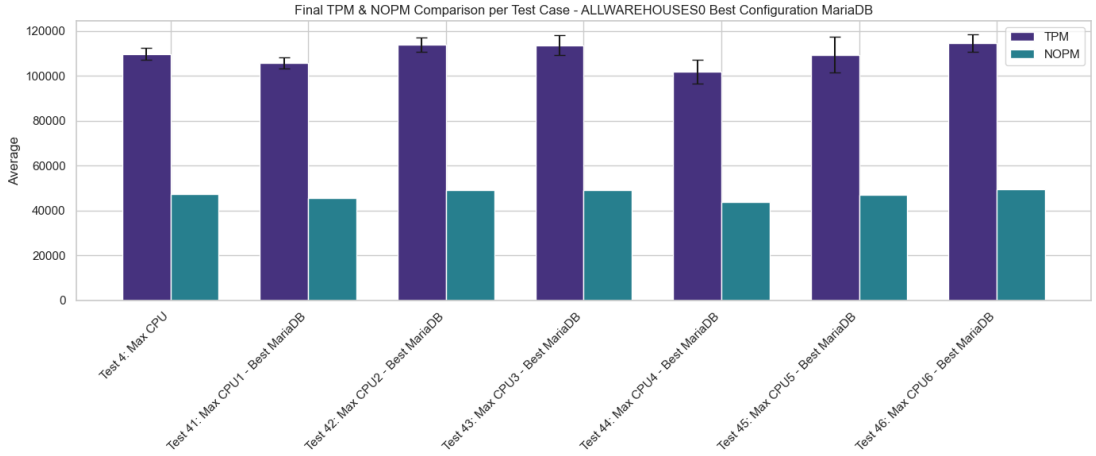


Fig. 8: Average TPM and NOPM per Configuration – Best MariaDB Scenarios (AllWarehouses=False)

Table 5: Top MariaDB Best Configurations by Average TPM

Test Name	Avg TPM	Avg NOPM
Max CPU6 - Best MariaDB	114596.20	49343.60
Max CPU2 - Best MariaDB	113938.20	49060.80
Max CPU3 - Best MariaDB	113655.40	48974.20
Max CPU	109816.80	47253.80
Max CPU5 - Best MariaDB	109439.60	47112.60

Table5 presents the top five configurations with the highest average TPM values from our MariaDB tuning experiments. Several important trends emerge:

- **Max CPU6 – Best MariaDB** achieved the highest performance, outperforming the baseline Max CPU configuration by approximately **4.17% in both TPM and NOPM**. This confirms that increasing the `log_buffer_size` from 64MB to 128MB has a measurable positive impact on throughput.
- **Max CPU2 and Max CPU3** also outperformed the baseline by over **3.61%** and **3.37%**, suggesting that increasing the number of warehouses (from 20 to 100) or virtual users (from 50 to 60) can further unlock performance — although the gains are slightly smaller than those seen with memory tuning.
- The original **Max CPU** setup (Config 4), while strong, ranked 4$^{\text{th}}$, showing it was not yet fully optimized. This highlights the value of parameter exploration, even when performance appears saturated.
- **Max CPU5** (with increased buffer pool to 4GB) showed only a modest improvement over the baseline, suggesting that buffer pool size beyond 2GB yields diminishing returns under our current workload intensity.

Taken together, these results indicate that MariaDB performance under the HammerDB TPC-C workload is sensitive to **log_buffer_size**, and moderately affected by user and warehouse concurrency. However, memory settings (such as buffer pool) may plateau in effectiveness beyond certain thresholds.

Virtual Users vs Warehouses (MariaDB) To further analyze performance scaling dynamics, we designed four exploratory configurations where we varied the number of virtual users (VU) and warehouses (WH) under the "Small" test configuration with the All warehouses set to false. The objective was to observe how MariaDB improves or not in performance based on the virtual users and warehouse change alone.

Table 6: Benchmark Configuration Matrix for Virtual Users and Warehouse change in MariaDB

Test Case	MC	BPS	LBS	VU	WH	**Test Nº**
Small	100	256MB	8MB	5	5	1
Small1	100	256MB	8MB	10	5	11
Small2	100	256MB	8MB	20	5	12
Small3	100	256MB	8MB	5	10	13
Small4	100	256MB	8MB	5	20	14

Legend: MC = Max Connections, **BPS** = Buffer Pool Size, **LBS** = Log Buffer Size, **VU** = Virtual Users and **WH** = Warehouses

Figures 9 and 10 display the results from these "runs". The first shows the evolution of average TPM over time, while the second displays both TPM and NOPM across all test cases.

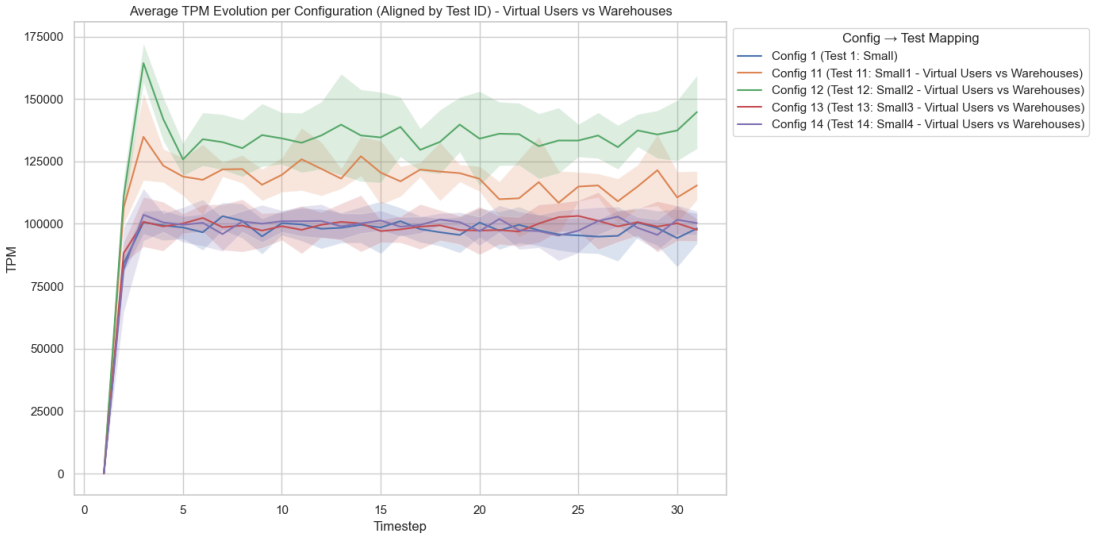


Fig. 9: TPM Evolution per Configuration for Virtual Users and Warehouse change in MariaDB

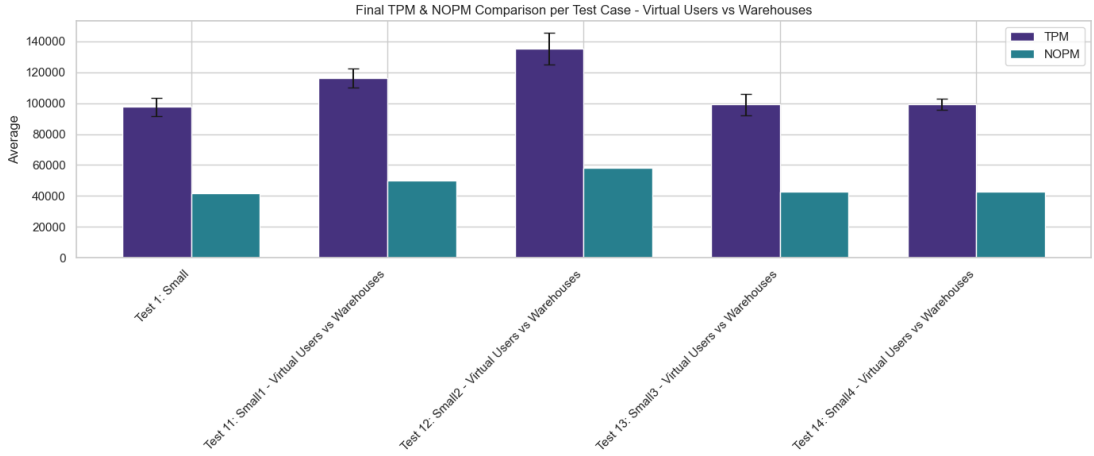


Fig. 10: Average TPM and NOPM per Configuration for Virtual Users and Warehouse change in MariaDB

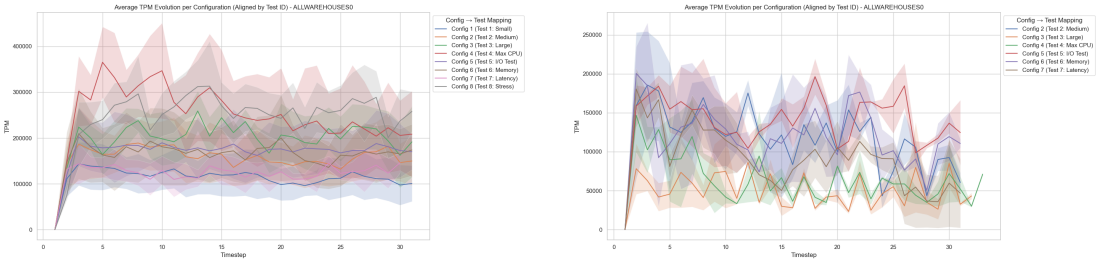
From the previous figures we can determine that:

- **Small2 (20 VUs, 5 WHs)** was the top performer. This suggests MariaDB scales effectively with additional concurrent virtual users under moderate warehouse count. We could extrapolate and say we could use additional improvements with even more users.
- **Small3 and Small4** (which increased warehouses instead of users) provided significantly lower TPM compared to Small2, despite sharing the same memory settings, and not that significant improvement versus the baseline test. This indicates that increasing dataset scale (warehouses) strains the system more than user concurrency.
- **Small1 (10 VUs, 5 WHs)** achieved a performance boost over the baseline Small config as well, showing that even modest increases in VU can unlock higher throughput. Makes sense based on the top performer of the group.

3.2 PostgreSQL Performance Analysis

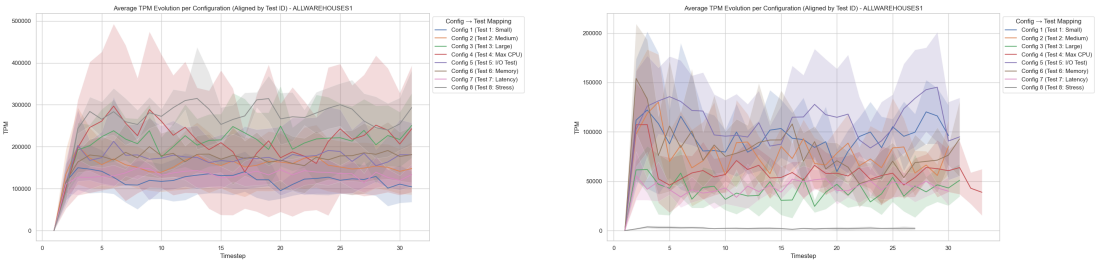
We will now have a similar look at the PostgreSQL database. We will follow a similar approach to what we saw with the MariaDB.

TPM Over Time (PostgreSQL) Figure11 and Figure12 illustrate the evolution of Transactions per Minute (TPM) over time for each configuration with `AllWarehouses = False` and `True`, respectively. The curves are averaged across the five benchmark runs, and the shaded areas represent the standard deviation.



(a) José – All-Warehouses = False (b) Diogo – All-Warehouses = False

Fig. 11: Average TPM Evolution (All-Warehouses = False) – PostgreSQL



(a) José – All-Warehouses = True (b) Diogo – All-Warehouses = True

Fig. 12: Average TPM Evolution (All-Warehouses = True) – PostgreSQL

Warehouse Setting Comparison (PostgreSQL) Figures 13 and 14directly compares the TPM and NOPM for each configuration under both warehouse settings. While the effect of warehouse locality is not dramatic in all scenarios, we observe some variance.

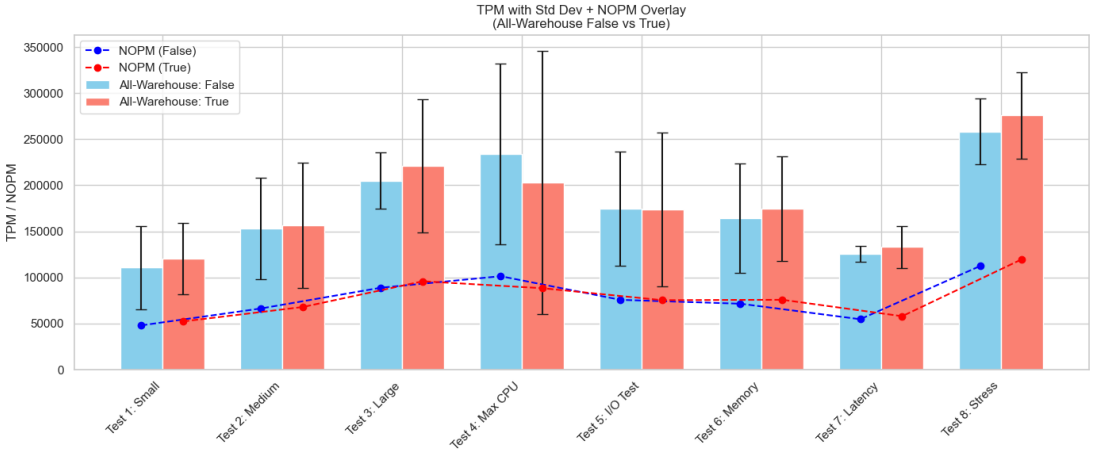


Fig. 13: Average TPM and NOPM per configuration – PostgreSQL

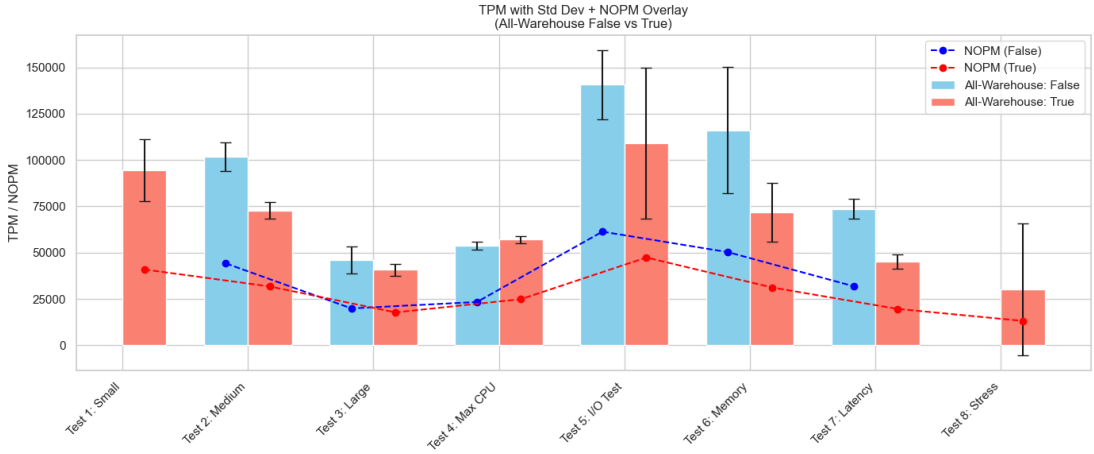


Fig. 14: Average TPM and NOPM per configuration – PostgreSQL

Unlike MariaDB where we could broadly say that we achieve better performance under the All warehouses to false here in PostgreSQL it is essentially the opposite. The following table represents the top 5 performing configurations.

Table 7: Top PostgreSQL Configurations by Average TPM - José

Test Name	AllWH	Avg TPM	Std	Min	Max	Avg NOPM	Std	Min	Max
Stress	True	275695.80	46739.18	204093	327289	119838.80	20159.39	88900	141984
Stress	False	258442.80	35898.69	223866	307045	112386.60	15476.72	97398	133278
Max CPU	False	233892.60	97729.76	145716	346923	101399.20	42428.42	63098	150061
Large	True	221098.60	71836.31	101201	290429	95918.80	31202.01	43852	125906
Large	False	205094.60	30461.28	163263	230268	88943.00	13049.33	71090	99528

Table 8: Top PostgreSQL Configurations by Average TPM - Diogo

Test Name	AllWH	Avg TPM	Std	Min	Max	Avg NOPM	Std	Min	Max
I/O Test	False	140619.50	18597.62	127469.00	153770.00	61317.00	8061.02	55617.00	67017.00
Memory	False	115970.00	34092.45	91863.00	140077.00	50249.00	14577.71	39941.00	60557.00
I/O Test	True	108947.80	40831.67	64468.00	168033.00	47352.00	17832.69	27930.00	73101.00
Medium	False	101685.00	7690.49	96247.00	107123.00	44175.00	3599.17	41630.00	46720.00
Small	True	94308.40	16764.50	76820.00	114949.00	40913.80	7218.65	33422.00	49805.00

Best PostgreSQL Configuration The `Stress` configuration with `AllWarehouse=True` achieved the highest performance for PostgreSQL in José's system. To assess whether this configuration was already near the system's throughput ceiling, we designed six extended variations similar to what we did with MariaDB. These aimed to test the impact of increasing core parameters like virtual users, warehouse count, and memory buffers, and to evaluate if further performance gains could be unlocked or if PostgreSQL had already reached a practical limit within our testing setup.

Table 9: Benchmark Configuration Matrix for Best PostgreSQL

Test Case	MC	BPS	LBS	VU	WH	**Test Nº**
Stress	1000	4GB	128MB	100	50	8
Stress1	1000	4GB	128MB	100	100	81
Stress2	1000	4GB	128MB	100	150	82
Stress3	1000	4GB	128MB	150	50	83
Stress4	1000	4GB	128MB	200	50	84
Stress5	1000	8GB	128MB	100	50	85
Stress6	1000	4GB	256MB	100	50	86

Legend: MC = Max Connections, **BPS** = Shared Buffers, **LBS** = Wal Buffers, **VU** = Virtual Users and **WH** = Warehouses

The figure 15 illustrate the evolution of Transactions per Minute (TPM) over time for each configuration and figure 16 better showcases the performance achived by the different configurations.

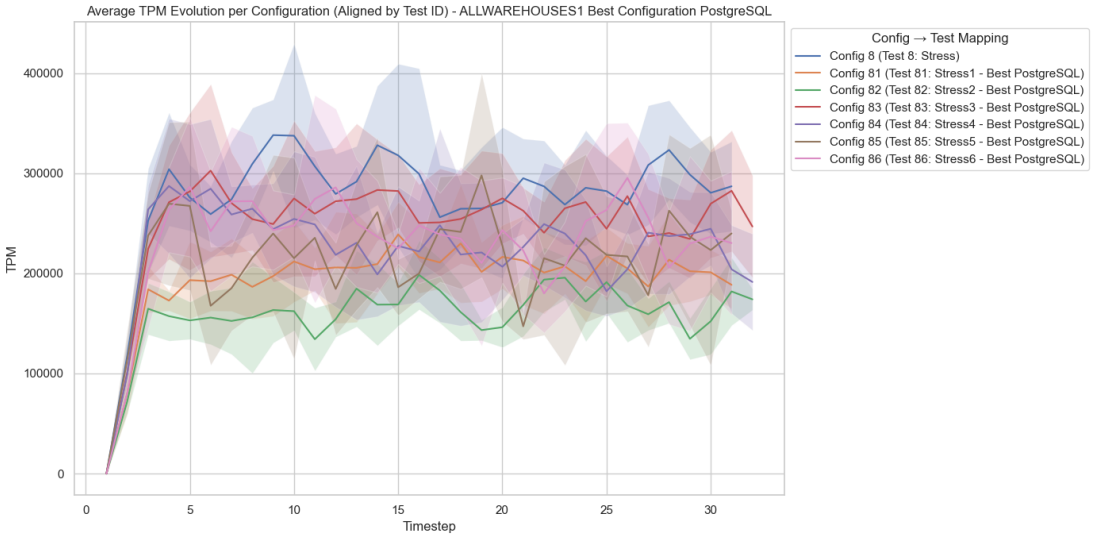


Fig. 15: TPM Evolution per Configuration – Best PostgreSQL Scenarios (AllWarehouse=True)

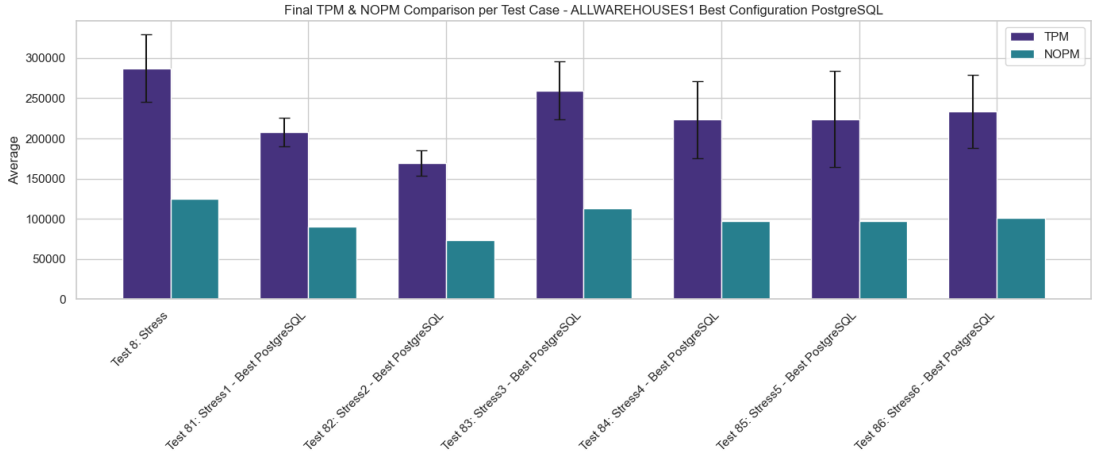


Fig. 16: Average TPM and NOPM per Configuration – Best PostgreSQL Scenarios (AllWarehouse=True)

Table 10: Top PostgreSQL Best Configurations by Average TPM

Test Name	Avg TPM	Avg NOPM
Stress	287466.80	124943.40
Stress3 - Best PostgreSQL	259792.80	112883.40
Stress6 - Best PostgreSQL	233617.80	101624.80
Stress5 - Best PostgreSQL	223997.40	97464.80
Stress4 - Best PostgreSQL	223325.40	97085.20

Table10 presents the top five configurations with the highest average TPM values from our PostgreSQL tuning experiments.

From the results in Table 10, we observe that the original `Stress` configuration remains the best performer, achieving the highest average TPM and NOPM. However, unlike in the MariaDB case, some of the exploratory configurations show noticeable degradation in throughput.

In particular, `Stress3`—which increases the number of virtual users from 100 to 150—performs relatively well, suggesting PostgreSQL can handle more concurrency up to a point. However, attempts to push beyond that, such as in `Stress4` (200 virtual users), result in diminishing returns. Similarly, increasing buffer sizes (`Stress5` and `Stress6`) does not yield further gains and in some cases appears to negatively impact performance.

These findings suggest that while PostgreSQL is capable of high throughput under pressure, its performance is sensitive to over-provisioning. The original `Stress` setup likely represents a balanced point where CPU, memory, and connection limits are optimally aligned for our test environment.

3.3 MariaDB VS. PostgreSQL Performance Analysis

To better understand the difference between MariaDB and PostgreSQL, we conducted a direct side-by-side comparison across all benchmark scenarios. Figure17 illustrates the performance of both Databases side by side and Figure18 represents coefficient of variation (CV) of TPM for each test case—a key metric for assessing the stability of each system under different loads.

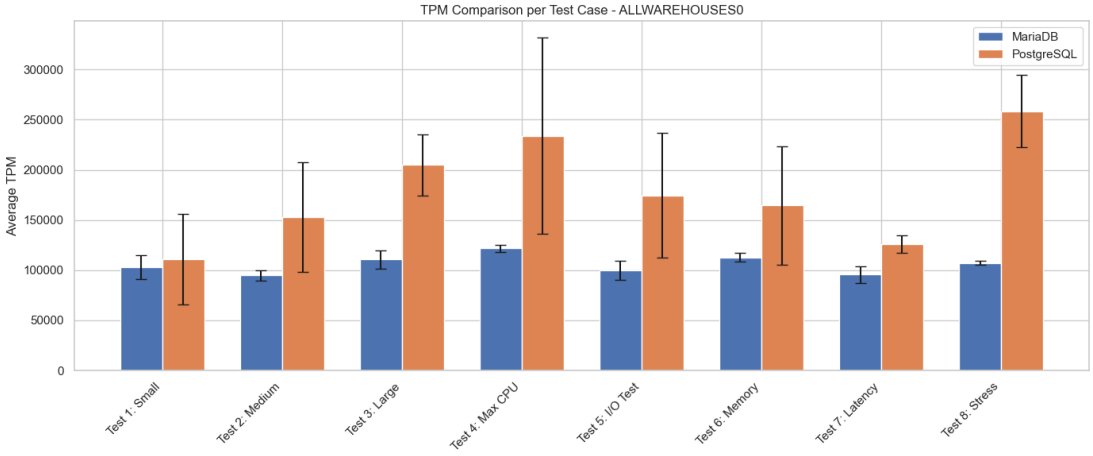


Fig. 17: Performance – MariaDB vs. PostgreSQL

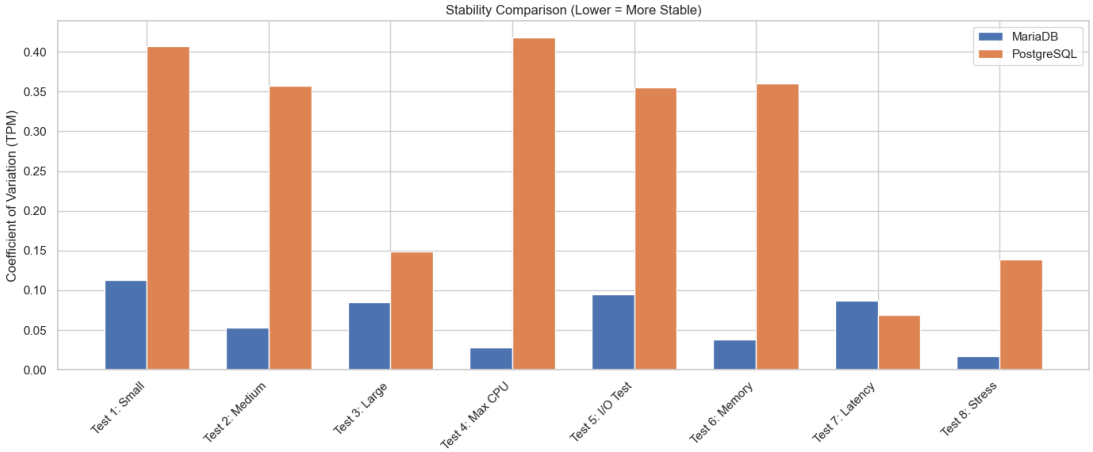


Fig. 18: Stability Comparison (Coefficient of Variation in TPM) – MariaDB vs. PostgreSQL

The results show a clear performance advantage for PostgreSQL. It consistently achieved higher throughput (TPM). However, this performance gain comes with a notable trade-off in stability. The CV values for PostgreSQL are significantly higher in all but one test, indicating greater variability and lower consistency between runs.

In contrast, MariaDB exhibited lower performance ceilings but demonstrated greater stability across configurations. Its tighter variability suggests more predictable behavior—an advantage in some environments.

In summary, our benchmarks suggest that:

- PostgreSQL is capable of achieving higher peak performance but with higher volatility.
- MariaDB offers more consistent and predictable performance but with lower overall throughput.

These findings highlight the classic trade-off between throughput and stability, reinforcing the importance of selecting a database system based on specific workload requirements and operational constraints.

4 Conclusions

From the results of our comprehensive MariaDB testing we can determine that system performance is highly influenced by both hardware and workload scenario. On José's M4 Pro-based system, MariaDB demonstrated strong stability even under increased transactional complexity (e.g., when the All-Warehouses flag was enabled). In contrast, on Diogo's machine, the same setting introduced significant performance penalties, particularly under high-concurrency or high-warehouse conditions.

Across the all benchmark, the Max CPU configuration stood out as the best for MariaDB. Further exploratory tuning confirmed this could be improved by modest adjustments, particularly increasing the log_buffer_size, which delivered up to 4.17% gains in performance.

Finally, the targeted experiments comparing the effect of virtual users vs. warehouses on a small workload confirmed that MariaDB benefits significantly from increased user concurrency, whereas scaling the warehouse had insignificant—and sometimes negative—impact on performance. These results reinforce the importance of aligning database configuration with expected usage patterns.

The PostgreSQL test show a performance profile that contrasts notably with that of MariaDB. Most significantly, PostgreSQL consistently achieved better throughput when the All-Warehouses setting was enabled. This setting—designed to increase I/O and inter-warehouse activity—appears to better utilize PostgreSQL's internal architecture.

On José's system, the top-performing configuration by a clear margin was the Stress test with All-Warehouses=True, which is substantially higher than any result observed in the MariaDB tests. Other high-ranking PostgreSQL configurations also followed the same pattern: more demanding workloads with All-Warehouses=True yielded superior performance. This highlights PostgreSQL's capacity to scale with complexity, provided the system has sufficient hardware.

On the oposite side, on Diogo's system PostgreSQL did not benefit as dramatically from All-Warehouses. In fact, the highest performing configuration was the I/O Test with All-Warehouses=False, suggesting that Diogo's setup hit resource limits earlier when faced with PostgreSQL tests. Performance degraded under higher concurrency and warehouse distribution, likely due to I/O contention and limited caching capacity.

In summary, PostgreSQL demonstrates excellent performance scalability, particularly under complex transactional scenarios, but this comes with greater hardware demands and potential instability.

All of the test data and aditional analisys of the data can be acessed at our GitHub repo: `https://github.com/joko1712/SDB_Project_2025`

5 Individual Contributions

In this section, we outline the contributions of each group member to the project.

5.1 Group Working Methodology

The work was divided between the group members in the following way:

- Diogo was responsible for (Adapting the scripts to work on Windows 10).
- José was responsible for (Creating the tests used in this analysis, and creating the original scripts for them. Also responsible for creating the figures to analyse the data that was received).

References

1. Corporation, O.: Oracle database installation guide (2023), `https://docs.oracle.com/en/database/`

2. Foundation, M.: Mariadb documentation (2024), https://mariadb.com/kb/en/
3. Group, P.G.D.: Postgresql documentation (2024), https://www.postgresql.org/docs/
4. HammerDB Project: HammerDB User Guide: Chapter 4 - Running a Benchmark (2023), https://hammerdb.com/docs4.0/ch04s06.html, accessed: 2025-06-08
5. Shaw, S.: Hammerdb documentation (2023), https://www.hammerdb.com/docs/