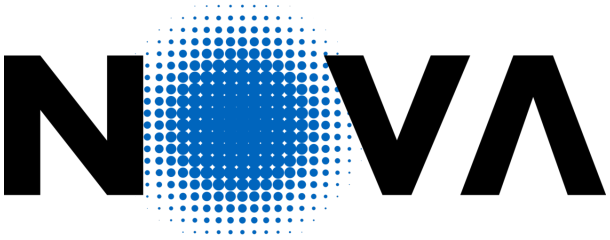


NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

NOVA UNIVERSITY OF LISBON

MSC IN COMPUTER SCIENCE

PERFORMANCE COMPARISON BETWEEN DBMSS UNDER TPROC-C WORKLOADS

José Costa (62637)
Rodrigo Albuquerque (70294)
Rodrigo Silva (70567)

DATABASES SYSTEMS

JUNE 9, 2025

Contents

1 Introduction

This project aims to benchmark and compare the performance of different database systems using the TPC-C workload, a standard for evaluating OLTP (Online Transaction Processing) environments.

Automated scripts are used to run identical tests across multiple databases, each configured with similar settings to ensure a fair comparison.

Key metrics such as Transactions Per Minute (TPM) are measured under varying levels of concurrent users.

The results help identify the strengths and limitations of each database in handling transactional workloads.

In this project, the databases used were PostgreSQL, MySQL and MariaDB.

In total, we ran 54 tests:

- 4 tests scaling the number of virtual users (2 4 8 12) and warehouses (VU*5) on all PCs and databases (48 tests in total);
- 1 test with the number of virtual users set to the same as the number of threads in that PC, warehouses set to VU*5, *allwarehouse* = true on all databases on just one PC (3 test in total);
- 1 test with the number of virtual users set to the same as the number of threads in that PC, warehouses set to VU*5, with default config on all databases on just one PC (3 test in total).

2 Overview of HammerDB

HammerDB is a free, open-source tool for benchmarking the performance of relational databases [1].

It supports popular databases like Oracle, SQL Server, PostgreSQL, MySQL, and more. HammerDB uses industry-standard workloads such as TPROC-C and TPROC-H to simulate real-world database activity.

It offers both a graphical interface and command-line options, making it suitable for developers, DBAs, and system administrators to test, compare, and tune database performance.

In some cases we used HammerDB in docker containers to run the tests, which allows for easy setup and isolation of the testing environment.

In another case, we used the Windows version of HammerDB to run the tests on a Windows machine.

2.1 Overview of TPROC-C

TPROC-C is a benchmark designed to evaluate the performance of database management systems (DBMS) using a transactional workload. It simulates a typical online transaction processing (OLTP) environment, focusing on operations like inserts, updates, and deletes across multiple tables.

2.2 TPROC-C vs TPROC-H

TPROC-H is a benchmark designed for data warehousing and analytical workloads, while TPROC-C is focused on transactional processing. TPROC-H emphasizes complex queries and large data sets, whereas TPROC-C simulates real-time transactions with a focus on insert, update, and delete operations.

3 Problem & DBMS Summary

Modern applications progressively depend on robust, scalable database systems to effectively manage workloads in a transactional manner. Choosing the right Database Management System (DBMS) is critical to achieve the best possible performance, especially where concurrency is high and the workload is varied. With so many DBMSs to select from, with their own strengths, configurations, and community support, getting the right one can be problematic.

The main objective of this study is to provide a clear comparison of how three of the most popular open-source DBMSs like,PostgreSQL, MySQL, and MariaDB perform under TPROC-C workloads. This is a close approximation of OLTP environments and thus is suitable to use in evaluating systems for high-throughput transaction processing.c

A brief overview of the DBMSs under test is provided below:

- **PostgreSQL**[2]: A feature-rich, relational database system known for its standards compliance and emphasis on data integrity. It distinguishes itself through support for advanced SQL features such as window functions, full outer joins and recursive queries. It also supports full ACID transactions, MVCC (Multiversion Concurrency Control) for high concurrency, and powerful indexing options, making it ideal for complex, analytical, and high-transaction applications.
- **MySQL**[3]: Recognized for its simplicity, reliability, and speed, MySQL is a relational database system optimized for read-heavy workloads and fast query execution. Its architecture is modular, featuring pluggable storage engines such as InnoDB (the default, which offers ACID compliance and support for transactions). While traditionally less feature-rich than PostgreSQL, recent versions have added support for several features that improved its suitability for modern applications.
- **MariaDB**[4]: Due to it being a fork of MySQL, created to ensure continued open-source development, we are expecting performance to be similar or better than MySQL. MariaDB offers unique enhancements such as more advanced storage engines (e.g., Aria) and better performance optimizations. Its optimizer and query planner are designed to handle complex queries more efficiently, and it places a strong emphasis on performance tuning. MariaDB's ecosystem is tailored for users who want MySQL's simplicity with additional flexibility and modern database capabilities.

Each DBMS was installed with identical hardware and software configurations to provide a level playing field for the tests. Experiments were designed to highlight differences in how each system handles transaction loads, concurrency, and configuration parameters. Through comparison of performance in a systematic manner across controlled tests, this study aims to guide database selection on the grounds of empirical evidence and not assumptions.

4 Benchmark Description

The benchmark used in the research is the TPROC-C workload using HammerDB. TPROC-C is designed to simulate an OLTP environment in the average case and is composed of transactions containing new orders, orders to make payment, checking orders status, delivering orders, and updating stock status. The aforementioned operations can be likened to real-world application environments.

4.1 Benchmark Goals

The main objectives of the benchmark are to:

- Compare the throughputs of the DBMSs at different levels of concurrency in terms of Transactions Per Minute, or TPM.
- Monitor the scalability of the systems as virtual users and additional warehouses get created.
- Measure consistency across repeated experiments as well as with different setups.

4.2 Test Parameters

To ensure consistency and fairness, the same configuration template was used for all tests, with the sole variations being the number of virtual users, warehouses, and the specific DBMS being tested. The significant parameters were:

- **Virtual Users (VU):** Simulated clients making postings simultaneously. The values 2, 4, 8, and 12 were chosen to perform the scaling test, maxing at the number of threads available on the test machines.
- **Warehouses**: A TPROC-C scale unit. We choose to use five times the number of virtual users, since it's the number used by the default tests provided by HammerDB.

- **Ramp-Up and Cleanup Time**: Ramp-up time was set to 2 minutes and Cleanup time after finishing every test for 1 minute. This ensures more accurate results and load stabilization.
- **Test Duration**: Each test was executed for 10 minutes (2 Ramp-up + 8 Run).

4.3 Execution Environment

All the testing was automated with custom scripts offering the same setup routines and execution across the systems. The environment included combinations of Windows and Linux systems, based on the setup. HammerDB was run in some cases in Docker containers to offer isolation to the test environment and ensure repeatability.

4.4 Metrics Collected

A critical measurement during benchmarking was the Transactions Per Minute (TPM), as reported by HammerDB. This metric reflects the system's capability to process new orders and associated transactions within a single minute. Additionally, we collected the New Orders Per Minute (NOPM), which provides a more focused measure of transactional throughput.

Secondary metrics included CPU and memory usage as reported by the operating system. We were unable to use HammerDB to collect performance metrics, since for PostgreSQL, the required pgsentinel plugin involves a complex installation process and for MySQL and MariaDB, we were unable to complete the setup on Machine 1 — according to one of the contributors on GitHub, this may be due to issues related to the Windows firewall.

4.5 Limitations

Even though control over variables, consistency across runs, and bias to a variable as limited as possible was the focus of this testing, there are several limitations:

- Variability in network latency, and relative performance of hardware and storage may be negligible in each of the testing environments.
- Default DBMS tuning parameters were used unless stated otherwise, which may not represent the best performance possible for each system.
- The focus was on relative performance under specific workloads, not exhaustive optimization.

5 Methodology

5.1 Hardware and Software Setup

PC	**1**	**2**	**3**	**4**
OS	Windows 11	Windows 11	Linux (Unraid)	MacOS Sequoia
CPU	AMD Ryzen 5 3600	Intel i7-13700H	Intel i3-10100F	Apple M1
Cores	6	14 (6P 8E)	4	8
Threads	12	20	8	8
RAM	16GB	16GB	32GB	16GB
Disk	SSD M.2 NVMe	SSD M.2 NVMe	SSD M.2 NVMe	SSD M.2 NVMe
Read	2500 MB/s	3500 MB/s	3500 MB/s	3400 MB/s
Write	2100 MB/s	2700 MB/s	3300 MB/s	2800 MB/s
Test type	Bare metal	Docker	Docker	Docker

Table 1: Hardware used in the benchmarks

The benchmarking tests were conducted on four various personal computers, each of which satisfied the same specification for the purpose of ongoing performance measurement.

- **PC 1:** All the components, such as HammerDB and the database systems, were installed natively. on the native operating system (bare metal). The reason for this installation was to test performance without the overhead of running in a container.
- **PCs 2 and 3:** HammerDB and each DBMS were run within Docker containers.
- **PC 4:** On this machine, HammerDB is also run within a Docker container. However, since the image is built for x86_64 systems, it must emulate those instructions on the ARM64 architecture. This setup allows us to test whether the emulation layer introduces any significant performance overhead.

We used Docker Compose as our orchestration tool for the Docker containers, allowing the deployment and usage of the same setup as configured, in multiple machines with minimal processing. Each container has a static IP inside a docker network.

For MySQL and MariaDB, we developed custom Dockerfiles that allow external configuration files. This is due to the fact that those databases defaulted to read-only access to the configuration file, unlike PostgreSQL, within the container.

5.2 Database Setup

To prevent inconsistency and promote fairness in benchmarking, MySQL, MariaDB, and PostgreSQL were all installed with their respective configuration files to define the relevant operating configurations needed in the benchmarks. Configurations were made as consistent as possible on the systems, taking into consideration the individual limitations of each DBMS.

The configs used in MariaDB and MySQL are exactly the same when it comes to performance related configurations. For PostgreSQL, the one used was created to be very similar to the others.

Each database instance was restarted with a new schema for each test iteration to eliminate residual data artifacts and ensure consistency. Loading a schema and provisioning a user were also scripted to minimize the level of manual intervention and preserve reproducibility between testing cycles.

5.3 Experimental Setup

In order to provide repeatability and consistency for all the benchmark tests, we employed automated TCL scripts for each DBMS, MariaDB, MySQL, and PostgreSQL, run through HammerDB's command-line interface. Schema creation, user provisioning, workload generation, and results collection were all scripted to reduce human error.

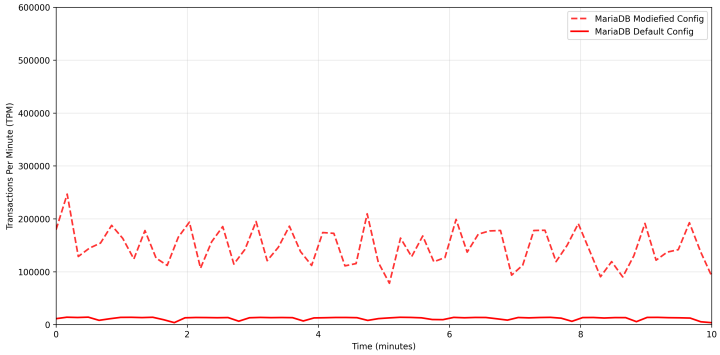
MariaDB TCL Script	MySQL TCL Script	PostgreSQL TCL Script

Every script was adjusted to fit the respective connection parameters, database drivers, and environment variables of the target DBMS but was kept identical in logic sequence to provide equality during test runs.

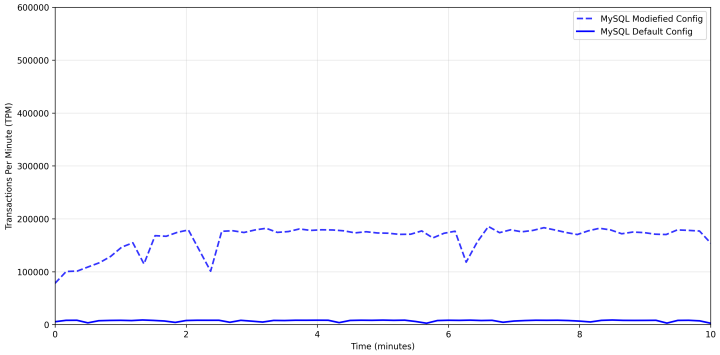
Scripts were invoked using HammerDB's CLI to run schema builds and benchmark runs automatically.

6 Results

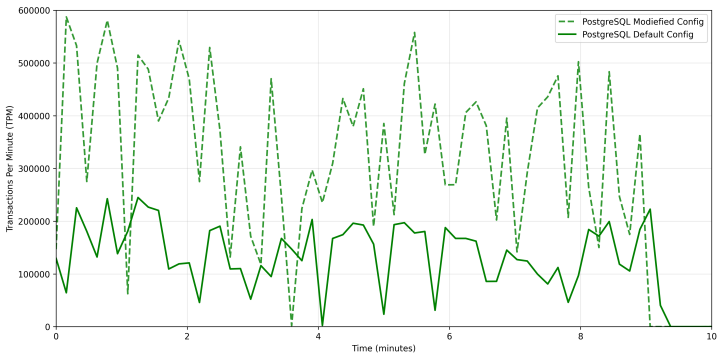
6.1 Database Configuration Comparison



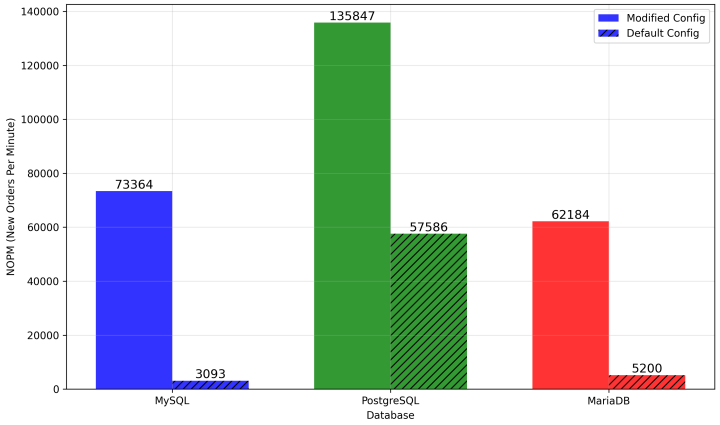
(a) Impact of Configuration on MariaDB (TPM)



(b) Impact of Configuration on MySQL (TPM)



(c) Impact of Configuration on PostgreSQL (TPM)



(d) Impact of Configuration across all Databases (NOPM)

Figure 1: Performance analysis of custom configurations for VU=12 and WH=60 on PC 1

Figure 1 compares the effect of tailored configuration parameters on the performance of three DBMS - MariaDB, MySQL and PostgreSQL - based on a controlled load setting: 12 virtual users (VUs) and 60 warehouses, executed on **PC1**.

- **MariaDB (Figure 1a):** On average, it achieved a TPM comparable to MySQL, though with slightly less stability, which resulted on a lesser total number of transactions. The results with the default configuration were unexpectedly low, which we suspect is largely due to the small InnoDB buffer pool size in the default settings (128 MB). Nevertheless, the performance improvement brought by the configuration is remarkable, increasing the results by several factors.
- **MySQL (Figure 1b):** Recorded a high total TPM, indicating greater usage of the modified installation. Although its default configuration performed worse than that of MariaDB, the InnoDB engine in MySQL appears to benefit more from tuning. However, the final results are quite similar, with differences falling within the expected standard deviation.

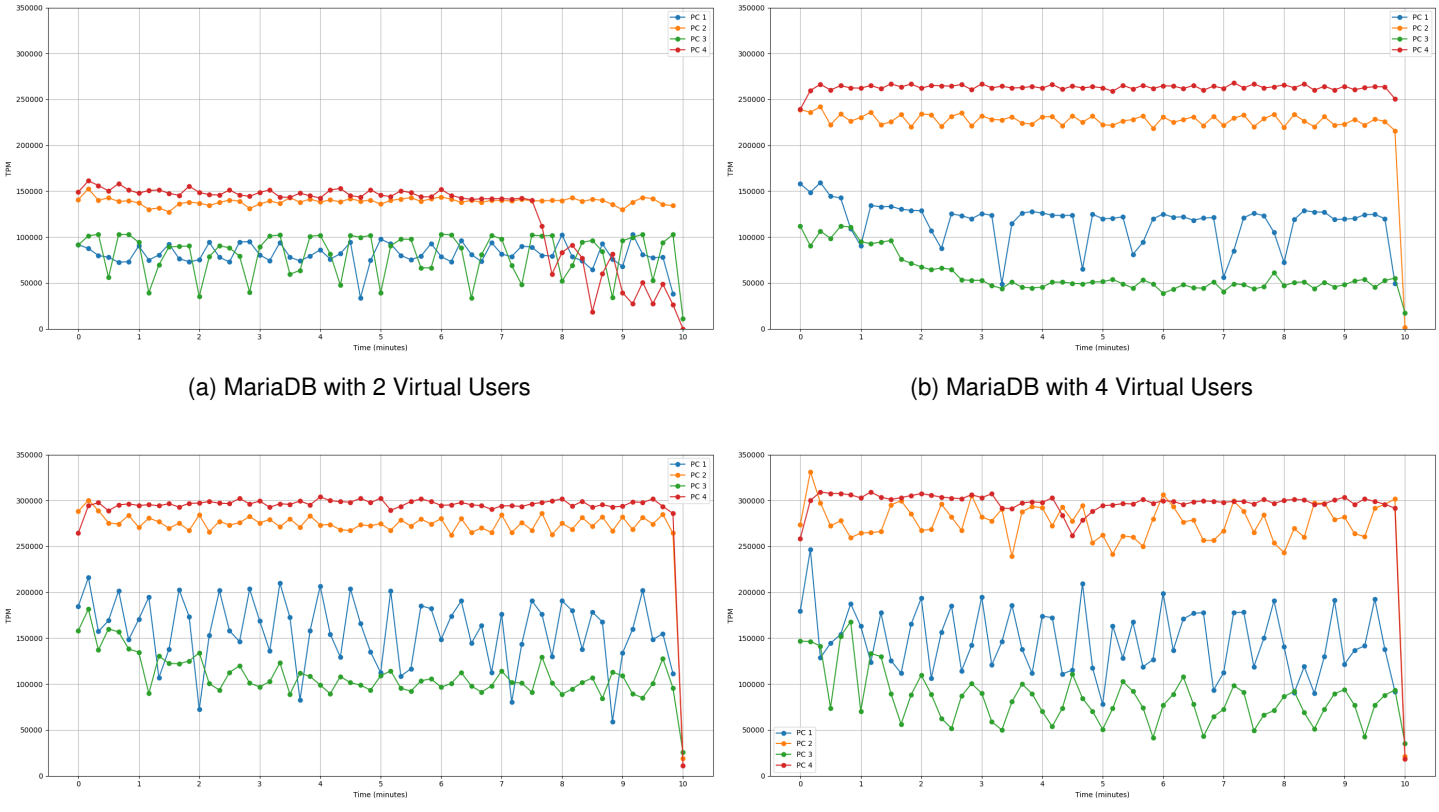
- **PostgreSQL (Figure 1c):** The biggest surprise of these tests was PostgreSQL's performance, which, on average, doubled the TPM results of both MySQL and MariaDB. Despite high fluctuations in throughput, PostgreSQL clearly outperformed the others. Even when comparing the NOPM results[1d] with its default configuration, it managed to only achieve results 10–20% lower than the tuned versions of MariaDB and MySQL, respectively. This outcome was highly unexpected. After multiple reruns and thorough verification of the test environments and configurations, we attribute this advantage primarily to PostgreSQL's Multiversion Concurrency Control (MVCC) architecture, which handles concurrent transactions more efficiently, and its superior query planner, which often generates more optimized execution paths.

Conclusion: The benchmark executed on **PC 1** shows that MySQL and MariaDB benefit significantly from configuration tuning, even though they are still outperformed by PostgreSQL in both TPM and NOPM. As it stands, PostgreSQL appears to be the clear leader in performance by a substantial margin. To ensure that this result is not due to a software or hardware issue specific to PC 1, we will next evaluate how each database performs with its modified configuration across all machine setups.

6.2 Performance along the different Machines

This chapter provides TPM (Transations Per Minute) performance for MariaDB, MySQL and PostgreSQL under four concurrency levels: 2, 4, 8 and 12 Virtual Users (VUs). The goal is to see how well each DBMS scales to higher concurrency and workload intensity for the four computers.

6.2.1 MariaDB



(a) MariaDB with 2 Virtual Users

(b) MariaDB with 4 Virtual Users

(c) MariaDB with 8 Virtual Users

(d) MariaDB with 12 Virtual Users

Figure 2: Performance of MariaDB with different numbers of virtual users

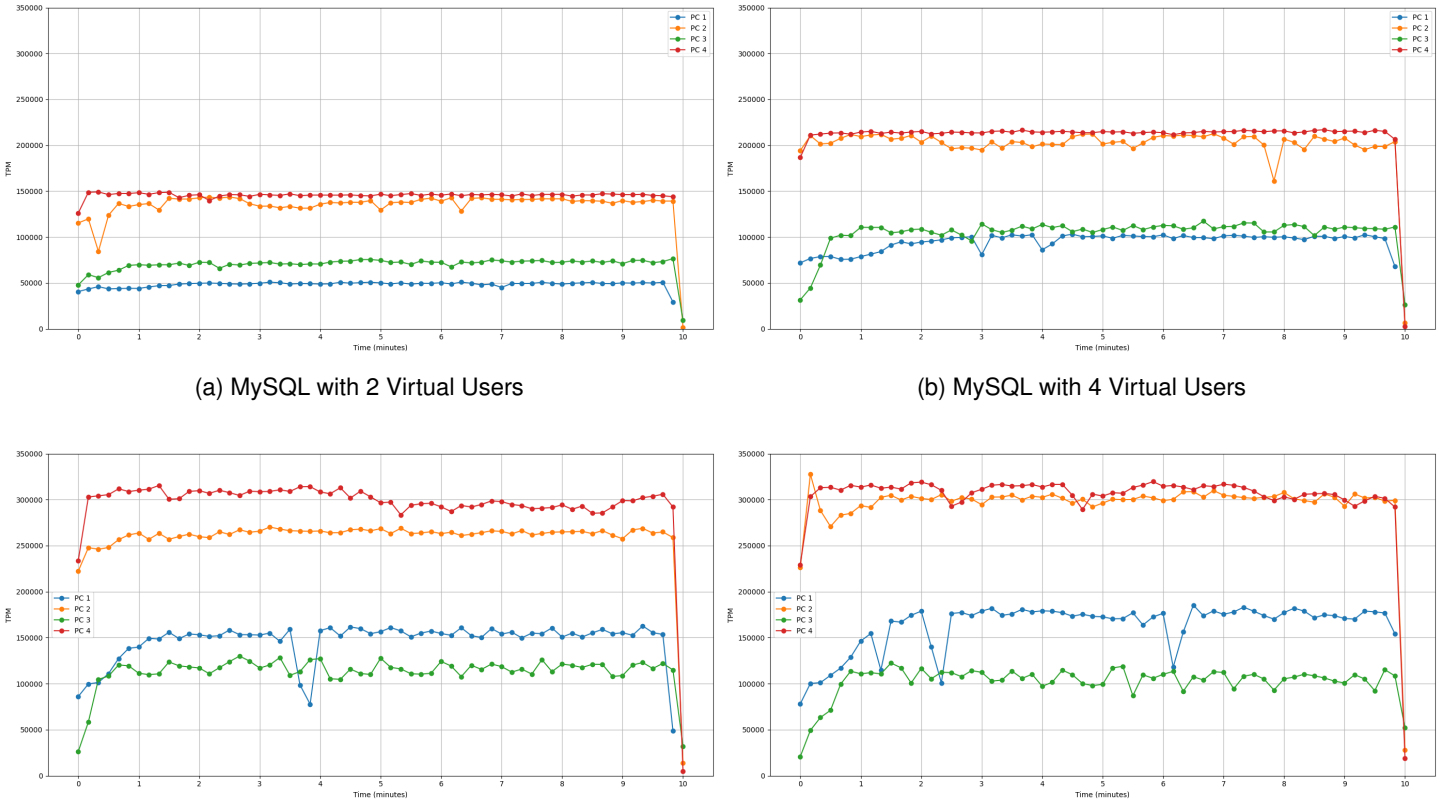
Figure 2 shows that MariaDB exhibits stable behavior with the increase in virtual users:

- **2 to 4 VUs:** A modest TPM growth, showing that MariaDB is actually benefiting from medium concurrency.

- **8 VUs:** System begins to experience stress and TPM growth stabilizes or decreases to some extent.
- **12 VUs:** Performance is consistent but is demonstrating that it is reaching its peak. The results are very identical to the ones in 8 virtual users which indicates a possible bottleneck.

Overall, MariaDB scales pretty well but with diminishing returns after 8 VUs. It's possible that it needs additional tuning or more sophisticated storage engines to scale adequately at even higher loads.

6.2.2 MySQL



(a) MySQL with 2 Virtual Users

(b) MySQL with 4 Virtual Users

(c) MySQL with 8 Virtual Users

(d) MySQL with 12 Virtual Users

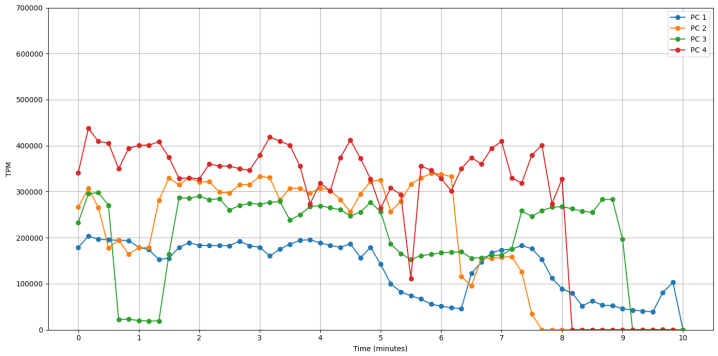
Figure 3: Performance of MySQL with different numbers of virtual users

Figure 3 shows a more responsive performance curve than MariaDB in some machines:

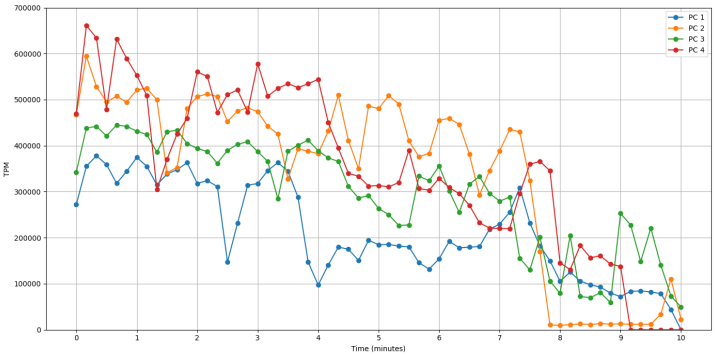
- **2 to 8 VUs:** Steady linear growth in TPM. The InnoDB engine handles increasing concurrency well.
- **12 VUs:** There are better results and stability in all machines compared to MariaDB.

MySQL shows good scalability and concurrency support. It is highly suitable for high-throughput OLTP workloads up to a moderate level of concurrency.

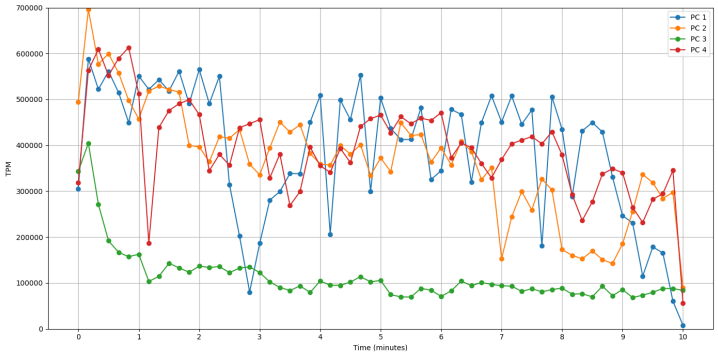
6.2.3 PostgreSQL



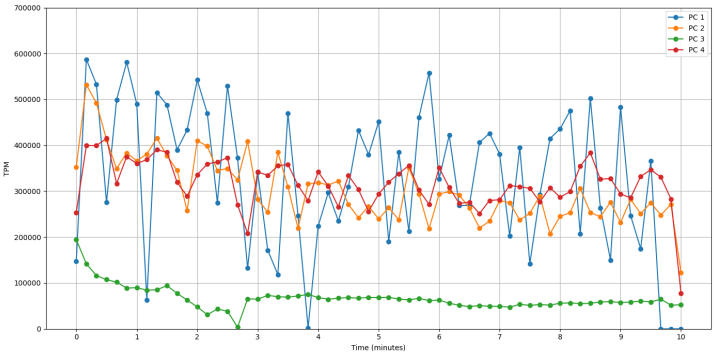
(a) PostgreSQL with 2 Virtual Users



(b) PostgreSQL with 4 Virtual Users



(c) PostgreSQL with 8 Virtual Users



(d) PostgreSQL with 12 Virtual Users

Figure 4: Performance of PostgreSQL with different numbers of virtual users

Figure 4 confirms PostgreSQL's reputation for dealing with concurrency very well:

- **2 to 8 VUs:** TPM rises steadily and markedly, with great use of existing threads and MVCC.
- **12 VUs:** PostgreSQL still scales, holding or even increasing throughput to some extent in some machines, with little sign of performance degradation.

In general, PostgreSQL is the most scalable DBMS here. Its architecture (MVCC, indexing, planner) allows it to preserve high performance at high load, which prepares it for highly concurrent environments.

6.2.4 Summarize

We initially expected PC 2 to be the top performer. However, somewhat surprisingly, PC 4 slightly outperformed it, despite running HammerDB under an x86_64 emulation layer on ARM64. On the other hand, PC 1, which is the only machine running on bare metal, was among the worst performers—only consistently outperforming PC 3 in MySQL and MariaDB tests, while showing mixed results in PostgreSQL. These findings suggest that disk speed may be the key performance factor in this setup. PC 1 uses the oldest and slowest disk, while PC 2 and PC 4 have similar storage performance despite different CPU architectures, what could explain the results. Meanwhile, despite having a fast SSD, PC 3 has the weakest CPU among all machines, what could have dragged it down.

6.3 Metrics of Different Machines

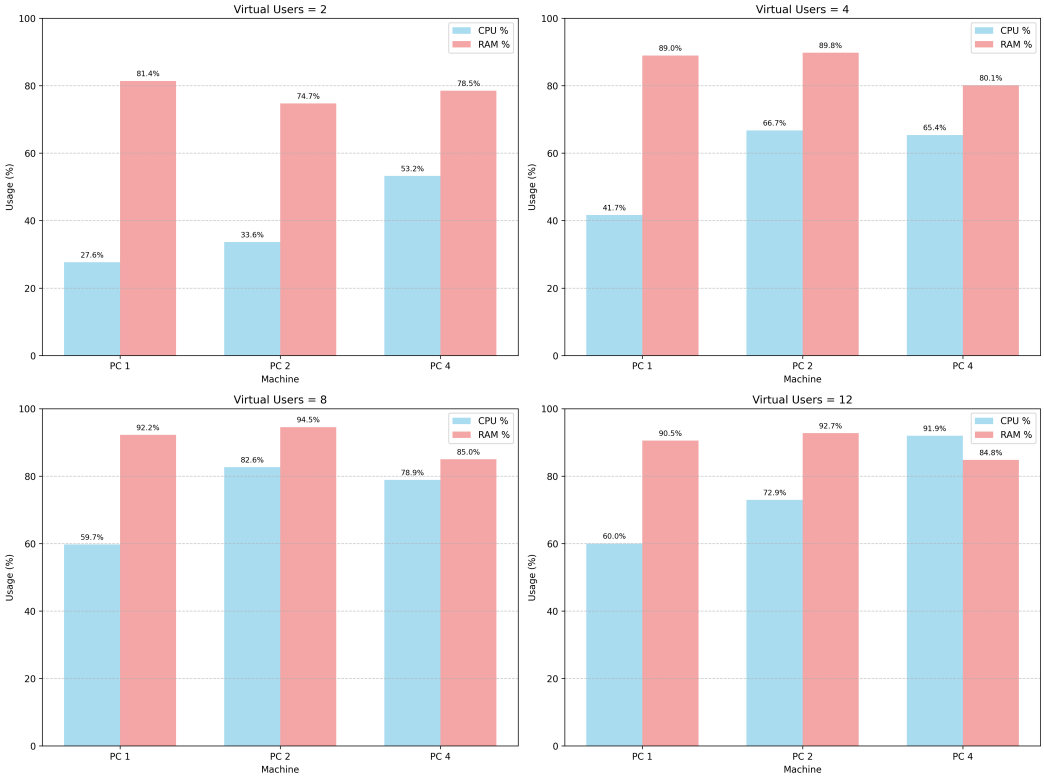


Figure 5: CPU and RAM usage on all tests using MariaDB

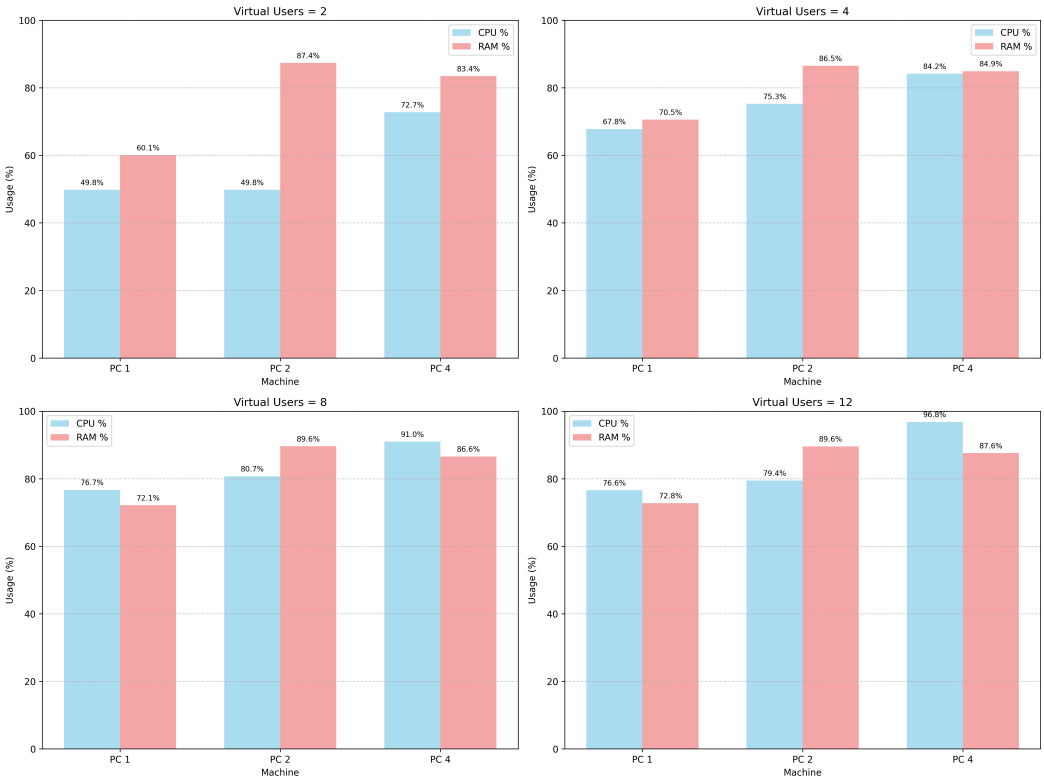


Figure 6: CPU and RAM usage on all tests using MySQL

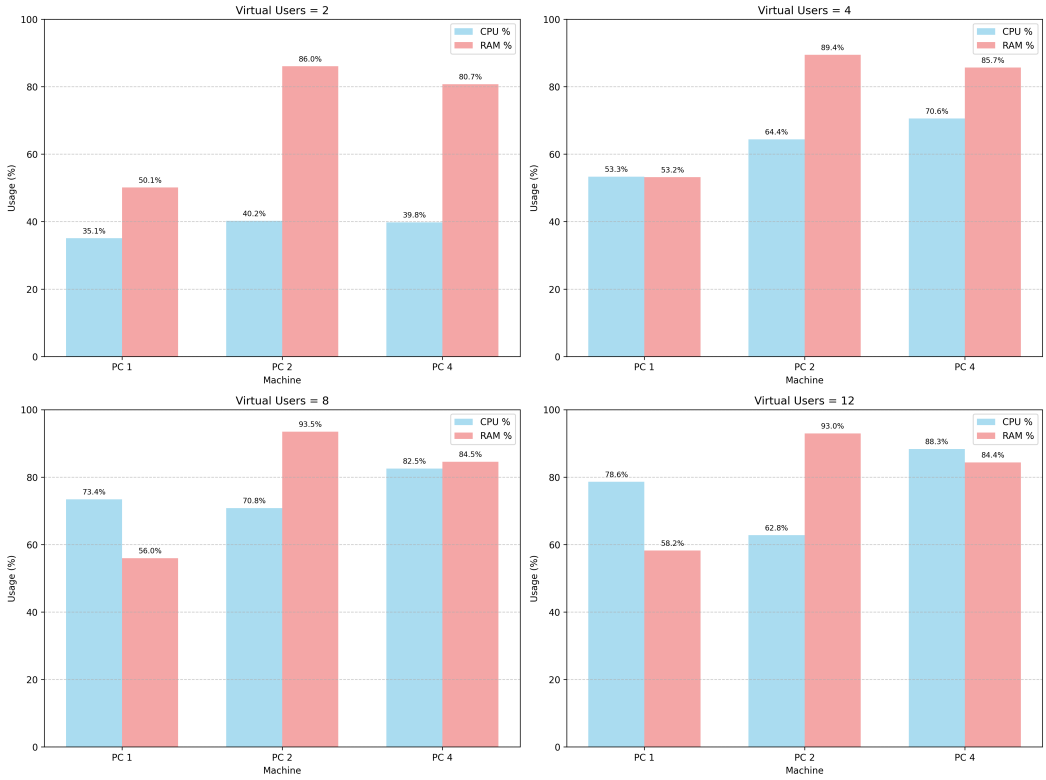


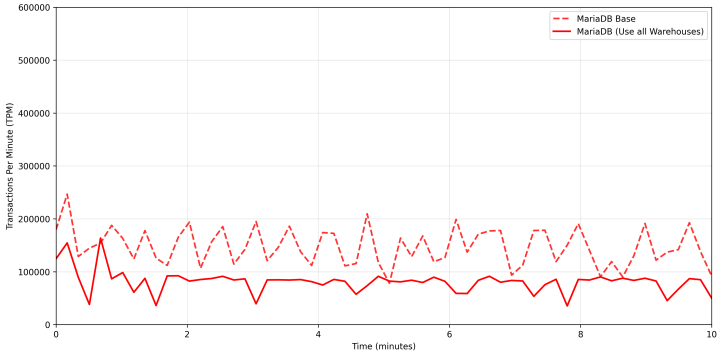
Figure 7: CPU and RAM usage on all tests using PostgreSQL

To ensure that there were no unexpected system bottlenecks that affected the benchmark results, we monitored both CPU and RAM usage during the tests with custom scripts. Disk usage was not recorded, as it is consistently at 100% on all machines and is the main bottleneck in this type of workload.

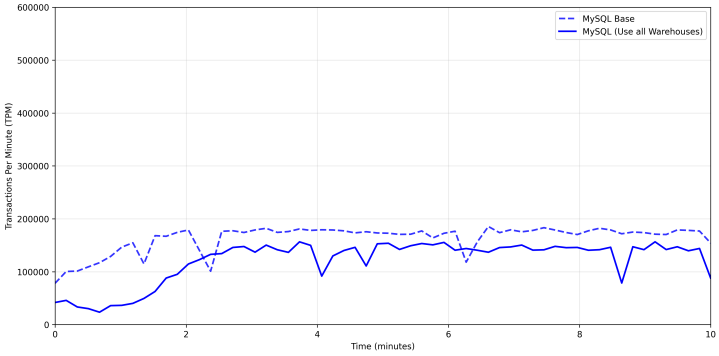
In general, **PC 1** exhibited lower CPU and RAM usage compared to the others. This was expected as the slower disk likely acted as the limiting factor, preventing the CPU and memory from being fully utilized.

When comparing **PC 2** and **PC 4**, the first showed lower CPU usage. This can be attributed to its higher core count, an Intel processor with 14 cores (6 performance cores and 8 efficiency cores), compared to the Apple Silicon chip with only 8 cores (4 performance and 4 efficiency cores). On the other hand, PC 4 demonstrated lower RAM usage, which is likely due to the more efficient memory management capabilities of macOS.

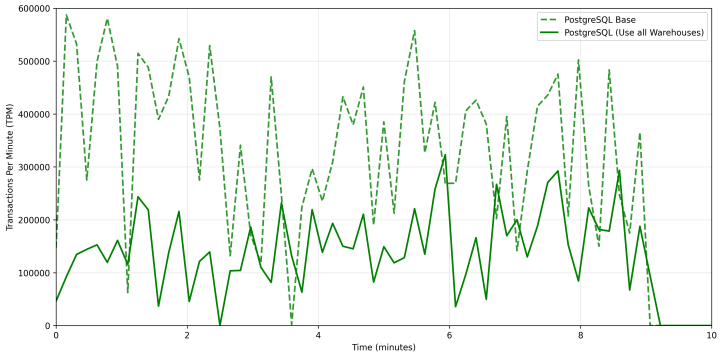
6.4 Use All Warehouses Comparison



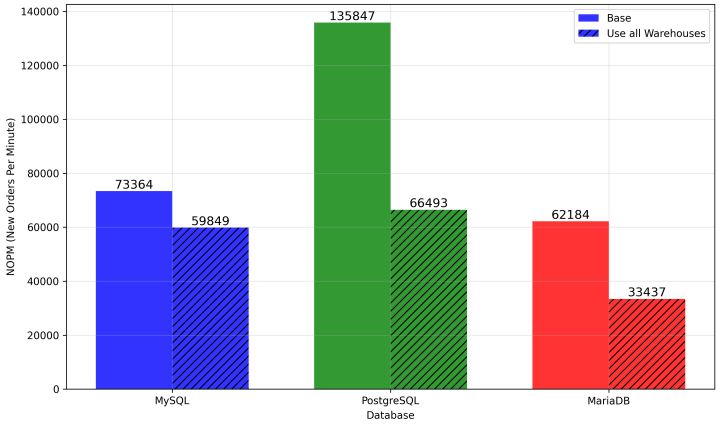
(a) Comparison in MariaDB



(b) Comparison in MySQL



(c) Comparison in PostgreSQL



(d) NOPM results for all Databases

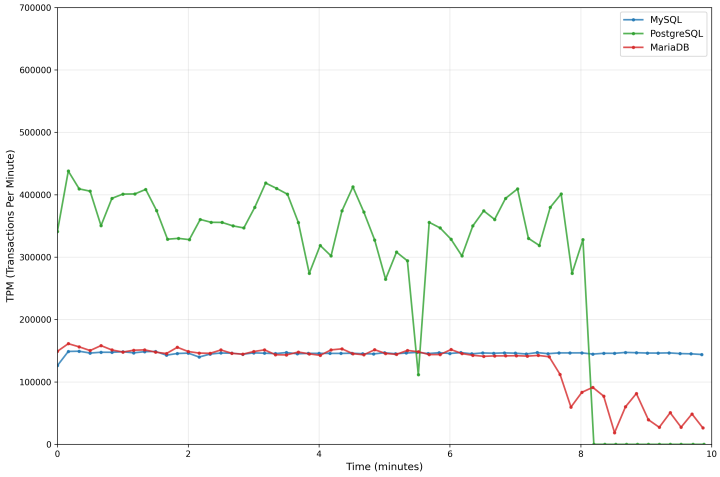
Figure 8: 'Use All Warehouses' performance analysis for VU=12 and WH=60 on PC 1

One of the configurable test parameters is the "Use All Warehouses" option, which simulates a more realistic multi-user environment by allowing clients to access different data partitions concurrently. This setup provides a better assessment of how well a database handles distributed load and concurrent access. However, it also introduces increased randomness in data access, reduces cache effectiveness, and places additional pressure on the disk subsystem. To better observe the impact of this option, the test was run on **PC 1**, which has the slowest disk among all machines.

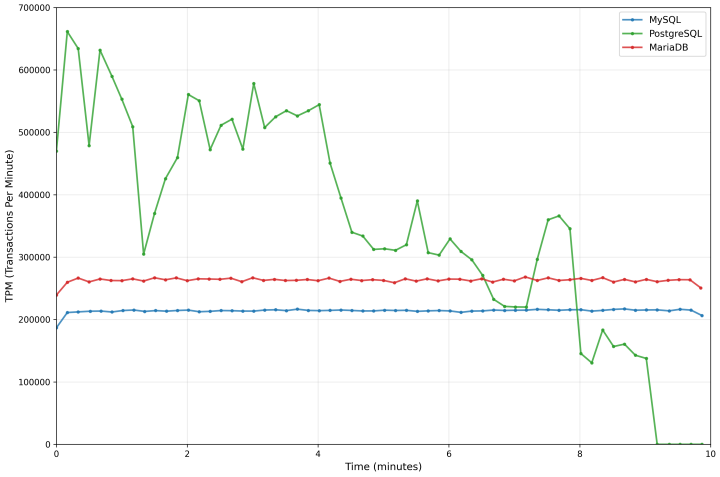
As expected, the average TPM dropped across all database configurations when the "Use All Warehouses" option was enabled, likely due to increased disk I/O demands. Interestingly, MySQL handled the change relatively well, with only an 18.4% drop in NOPM performance. In contrast, PostgreSQL and MariaDB experienced much steeper declines, losing 51.1% and 46.27% of their NOPM performance, respectively. This outcome was somewhat surprising, as PostgreSQL is known for its robust concurrency features. As for the differences between MySQL and MariaDB, they could be attributed to several factors, such as concurrency improvements in newer versions of InnoDB that are present in MySQL but not in MariaDB, or additional overhead in MariaDB's lock and thread management implementations, under high-concurrency and I/O-intensive workloads.

In summary, "Use All Warehouses" provides a more realistic and comprehensive performance scenario, which reveals potential I/O bottlenecks, especially on systems with limited storage throughput. It serves as a valuable stress test for high-concurrency environments.

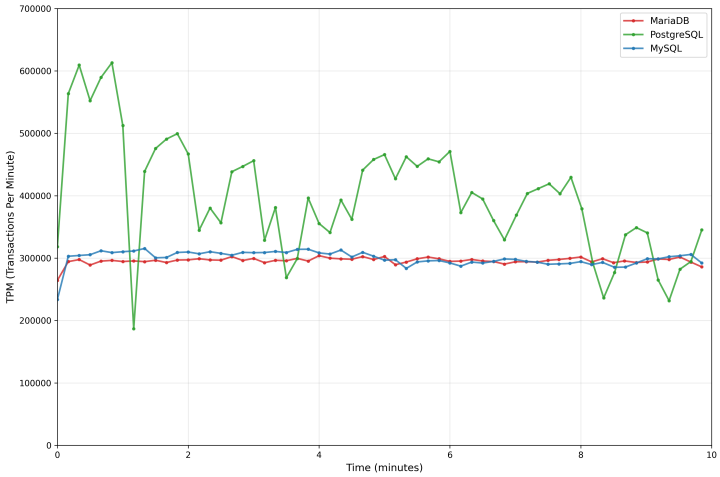
6.5 Final comparison between Databases



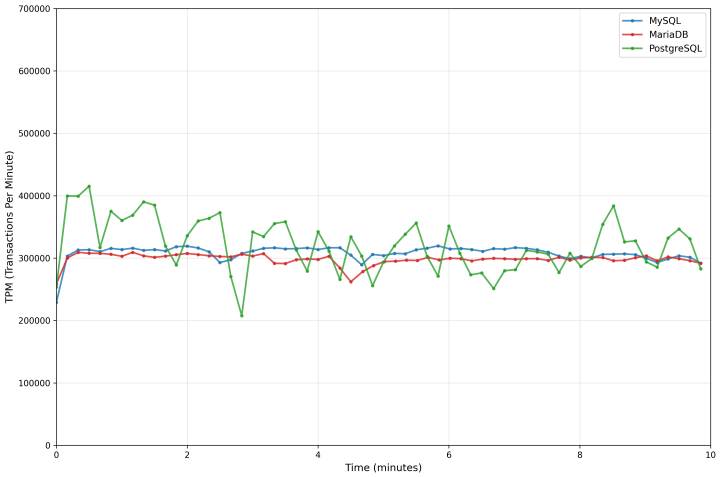
(a) Database results with 2 Virtual Users



(b) Database results with 4 Virtual Users



(c) Database results with 8 Virtual Users



(d) Database results with 12 Virtual Users

Figure 9: Performance of all Databases for PC 4, with different numbers of virtual users

This graphs show the differences between the three databases with varied concurrency levels: 2, 4, 8 and 12 Virtual Users (VUs) using **PC 4**.

As the previous benchmarks indicated, PostgreSQL continues to be the top-performing database, achieving the highest average TPM across most test scenarios. While MySQL and MariaDB demonstrate more consistent transaction rates, PostgreSQL delivers significantly higher peak throughput—the peak TPM recorded in Figure 9b is the highest observed across all tests, which was unexpected.

Interestingly, despite overall consistency across machines, this particular benchmark revealed a performance decline for PostgreSQL at 12 Virtual Users (Figure 9d). While this is likely influenced by the Apple M1's 8-core architecture, it also suggests that PostgreSQL does not handle the increased concurrency load as effectively as MySQL or MariaDB, which maintained steadier performance under the same conditions.

7 Discussion

The benchmark findings in this study carry useful data on how MySQL, PostgreSQL and MariaDB perform when subjected to TPROC-C workloads, which simulate atual online transaction processing (OLTP). The test setup was closely planned with identical configurations (where feasible), and changing concurrency levels.

PostgreSQL proved to be the most consistently performing DBMS. The architectural merits of MVCC, lightweight connection handling and a sophisticated query planner enabled it to scale very well with the number of virtual users. Event at high concurrency and distributed warehouse access, PostgreSQL reported higher TPM and NOPM figures than its competitors.

MySQL performed well, with particularly good results at moderate concurrency. The InnoDB storage engine, after being properly tuned, provided good scalability and transactional efficiency. It started to stagnate at full concurrency, indicating some contention or design limitation, but still in high regard as a general-purpose database management system.

MariaDB, while being a fork of MySQL, lagged behind on many test cases. Despite tuning optimizations, it could not keep up with MySQL under load, possibly due to differing implementations or handling of storage engines. It performed well under light workloads but showed saturation and lack of efficiency at high concurrency levels.

8 Conclusions

This study presents experimental performance comparison of PostgreSQL, MySQL and MariaDB with simulated OLTP workloads using TPROC-C benchmarks. The results are as follows:

- PostgreSQL performs the best in both throughput and scalability. It is particularly well suited to the demands of complex transactional systems with high concurrency.
- MySQL is also a strong candidate, offering competitive performance for most workloads, especially read-heavy or moderately concurrent workloads.
- MariaDB delivered performance very similar to MySQL, as expected. Interestingly, it appears to benefit slightly more from faster storage. This can be observed by comparing the results in Figures8 and2: in the former, where the test was run on a slower SSD, MariaDB performs slightly below MySQL; in the latter, which uses a faster SSD, MariaDB matches or slightly outperforms MySQL. This suggests that MariaDB may be more sensitive to disk speed, potentially due to differences in how it handles I/O or manages internal buffers.

Generally speaking, PostgreSQL is recommended for high-performance OLTP workloads requiring concurrency and consistency. MySQL is always a good and efficient choice and MariaDB is a better fit for less complicated situations or MySQL compatibility-oriented environments with less expected load.

9 Bibliography

[1] Wikipedia contributors. Hammerdb — Wikipedia, the free encyclopedia, 2025. URL: https://en.wikipedia.org/w/index.php?title=HammerDB&oldid=1275860580. [Online; accessed 25-May-2025].

[2] The PostgreSQL Global Development Group. *PostgreSQL 16 Documentation*. 2023. URL: https://www.postgresql.org/docs/16/index.html.

[3] MySQL Documentation Team. *MySQL 8.4 Reference Manual*. 2024. URL: https://dev.mysql.com/doc/refman/8.4/en/.

[4] MariaDB Foundation. *MariaDB Documentation*. 2024. URL: https://mariadb.com/kb/en/documentation/.

A TCL Scripts

A.1 MySQL TCL Script

Listing 1: TCL script for MySQL

```
#!/usr/bin/tclsh
# Set Database & Benchmark
dbset db mysql
dbset bm TPC-C

# DB configs
diset connection mysql_host 172.22.0.3
diset connection mysql_port 3306
diset connection mysql_socket "/var/run/mysqld/mysqld.sock"
diset tpcc mysql_user root
diset tpcc mysql_pass 1234
diset tpcc mysql_dbase tpcc

# Default for WH and VU
diset tpcc mysql_count_ware 50
diset tpcc mysql_num_vu 10

# Driver script options
diset tpcc mysql_timeprofile true
diset tpcc mysql_async_scale false
diset tpcc mysql_driver timed
# Ensure test is limited by time
diset tpcc mysql_total_iterations 1000000
# Timed duration
diset tpcc mysql_rampup 2
diset tpcc mysql_duration 8
# Distribute load
diset tpcc mysql_allwarehouse false

# Transactions options
tcset refreshrate 10
tcset logtotemp 1
tcset unique 1
tcset timestamps 1
# Run
foreach z {2 4 8 12} {
    set w [expr {$z * 5}]
    diset tpcc mysql_count_ware $w
    puts "Building Schema for $z TEST"
    # Delete possible previous data
    deleteschema
    vudestroy
    # Build and load
    buildschema
    vudestroy
    loadscript
    # Vuser options
    vuset delay 500
    vuset repeat 500
    vuset iterations 1
    vuset showoutput 0
    vuset logtotemp 1
    vuset unique 1
    vuset nobuff 0
    vuset timestamps 1
    puts "Starting $z VU TEST"
    tcstart
    vuset vu $z
    vucreate
    vurun
    puts "Waiting 1 for cleanup and collection"
    after 60000
    puts "Destroying VU"
    vudestroy
    tcstop
}
```

A.2 MariaDB TCL Script

Listing 2: TCL script for MariaDB

```
#!/usr/bin/tclsh
# Set Database & Benchmark
dbset db maria
dbset bm TPC-C

# DB configs
diset connection maria_host 172.22.0.4
diset connection maria_port 3306
diset connection mysql_socket "/var/run/mysqld/mysqld.sock"
diset tpcc maria_user root
diset tpcc maria_pass 1234
diset tpcc maria_dbase tpcc

# Default for WH and VU
diset tpcc maria_count_ware 50
diset tpcc maria_num_vu 10

# Driver script options
diset tpcc maria_timeprofile true
diset tpcc maria_async_scale false
diset tpcc maria_driver timed

# Ensure test is limited by time
diset tpcc maria_total_iterations 1000000

# Timed duration
diset tpcc maria_rampup 2
diset tpcc maria_duration 8
# Distribute load
diset tpcc maria_allwarehouse false

# Transactions options
tcset refreshrate 10
tcset logtotemp 1
tcset unique 1
tcset timestamps 1

# Run
foreach z {2 4 8 12} {
    set w [expr {$z * 5}]
    diset tpcc maria_count_ware $w
    puts "Building Schema for $z TEST"
    # Delete possible previous data
    deleteschema
    vudestroy
    # Build and load
    buildschema
    vudestroy
    loadscript
    # Vuser options
    vuset delay 500
    vuset repeat 500
    vuset iterations 1
    vuset showoutput 0
    vuset logtotemp 1
    vuset unique 1
    vuset nobuff 0
    vuset timestamps 1
    puts "Starting $z VU TEST"
    tcstart
    vuset vu $z
    vucreate
    vurun
    puts "Waiting 1 for cleanup and collection"
    after 60000
    puts "Destroying VU"
    vudestroy
    tcstop
}
```

A.3 PostgreSQL TCL Script

Listing 3: TCL script for PostgreSQL

```
#!/usr/bin/tclsh
# Set Database & Benchmark
dbset db pg
dbset bm TPC-C

# DB configs
diset connection pg_host 172.22.0.2
diset connection pg_port 5432
diset connection pg_sslmode prefer
diset tpcc pg_superuser postgres
diset tpcc pg_superuserpass 1234
diset tpcc pg_defaultdb postgres
diset tpcc pg_user tpcc
diset tpcc pg_pass tpcc
diset tpcc pg_dbase tpcc
diset tpcc pg_tspace pg_default

# Default for WH and VU
diset tpcc pg_count_ware 50
diset tpcc pg_num_vu 10

# Driver script options
diset tpcc pg_timeprofile true
diset tpcc pg_async_scale false
diset tpcc pg_driver timed
# Ensure test is limited by time
diset tpcc pg_total_iterations 1000000
# Timed duration
diset tpcc pg_rampup 2
diset tpcc pg_duration 8
# Distribute load
diset tpcc pg_allwarehouse false
# Transactions options
tcset refreshrate 10
tcset logtotemp 1
tcset unique 1
tcset timestamps 1
# Run
foreach z {2 4 8 12} {
    set w [expr {$z * 5}]
    diset tpcc pg_count_ware $w
    puts "Building Schema for $z TEST"
    # Delete possible previous data
    deleteschema
    vudestroy
    # Build and load
    buildschema
    vudestroy
    loadscript
    # Vuser options
    vuset delay 500
    vuset repeat 500
    vuset iterations 1
    vuset showoutput 0
    vuset logtotemp 1
    vuset unique 1
    vuset nobuff 0
    vuset timestamps 1
    puts "Starting $z VU TEST"
    tcstart
    vuset vu $z
    vucreate
    vurun
    puts "Waiting 1 for cleanup and collection"
    after 60000
    puts "Destroying VU"
    vudestroy
    tcstop
}
```

B Docker

B.1 Docker Compose File

Listing 4: Docker Compose file for the databases and HammerDB

```
version: "3.9"

services:
  mysql:
    build:
      context: .
      dockerfile: Dockerfiles/MySQL/Dockerfile
    restart: always
    container_name: MySQL_SBD
    environment:
      MYSQL_ROOT_PASSWORD: 1234
    volumes:
      - mysql-volume:/var/lib/mysql
    networks:
      sbd_network:
        ipv4_address: 172.22.0.3

  postgres:
    image: postgres
    container_name: Postgres_SBD
    command: postgres -c config_file=/etc/postgresql/postgresql.conf
    environment:
      POSTGRES_PASSWORD: "1234"
    volumes:
      - postgres-volume:/var/lib/postgresql/data
      - ./Configs/Hard_Test/postgresql.conf:/etc/postgresql/postgresql.conf:ro
    networks:
      sbd_network:
        ipv4_address: 172.22.0.2

  mariadb:
    build:
      context: .
      dockerfile: Dockerfiles/MariaDB/Dockerfile
    restart: always
    container_name: MariaDB_SBD
    environment:
      MYSQL_ROOT_PASSWORD: 1234
      MYSQL_DATABASE: sbdDatabase
    volumes:
      - mariadb-volume:/var/lib/mysql:rw
    networks:
      sbd_network:
        ipv4_address: 172.22.0.4

  hammer-gui:
    image: tpcorg/hammerdb:latest-cloudtk
    container_name: HammerDB_SBD
    restart: always
    ports:
      - "8081:8081"
      - "8082:8082"
      - "8080:8080"
    depends_on:
      - mysql
      - postgres
      - mariadb
    volumes:
      - ./Scripts:/home/HammerDB-4.10/scripts/tcl-scripts
      - ./stats-logs:/exp-logs
    networks:
      sbd_network:
        ipv4_address: 172.22.0.5

volumes:
  mysql-volume:
```

```
    driver: local
  postgres-volume:
    driver: local
  mariadb-volume:
    driver: local

networks:
  sbd_network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.22.0.0/16
```

B.2 Dockerfile for MySQL

Listing 5: Dockerfile for MySQL

```
FROM mysql:8.4

RUN mkdir -p /var/log/mysql && \
    chown -R mysql:mysql /var/log/mysql

COPY ./Configs/Hard_Test/mysql.cnf /etc/mysql/conf.d/mysql.cnf

RUN chmod 644 /etc/mysql/conf.d/mysql.cnf
```

B.3 Dockerfile for MariaDB

Listing 6: Dockerfile for MariaDB

```
FROM mariadb:11.4

COPY ./Configs/Hard_Test/my.cnf /etc/mysql/my.cnf

RUN chmod 644 /etc/mysql/my.cnf
```

C Configs

C.1 MySQL Configuration File

Listing 7: MySQL configuration file

```
[mysqld]

#------------------------------------------------------------------------------
# BASIC SETTINGS
#------------------------------------------------------------------------------
port = 3306
bind-address = 0.0.0.0
max_connections = 1000
max_connect_errors = 1000000
wait_timeout = 28800
interactive_timeout = 28800
connect_timeout = 10
back_log = 1500

#------------------------------------------------------------------------------
# CHARACTER SET & COLLATION
#------------------------------------------------------------------------------
character-set-server = utf8mb4
collation-server = utf8mb4_unicode_ci

#------------------------------------------------------------------------------
# STORAGE AND INNODB ENGINE
#------------------------------------------------------------------------------
innodb_buffer_pool_size = 4G
innodb_buffer_pool_instances = 4
innodb_log_file_size = 512M
innodb_log_buffer_size = 64M
innodb_file_per_table = 1
innodb_flush_log_at_trx_commit = 2
innodb_flush_method = O_DIRECT
innodb_io_capacity = 2000
innodb_io_capacity_max = 4000
innodb_read_io_threads = 8
innodb_write_io_threads = 8
innodb_purge_threads = 4
innodb_doublewrite = 1
innodb_autoinc_lock_mode = 2
innodb_stats_persistent = 1
innodb_lru_scan_depth = 2048
innodb_adaptive_flushing = 1
innodb_adaptive_hash_index = 0
innodb_change_buffering = none

#------------------------------------------------------------------------------
# TEMPORARY TABLE & BUFFERS
#------------------------------------------------------------------------------
tmp_table_size = 256M
max_heap_table_size = 256M
sort_buffer_size = 1M
join_buffer_size = 1M
read_buffer_size = 512K
read_rnd_buffer_size = 2M

#------------------------------------------------------------------------------
# LOGGING
#------------------------------------------------------------------------------
slow_query_log = 1
long_query_time = 2
slow_query_log_file = /var/lib/mysql/mysql-slow.log
general_log = 0

#------------------------------------------------------------------------------
# BYNARY LOGGING
#------------------------------------------------------------------------------
skip-log-bin
sync_binlog = 0
```

```
67
68 #------------------------------------------------------------------------------
69 # SECURITY & COMPATIBILITY
70 #------------------------------------------------------------------------------
71 local_infile = 0
72 sql_mode = STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION
73 transaction_isolation = REPEATABLE-READ
74
75 #------------------------------------------------------------------------------
76 # PERFORMANCE
77 #------------------------------------------------------------------------------
78 performance_schema = OFF
79 max_prepared_stmt_count = 12800
80 table_open_cache = 2000
81 table_open_cache_instances = 4
82 open_files_limit = 65535
83 thread_cache_size = 50
84 thread_stack = 256K
85
86 #------------------------------------------------------------------------------
87 # MONITORING
88 #------------------------------------------------------------------------------
89 innodb_monitor_enable = '%'
90
91 #------------------------------------------------------------------------------
92 # PLUGIN & AUTHENTICATION
93 #------------------------------------------------------------------------------
94 require_secure_transport = OFF
95 caching_sha2_password_auto_generate_rsa_keys = ON
```

C.2 MariaDB Configuration File

Listing 8: MariaDB configuration file

```
1  [mysqld]
2
3  #------------------------------------------------------------------------------
4  # BASIC SETTINGS
5  #------------------------------------------------------------------------------
6  port = 3306
7  bind-address = 0.0.0.0
8  max_connections = 1000
9  max_connect_errors = 1000000
10 wait_timeout = 28800
11 interactive_timeout = 28800
12 connect_timeout = 10
13 back_log = 1500
14
15 #------------------------------------------------------------------------------
16 # CHARACTER SET & COLLATION
17 #------------------------------------------------------------------------------
18 character-set-server = utf8mb4
19 collation-server = utf8mb4_unicode_ci
20
21 #------------------------------------------------------------------------------
22 # STORAGE AND INNODB ENGINE
23 #------------------------------------------------------------------------------
24 innodb_buffer_pool_size = 4G
25 innodb_buffer_pool_instances = 4
26 innodb_log_file_size = 512M
27 innodb_log_buffer_size = 64M
28 innodb_file_per_table = 1
29 innodb_flush_log_at_trx_commit = 2
30 innodb_flush_method = O_DIRECT
31 innodb_io_capacity = 2000
32 innodb_io_capacity_max = 4000
33 innodb_read_io_threads = 8
34 innodb_write_io_threads = 8
35 innodb_purge_threads = 4
36 innodb_doublewrite = 1
37 innodb_autoinc_lock_mode = 2
```

```
innodb_stats_persistent = 1
innodb_lru_scan_depth = 2048
innodb_adaptive_flushing = 1
innodb_adaptive_hash_index = 0
innodb_change_buffering = none

#------------------------------------------------------------------------------
# TEMPORARY TABLE & BUFFERS
#------------------------------------------------------------------------------
tmp_table_size = 256M
max_heap_table_size = 256M
sort_buffer_size = 1M
join_buffer_size = 1M
read_buffer_size = 512K
read_rnd_buffer_size = 2M

#------------------------------------------------------------------------------
# LOGGING
#------------------------------------------------------------------------------
slow_query_log = 1
long_query_time = 2
slow_query_log_file = /var/lib/mysql/mysql-slow.log
general_log = 0

#------------------------------------------------------------------------------
# BYNARY LOGGING
#------------------------------------------------------------------------------
skip-log-bin
sync_binlog = 0

#------------------------------------------------------------------------------
# SECURITY & COMPATIBILITY
#------------------------------------------------------------------------------
local_infile = 0
sql_mode = STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION
transaction_isolation = REPEATABLE-READ

#------------------------------------------------------------------------------
# PERFORMANCE
#------------------------------------------------------------------------------
performance_schema = OFF
max_prepared_stmt_count = 12800
table_open_cache = 2000
table_open_cache_instances = 4
open_files_limit = 65535
thread_cache_size = 50
thread_stack = 256K

#------------------------------------------------------------------------------
# MONITORING
#------------------------------------------------------------------------------
innodb_monitor_enable = '%'
```

C.3 PostgreSQL Configuration File

Listing 9: PostgreSQL configuration file

```
#------------------------------------------------------------------------------
# FILE LOCATIONS
#------------------------------------------------------------------------------
data_directory = '/var/lib/postgresql/data' # Important: match Docker volume
hba_file = '/var/lib/postgresql/data/pg_hba.conf'
ident_file = '/var/lib/postgresql/data/pg_ident.conf'

#------------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------
listen_addresses = '*' # Allow external connections (Docker host network)
port = 5432 # Default PostgreSQL port
max_connections = 100

#------------------------------------------------------------------------------
# RESOURCE USAGE
#------------------------------------------------------------------------------
shared_buffers = 512MB # Adjust depending on host memory (e.g., 25% of RAM)
work_mem = 64MB # Suitable for OLTP like TPC-C
maintenance_work_mem = 256MB
effective_cache_size = 2GB # Depends on total system RAM

#------------------------------------------------------------------------------
# WRITE-AHEAD LOG
#------------------------------------------------------------------------------
wal_level = replica
synchronous_commit = off # Can improve write performance (acceptable for benchmarks)
checkpoint_timeout = 15min
checkpoint_completion_target = 0.9
max_wal_size = 2GB
min_wal_size = 512MB

#------------------------------------------------------------------------------
# LOGGING (optional, but useful for debugging)
#------------------------------------------------------------------------------
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql.log'
log_statement = 'none'
log_min_duration_statement = 1000 # Log queries slower than 1s
```

D Runner Scripts

D.1 Batch Script

Listing 10: Batch script to run the TCL scripts

```
@echo off
setlocal

:: Set container names
set HAMMER_CONTAINER=HammerDB_SBD

echo Starting HammerDB benchmark runner...

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: Setup containers
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

:: Compose down to ensure a clean start
echo Stopping and removing existing containers, images and volumes...
docker compose down --rmi all -v

timeout /t 5

:: Compose up to start the containers
echo Starting containers...
docker compose up -d

timeout /t 10

echo Containers are up and running!

echo Starting HammerDB benchmark...

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: POSTGRESQL
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

echo Starting up PostgreSQL benchmark...

:: Start the stats logging in a new window using the temp file
start "PCStats" cmd /c PCStats.bat "postgres"

timeout /t 5 >nul

docker exec -i %HAMMER_CONTAINER% /home/HammerDB-4.10/hammerdbcli auto scripts/tcl-scripts/largeTestPG.tcl

for /f "tokens=2 delims=," %%i in ('tasklist /v /fo csv ^| findstr /i /c:"PCStats"') do (
    taskkill /PID %%~i /F >nul 2>&1
)

echo Benchmark and monitoring complete for PostgreSQL.

timeout /t

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: MYSQL
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

echo Starting up MySQL benchmark...

:: Start the stats logging in a new window using the temp file
start "PCStats" cmd /c PCStats.bat "mysql"

timeout /t 5 >nul

docker exec -i %HAMMER_CONTAINER% /home/HammerDB-4.10/hammerdbcli auto scripts/tcl-scripts/largeTestMySQL.
    ↪ tcl

for /f "tokens=2 delims=," %%i in ('tasklist /v /fo csv ^| findstr /i /c:"PCStats"') do (
    taskkill /PID %%~i /F >nul 2>&1
)
```

```

echo Benchmark and monitoring complete for MySQL.

timeout /t 30

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: MARIADB
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

echo Starting up MariaDB benchmark...

:: Start the stats logging in a new window using the temp file
start "PCStats" cmd /c PCStats.bat "mariadb"

timeout /t 5 >nul

docker exec -i %HAMMER_CONTAINER% /home/HammerDB-4.10/hammerdbcli auto scripts/tcl-scripts/
    ↪ largeTestMariaDB.tcl

for /f "tokens=2 delims=," %%i in ('tasklist /v /fo csv ^| findstr /i /c:"PCStats"') do (
    taskkill /PID %%~i /F >nul 2>&1
)

echo Benchmark and monitoring complete for MariaDB.

echo Benchmark and monitoring complete.
endlocal
```

D.1.1 PCStats Script

Listing 11: PCStats script to monitor the system in Windows

```
@echo off
setlocal enabledelayedexpansion

set DB=%1

:: Format date and time safely for filename
for /f "tokens=1-3 delims=/" %%a in ("%date%") do (
    set mm=%%a
    set dd=%%b
    set yyyy=%%c
)
for /f "tokens=1-3 delims=:." %%a in ("%time%") do (
    set hh=%%a
    set min=%%b
    set ss=%%c
)
set "logfile=sys_usage_%mm%_%dd%_%hh%_%min%_%DB%.csv"

:: Set interval (in seconds)
set interval=10

:: Write CSV header
echo Timestamp,CPU_Usage_Percent,RAM_Used_MB > "%logfile%"

:loop
:: Get timestamp in ISO format using PowerShell
for /f %%x in ('powershell -command "Get-Date -Format yyyy-MM-dd_HH:mm:ss"') do set timestamp=%%x

echo Current timestamp: %timestamp%

:: Get CPU usage
for /f "skip=1" %%x in ('wmic cpu get loadpercentage') do (
    if not "%%x"=="" (
        set cpu=%%x
        goto :gotCPU
    )
)
:gotCPU

```

```
:: Get RAM usage (in MB)
for /f "skip=1 tokens=2,3 delims=," %%a in ('wmic OS get FreePhysicalMemory^,TotalVisibleMemorySize /
    ↪ format:csv') do (
    set "free=%%a"
    set "total=%%b"
)
set /a usedMB=(%total% - %free%) / 1024

:: Write to CSV
echo %timestamp%,%cpu%,%usedMB% >> "%logfile%"

:: Wait and repeat
timeout /t %interval% >nul
goto loop
```

D.2 Shell Script

Listing 12: Shell script to run the TCL scripts

```
#!/bin/bash

# Set your container names
HAMMER_CONTAINER=HammerDB_SBD

echo "Starting HammerDB benchmark runner..."

######################################################################
# Setup containers
######################################################################

# Compose down to ensure a clean start
echo "Stopping and removing existing containers, images and volumes..."

docker compose down --rmi all -v

sleep 5

# Compose up to start the containers
echo "Starting containers..."

docker compose up -d

sleep 10

echo "Containers are up and running!"

echo "Starting HammerDB benchmark..."

######################################################################
# POSTGRESQL
######################################################################

echo "Starting up PostgreSQL benchmark..."

# Start the stats logging in the background
./PCStats.sh "postgres" &
PCSTATS_PID=$!

# Wait 5 seconds for stats logger to initialize
sleep 5

# Run the benchmark via Docker
docker exec -i "$HAMMER_CONTAINER" /home/HammerDB-4.10/hammerdbcli auto scripts/tcl-scripts/largeTestPG.
    ↪ tcl

# Kill the PCStats logger
kill "$PCSTATS_PID"

echo "Benchmark and monitoring complete for PostgreSQL."

# Wait 30 seconds before exiting
```

```
sleep 30

######################################################################
# MYSQL
######################################################################

echo "Starting up MySQL benchmark..."

# Start the stats logging in the background
./PCStats.sh "mysql" &
PCSTATS_PID=$!

# Wait 5 seconds for stats logger to initialize
sleep 5

# Run the benchmark via Docker
docker exec -i "$HAMMER_CONTAINER" /home/HammerDB-4.10/hammerdbcli auto scripts/tcl-scripts/largeTestMySQL
    ↪ .tcl

# Kill the PCStats logger
kill "$PCSTATS_PID"

echo "Benchmark and monitoring complete for MySQL."

# Wait 30 seconds before exiting
sleep 30

######################################################################
# MARIADB
######################################################################

echo "Starting up MariaDB benchmark..."

# Start the stats logging in the background
./PCStats.sh "mariadb" &
PCSTATS_PID=$!

# Wait 5 seconds for stats logger to initialize
sleep 5

# Run the benchmark via Docker
docker exec -i "$HAMMER_CONTAINER" /home/HammerDB-4.10/hammerdbcli auto scripts/tcl-scripts/
    ↪ largeTestMariaDB.tcl

# Kill the PCStats logger
kill "$PCSTATS_PID"

echo "Benchmark and monitoring complete for MariaDB."

echo "Benchmark and monitoring complete."
```

D.2.1 PCStats Script

Listing 13: PCStats script to monitor the system in Linux

```
#!/bin/bash

DB="$1"
interval=10

# Create log file name based on current date and time
timestamp=$(date "+%m_%d_%H_%M")
logfile="sys_usage_${timestamp}_${DB}.csv"

# Write CSV header
echo "Timestamp,CPU_Usage_Percent,RAM_Used_MB" > "$logfile"

while true; do
    # Get current timestamp
    timestamp=$(date "+%Y-%m-%d_%H:%M:%S")
    echo "Current timestamp: $timestamp"
```

```
17
18      # Get CPU usage percentage (average over 1 second)
19      cpu=$(top -bn2 | grep "Cpu(s)" | tail -n 1 | awk -F'id,' -v prefix="" '{ split($1, vs, ","); cpu=100 -
            ↪  vs[length(vs)]; printf "%.0f", cpu }')
20
21      # Get RAM usage in MB
22      mem_total=$(free -m | awk '/Mem:/ {print $2}')
23      mem_used=$(free -m | awk '/Mem:/ {print $3}')
24
25      # Write to CSV
26      echo "$timestamp,$cpu,$mem_used" >> "$logfile"
27
28      # Wait for interval
29      sleep "$interval"
30  done
```