Sistemas de Base de Dados

Martin Magdalinchev & Rafael Mira & James Furtado
58172 & 59243 & 61177

Faculdade de Ciências e Tecnologia
Universidade NOVA de Lisboa

June 9, 2024

1 Introduction

The aim of this project is to gain practical experience in benchmarking database systems using HammerDB, a leading open benchmarking tool and to acquire knowledge on how the database configuration can have such a impact on the system performance. Our project will focus on evaluating PostgreSQL performance under different configurations and workloads. The key objectives include analyzing the impact on the database performance by varying the number of virtual users connecting to the database, changing some configuration parameters and using a HDD versus a SDD.

These objectives will be investigated using HammerDB benchmarks, specifically TPROC-C for OLTP workloads.

2 HammerDB Overview

HammerDB is an open-source benchmarking and load testing software that supports the worlds most popular databases, such as PostgreSQL, Oracle Database, Microsoft SQL Server, and others. It offers two primary benchmarks: TPROC-C (for OLTP workloads) and TPROC-H (for OLAP workloads).

In our project, we will use the TPC-C benchmark, or more correctly, the TPROC-C benchmark which is the a hammerDB version of the TPC-C benchmark where the workload is simpler and easier to run. This is a type of benchmark used to measure database performance in the area of transaction processing. It simulates a e-commerce environment, replicating scenarios such as order placement, inventory management and payment processing. Through a benchmark simulation, TPC-C evaluates the ability of a database system to handle concurrent transactions efficiently, providing insights into its scalability and reliability.

HammerDB TPROC-C will run a CPU and memory intensive version of the TPC-C workload by having the number of virtual users and the required data set to be much smaller than a full TPC-C implementation to reach maximum levels of performance.

3 Chosen DBMS

PostgreSQL is a powerful, open-source object-relational database system. It is known for its robustness, extensibility, and standards compliance. Key features include full ACID compliance and support for complex queries and a wide range of data types.

4 Methodology to perform the benchmarking

4.1 Machines used in the tests

To run our experiments we used four different machines with different configurations. Two of the machines are from the University cluster, namely, machine 3 and 4, and the other machines are our own personal machines.

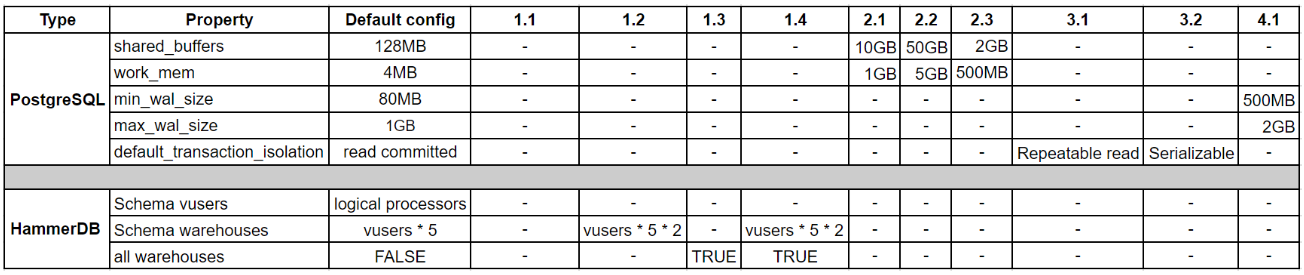
Id	OS	CPU	Cores	Threads	RAM	Disk
1	Windows 10	Intel(R) Core(TM) i5-9300H @ 2.40GHz	4	8	8GB	SSD
2	Windows 11	Intel(R) Core(TM) i7-12700H @ 2.70GHz	14	20	16GB	SSD
3	Linux	AMD EPYC 7281	16	32	128GB	HDD
4	Linux	2 x Intel Xeon Gold 6346	64	128	128GB	SSD

4.2 DBMS and HammerDB Configurations

To describe our benchmarks configurations in a concise way we have this table with PostgreSQL and hammerDB configurations for our experiments. We ran 10 different experiments, four of them being with the default PostgreSQL configuration and the remaining with some different values for some parameters.

In the first part we have the values for the default configuration that comes with PostgreSQL and also the values used in the experiments that we did. When a row has this value '-', it represents that the parameters has not altered for that experiment.

In the second part we have the hammerDB properties we changed during the experiments. For most o the experiments, we didn't changed the number of warehouses neither the number of virtual users used on the schema creation.



Type	Property	Default config	1.1	1.2	1.3	1.4	2.1	2.2	2.3	3.1	3.2	4.1
PostgreSQL	shared_buffers	128MB	-	-	-	-	10GB	50GB	2GB	-	-	-
	work_mem	4MB	-	-	-	-	1GB	5GB	500MB	-	-	-
	min_wal_size	80MB	-	-	-	-	-	-	-	-	-	500MB
	max_wal_size	1GB	-	-	-	-	-	-	-	-	-	2GB
	default_transaction_isolation	read committed	-	-	-	-	-	-	-	Repeatable read	Serializable	-
HammerDB	Schema vusers	logical processors	-	-	-	-	-	-	-	-	-	-
	Schema warehouses	vusers * 5	-	vusers * 5 * 2	-	vusers * 5 * 2	-	-	-	-	-	-
	all warehouses	FALSE	-	-	TRUE	TRUE	-	-	-	-	-	-

Figure 1: Experiments configurations.

For the creation of the schema, we used the number of threads of the computers where the experiments ran to define the number of virtual users. To define the number of warehouses we use the number of virtual users multiplied by 5.

When running the experiments, we used a .tcl script to automate the process of running multiple workloads for different virtual users in each test case. For each experiment, we used 8, 16, 32, 64 and 100 virtual users (independently of the number of virtual users used in the schema build).

The remaining configurations of hammerDB were the same throughout the experiments and can be found in detail in the .tcl script we used to run the experiments.

Now let's explain the database configurations that we changed. The shared_buffers property in the buffer sizer of the database in memory. This is set to 128MB which is a very modest size for a buffer. The wor_mem property is the buffer size for the sorting and hasing operation. Increasing this can also have a large impact on the database performance, depending on the queries and exectuon plans that are chosen.

Other configuration was the transactional level of isolation of the database, which is set to read commited as default. This can have a large impact on performance but is also a trade-off between performance and consistency.

Then we also messed with th min_wal_size and the max_wal_size wich are paremeters to control the minimum and maximum size that the in memory log can have before being written to stable storage.

4.3 Metodology

To run the benchmarks, we tried to reduce the bottleneck that our machines configurations could introduce in the results by having only the hammerdbcli running as well with the PostgreSQL server and the necessary services for the OS. To run the benchmarks in the cluster we used docker swarm to achieve the most accurate results possible by having the container with PostgreSQL running in a cluster node and the hammerdbcli running in another node.

During this benchmarks we collect metrics about the TPM - Transactions per minute, the procedure metrics for the five procedures that occur in the tests (NEWORD, PAYMENT, SLEV, DELIVERY, OSTAT) which contains information about the P99, P95, P90, max, min, avg, number of procedure calls, standard deviation and the total time the procedure executed.

Note: We tried to collect PostgreSQL metrics using pgsentinel extension but the configuration is not trivial and we ended up losing to much time trying to set pgsentinel so we reach a point where we gave up and collect cpu, ram and I/O metrics in a different way. To collect the metrics from the cluster machines we used the metrics that docker provides for the running container and for the windows machines we used a combination of batch and powershell scrips to achieve the same effect by monitoring the hammerDB and PostgreSQL server processes that were running.

5 Results

5.1 Test 1.1 - Default configuration

This is the first test we executed. It has the default configurations, as explained in section 4.2 and will serve as base for comparison with the rest of the tests.

5.1.1 Machine 1 Vs Machine 2 (windows)

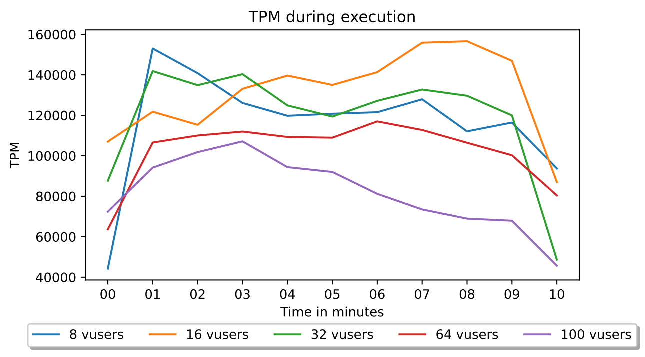


Figure 2: Machine 1 - Default configuration

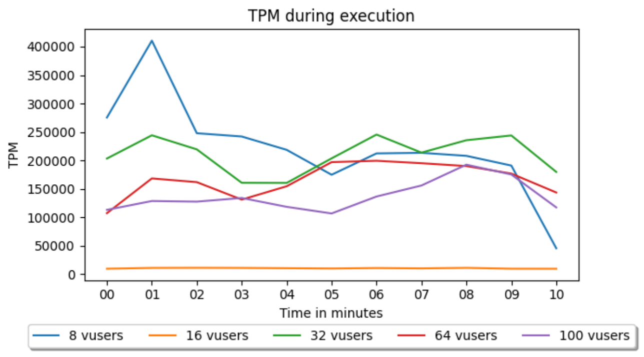


Figure 3: Machine 2 - Default configuration

Comparing the machines 1 and 2, we can see the second one had a better performance, as expected, since it has double the RAM and more cores as well. A interesting observation is the result for 16 virtual users of machine 2, which despite being around 8 to 12 thousands TPM, is much less than the results for the others virtual users.

5.1.2 Machine 3 VS Machine 4 (linux)

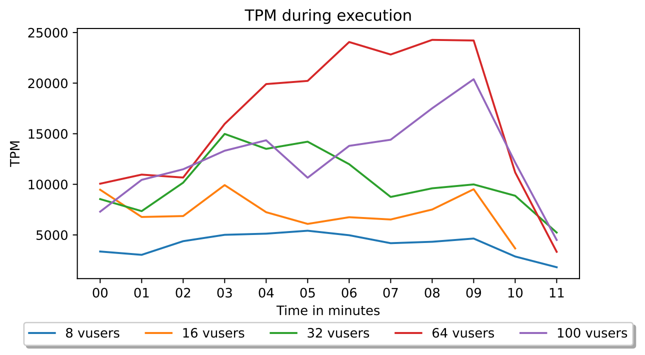


Figure 4: Machine 3 - Default configuration

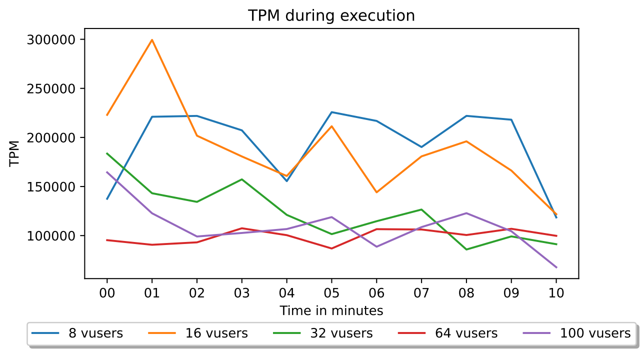


Figure 5: Machine 4 - Default configuration

The results indicate a significant performance disparity between HDD and SSD storage. On an HDD-equipped machine, the average Transactions Per Minute (TPM) hover around a few tens of thousands. In contrast, on an SSD-equipped machine, the average TPM skyrockets to several hundred thousand, demonstrating a tenfold increase in performance.

We do know that the machine 4 has twice as many cores as machine 3 but as we will see later, when using a bigger buffer size on machine 3, the usage of SSD does play a major role on the results.

5.2 Test 1.2 - Double of the warehouses

5.2.1 Machine 1 - 40 warehouses vs 80 warehouses

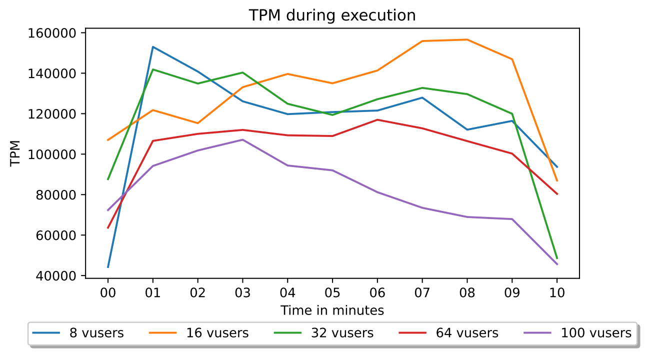


Figure 6: Machine 1 - 40 warehouses

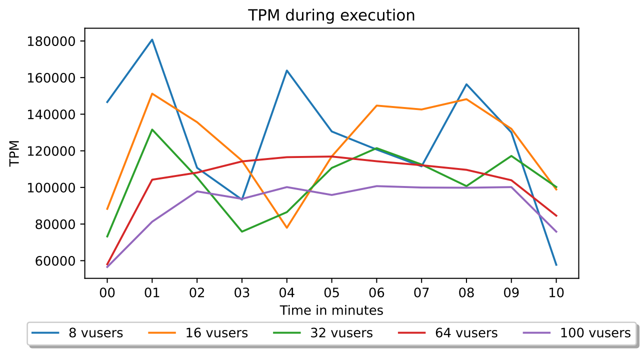


Figure 7: Machine 1 - 80 warehouses

Doubling the warehouses did not improved the performance, in fact, the performance became less constant than with less warehouses. A interesting fact is on the 100 vusers experiment where the performance improved and become stable with 80 warehouses than with 40.

5.2.2 Machine 2 - 100 warehouses vs 200 warehouses

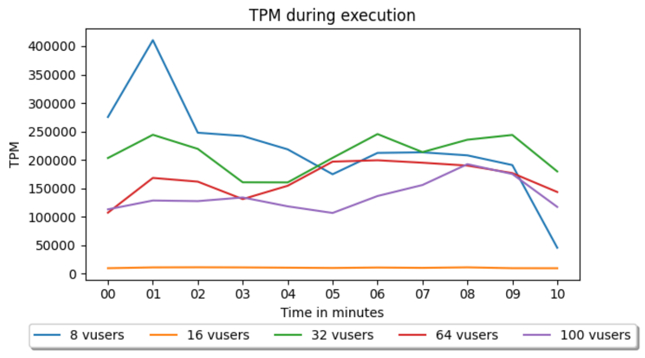


Figure 8: Machine 2 - 100 warehouses

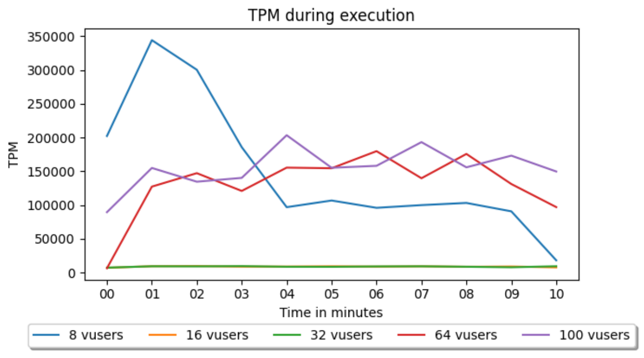


Figure 9: Machine 2 - 200 warehouses

As in the test 1.1, there was a significant reduction in the TPM for 16 virtual users, but this time this occurred for 32 virtual users as well. The highest peak and the average TPM were slightly reduced for 200 warehouses.

If we look at the table some sections before we can clearly notice that we went overboard by using 100 virtual users since the machine 2 doesn't actually have that much of computing power. But we wanted to leave here, just for the sake of it.

5.3 Test 1.3 - All warehouses true

In this test we set the 'all warehouses' option to true in the experiment. This will make a uniform load distribution across the warehouses instead of the normal workflow in which 90% of the load goes to the default warehouse attributed to a user.

5.3.1 Machine 1 - All warehouses false vs All warehouses true

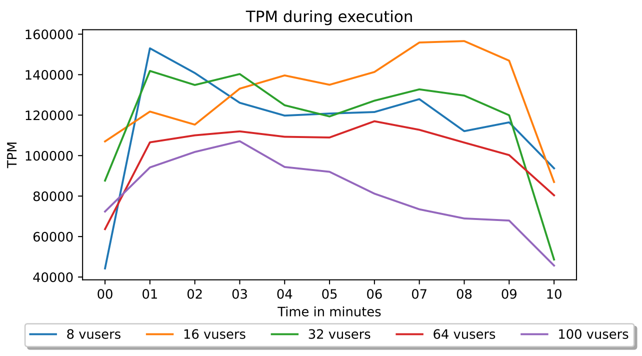


Figure 10: Machine 1 - All warehouses false

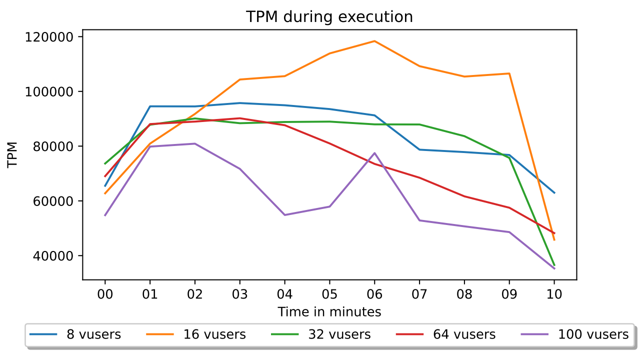


Figure 11: Machine 1 - All warehouses true

Setting the distribution to uniform reduced significantly the performance in all the tests.

5.3.2 Machine 2 - All warehouses false vs All warehouses true

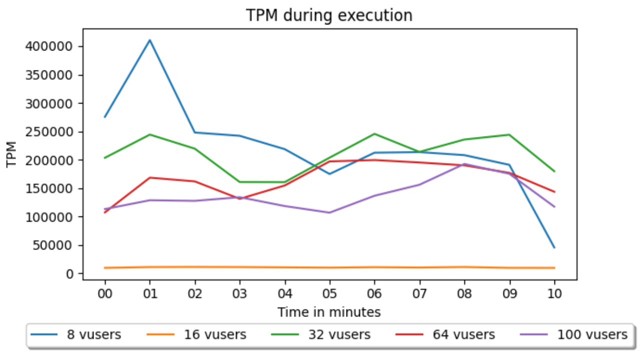


Figure 12: Machine 2 - All warehouses false

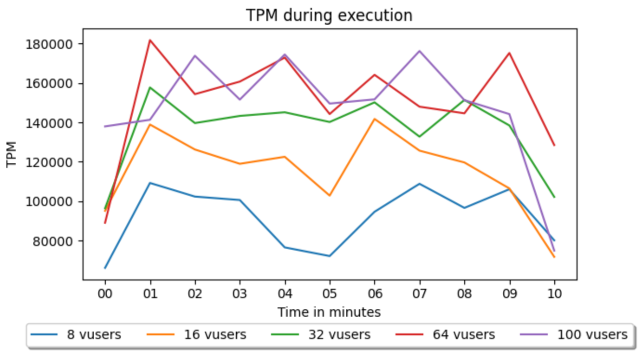


Figure 13: Machine 2 - All warehouses true

Machine 2 showed the same reduction in the TPM as Machine 1. Other consequence of setting the all warehouses variable to true is that the more users exist, the more transactions can be made.

5.4 Test 1.4

In this test, as in the test 1.3, we set the 'all warehouses' option to true. This time, the number of warehouses is double the default value for each machine.

5.4.1 Machine 1 - 80 warehouses & All warehouses false vs 80 warehouses & All warehouses true

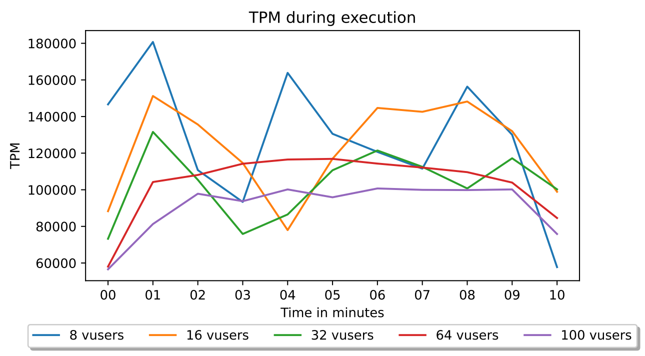


Figure 14: Machine 1 - 80 warehouses & All warehouses false

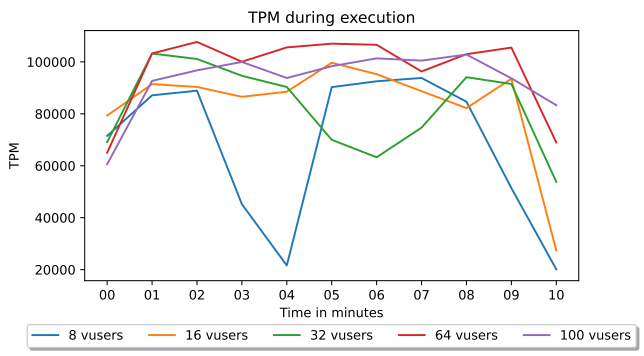


Figure 15: Machine 1 - 80 warehouses & All warehouses true

We can clearly see that there is a reduction of the highest peak of TPM when all warehouses is set to true. On the other hand, the average remains almost the same, showing just a small reduction. This is expected since, when doing this, the database will need to handle more concurrency conflict on the accesses to the tables. The roll backs potentically might be a bit more expensive.

5.4.2 Machine 2 - 200 warehouses & All warehouses false vs 200 warehouses All warehouses true

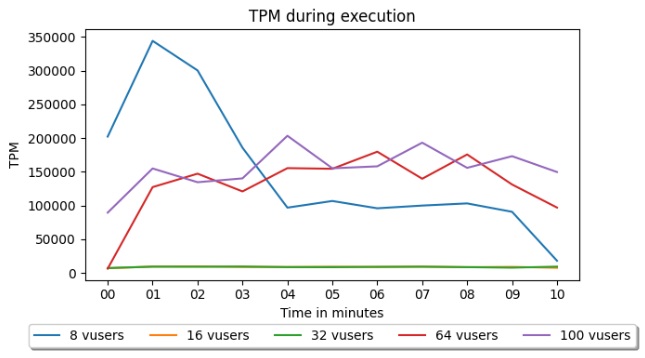


Figure 16: Machine 2 - 200 warehouses & All warehouses false

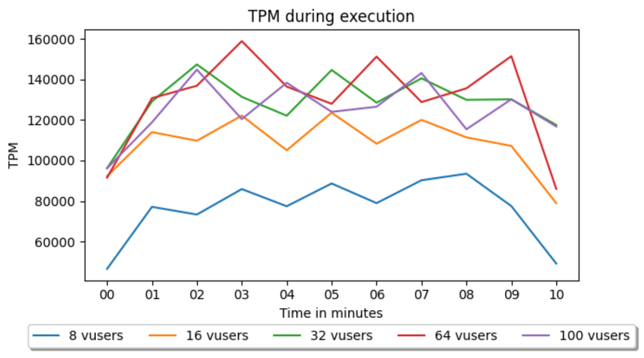


Figure 17: Machine 2 - 200 warehouses & All warehouses true

For this machine, the result is pretty much the same as for Machine 1: drop of the highest peak and almost consistent average in the TPM.

5.5 Test 2.1 & 2.2 - Increased buffer size

In this experiment we increased the shared buffer size from the default 128MB to 10GB and to 50GB and the work mem from 4MB to 1GB and to 5GB.

5.5.1 Machine 3 - Default configuration Vs Increased buffer size

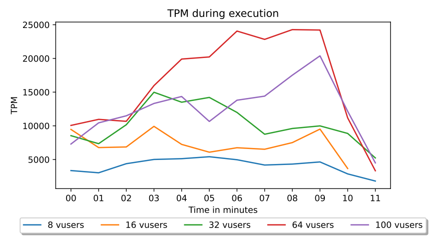


Figure 18: Machine 3 - Default configuration

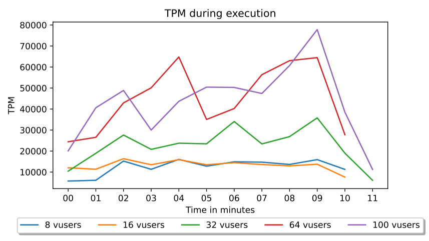


Figure 19: Machine 3 - Increased buffer (10GB)

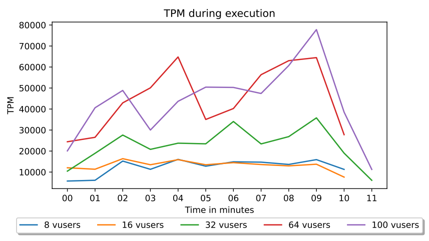


Figure 20: Machine 3 - Increased buffer (50GB)

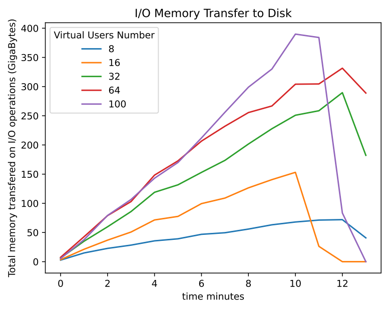


Figure 21: Machine 3 - Default configuration

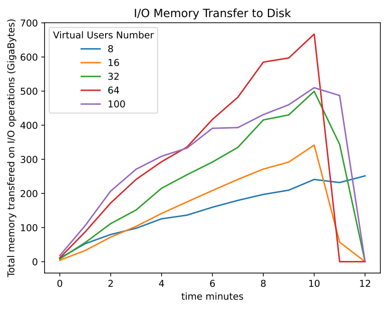


Figure 22: Machine 3 - Increased buffer (10GB)

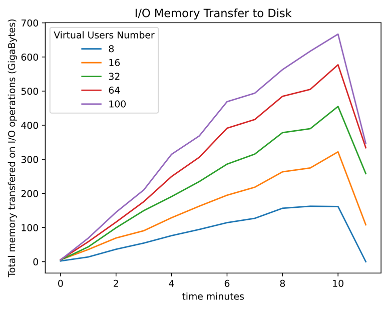


Figure 23: Machine 3 - Increased buffer (50GB)

This experiment clearly shows the difference of having a increased buffer size in our database when running a HDD as our storage. We could almost doubled the amount of transactions in every test case. Also when we look at the I/Os we notice two things: (1) is that they amount of memory transferred to disk grows faster, we believe is due to bigger buffered writes and so less I/O will be required and when performed they could pontentially be much bigger. We can also notice that (2) the amount of memory written to disk also grows. We didn't have the chance to investigate further on what postgres does that might have caused that, but we believe that in some way is taking advantage of bigger buffer to do more writes either by need to handle more resource or for some sort of performance reason.

5.5.2 Machine 4 - Default configuration Vs Increased buffer size

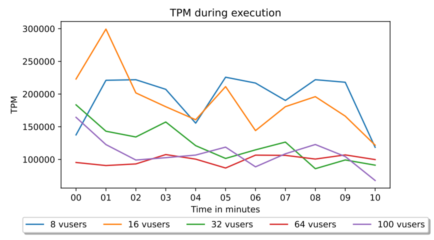


Figure 24: Machine 4 - Default configuration

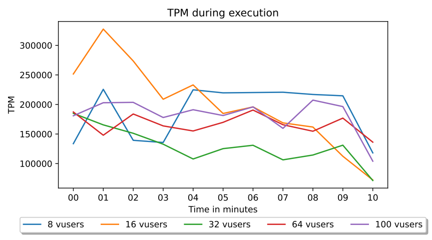


Figure 25: Machine 4 - Increased buffer (10GB

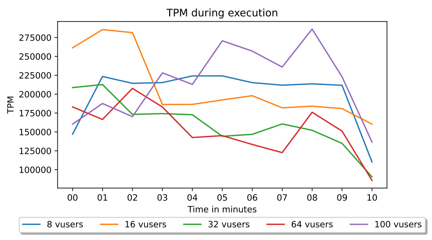


Figure 26: Machine 4 - Increased buffer (50GB

When running a larger buffer on a SSD we can only see a real improvement on the database performance when the number of users is larger our equal to the cores of our machine since that with the 8, 16 and 32 vusers the performance was more or less the same.

5.6 *Test 2.3 - Increased buffer size*

In this experiment we increased the shared buffer size from the default 128MB to 2GB and the work mem from 4MB to 500MB.

5.6.1 Machine 1

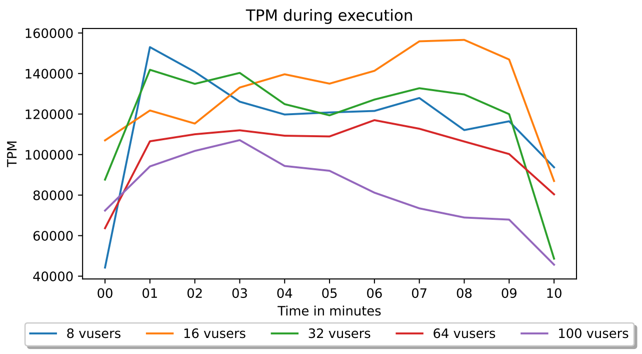


Figure 27: Machine 1 - Default configuration

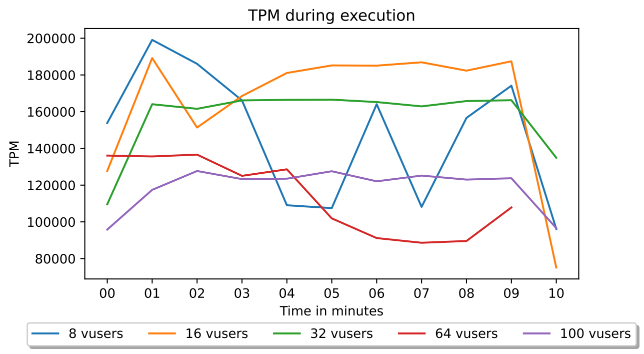


Figure 28: Machine 1 - Increased buffer (2GB)

Here we can see that the performance of the database become much more stable with the additional buffer size and could achive better results in terms of TPM.

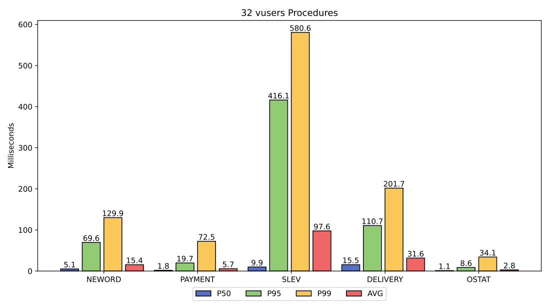


Figure 29: Machine 1 - Default configuration

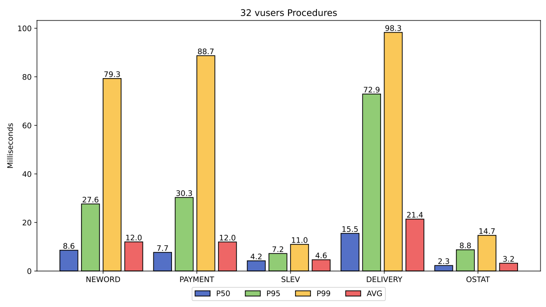


Figure 30: Machine 1 - Increased buffer (2GB)

Considering this two graphics representing the procedures for 32 virtual users, we can see that they are much faster when using additional buffer size, going from 580.6 ms for the slowest procedure of the first graphic against the 98.3 ms for the second graphic slowest procedure (6 times faster).

5.6.2 Machine 2

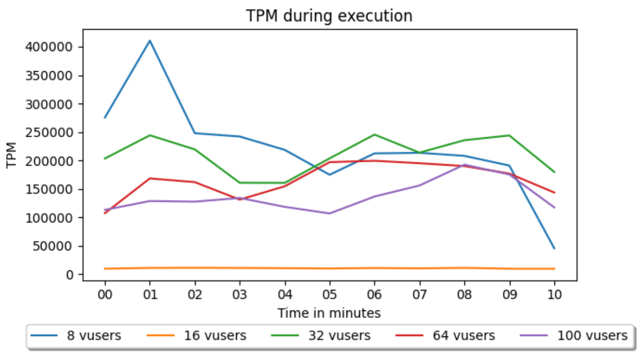


Figure 31: Machine 1 - test 1.4

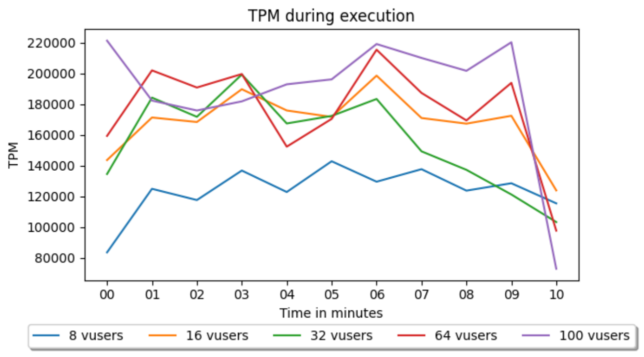


Figure 32: Machine 1 - test 1.4

Here we have some strange results regarding the worsening of the performance for 8 and 32 users. But we also have great improvements to when we have 64 and 100 users which makes sense since with more with a bigger buffer sizer we can achieve a better performance. The weaker performance of the 8 and 32 vusers tests could be due to some system unexpected behavior.

5.7 Test 3.1 e 3.2 Read commit Vs Repeatable read Vs Serializable

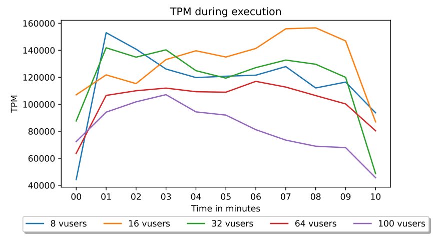


Figure 33: Machine 1 - Default configuration

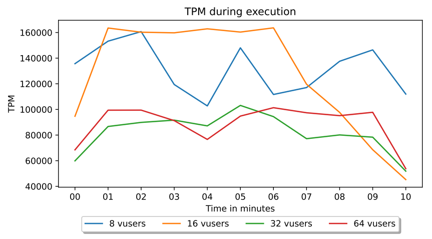


Figure 34: Machine 2 - Repeatable Read

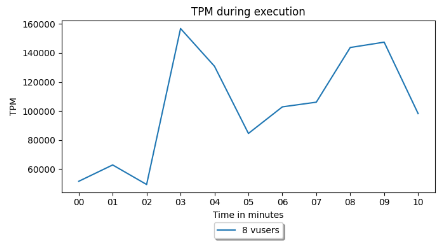


Figure 35: Machine 2 - Serializable

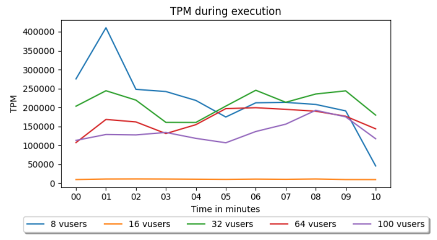


Figure 36: Machine 1 - Default configuration

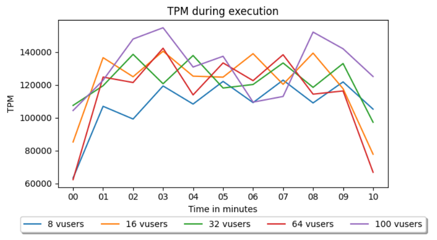


Figure 37: Machine 2 - Repeatable Read

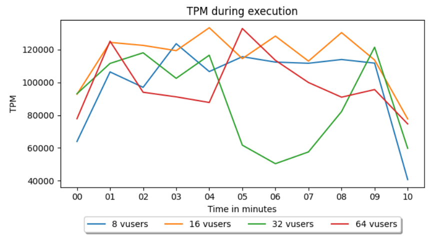


Figure 38: Machine 2 - Serializable

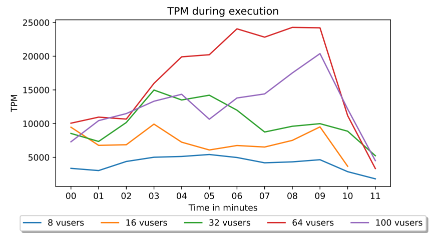


Figure 39: Machine 1 - Default configuration

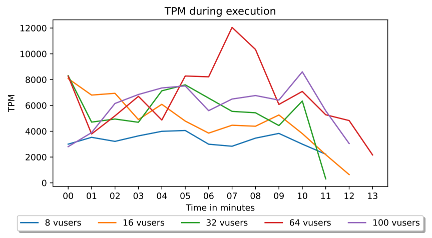


Figure 40: Machine 2 - Repeatable Read

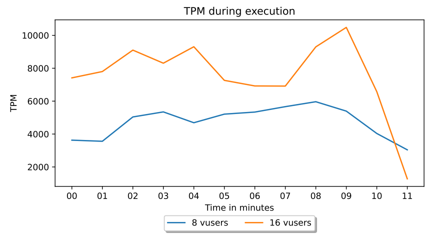


Figure 41: Machine 2 - Serializable

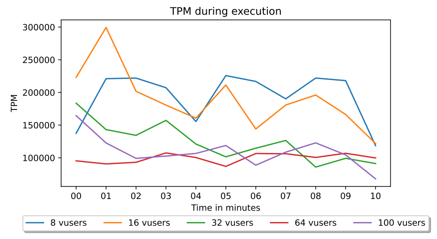


Figure 42: Machine 1 - Default configuration

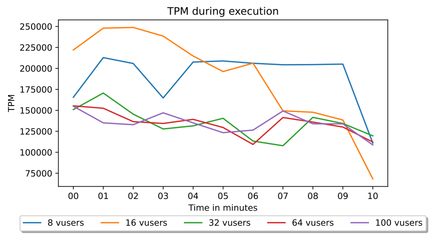


Figure 43: Machine 2 - Repeatable Read

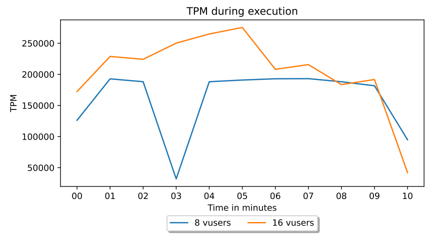


Figure 44: Machine 2 - Serializable

A very high level take of the results illustrated here is that: serializable is worst, the default is better and the repeatable read is in between. All of these is just what is expected. Serializable on itself is expensive and repeatable read makes rollbacks more expensive due the need to rollback several transactions and there is more book keeping on the database part to make sure that transactions only commit when others they depend due to reads also commit. Read committed is crazy simple when compared to these two so less overhead which leads to better performance.

5.8 Test 4.1 WAL

5.8.1 Machine 3 - Default config vs Increased WAL size

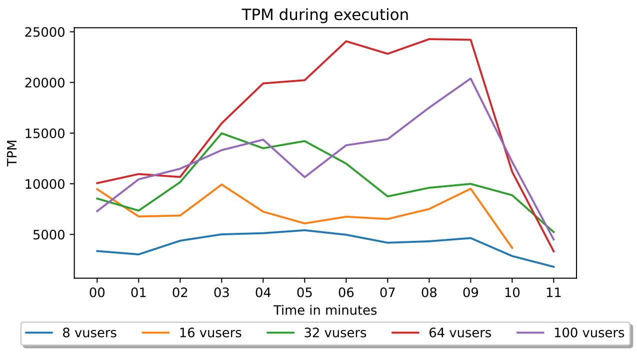


Figure 45: Machine 3 - Default config

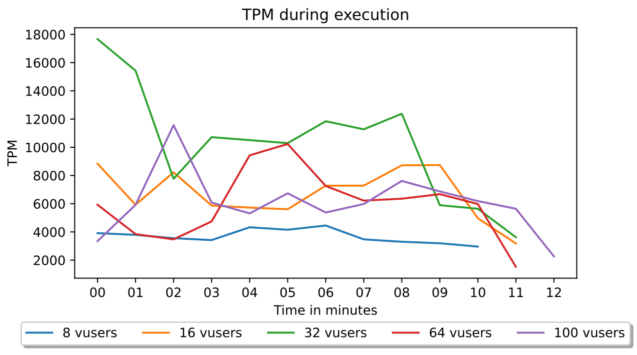


Figure 46: Machine 3 - Increased min and max WAL size

6 Conclusions

Our benchmarking project using HammerDB has provided valuable insights into the performance characteristics of PostgreSQL under different configurations and workloads.

The performance of SSD-equipped machines is significantly superior compared to machines with HDD storage. Even when the HDD machine is equipped with more memory and a better CPU, the storage type remains a critical factor for database performance. SSDs offer much faster read and write speeds, which is crucial for handling high transaction rates and large datasets efficiently.

In tests where we increased the number of warehouses to double the default value, we observed that the transaction rates remained relatively consistent regardless of the number of virtual users. This consistency is likely due to the even distribution of the load across all warehouses. With 90% of the requests targeting the same warehouse, the increased number of warehouses helps balance the load, leading to similar transaction rates across different user counts.

PostgreSQL shows good scalability with increased virtual users, as long as the underlying hardware (especially storage) can support the increased demand. The performance improvement from hardware upgrades like SSDs is evident, underscoring the importance of investing in high-quality storage for database performance.