NOVA

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

FCT NOVA

Databases systems

Project report

Project members:
Chatard Louis - 66865
Bogdan Nicoara - 66933
Sebastião Jerónimo - 60840
Łukasz Mirek - 65337

Project supervisor:
Carlos Augusto Isaac
Piló Viegas Damásio

Contents

1 Introduction

The field of database management systems plays a really important role in modern data-driven applications, powering the storage, retrieval, and manipulation of vast amounts of information. As organizations increasingly rely on data-driven decision-making processes, the need to evaluate and compare the performance and efficiency of different database systems is important. Benchmarking database systems provides valuable insights into their capabilities and assistance in identifying areas for optimization and improvement.
The objective of this project is to gain practical experience in benchmarking database systems using standard and open tools. To accomplish this, we will use HammerDB, a leading open-source benchmarking tool that supports various commercial and open source database management system.
Throughout this project, we will be performing database management system analysis of system behavior by increasing users, addition of new transactions or queries to the benchmarks, and comparison of multiple database systems within specific settings. By systematically investigating these aspects, we aim to uncover valuable insights into the strengths, weaknesses, and trade-offs associated with different database management system configurations.
This report presents an overview of our project, starting with an introduction to HammerDB, followed by a description of the problem to be addressed and a brief summary of the database management system that will be used for benchmarking. We will provide an in-depth description of the selected benchmark and outline the methodology employed to perform the benchmarking process, including machine configurations, database management system settings, experimental setup, and metrics used for evaluation. Subsequently, we will present the results of our tests, including the metrics employed and figures depicting the behavior of the systems under examination.
The report will conclude with a discussion of the results, highlighting key observations and insights gained from the benchmarking process, and will conclude with a summary of our findings. By undertaking this project, we aim to enhance our understanding of database system performance evaluation.

2 Overview of HammerDB

HammerDB is the leading open source tool for benchmarking relational databases, it has a GUI interface, a command line and a Web REST interface, it supports industry standard benchmarks, and it has high performance and scalability which is built into the design of the application. It is specifically designer to evaluate the performance of database systems, and it does that by simulating real-world workflows to which it measures performance and scalability. Some key features that it offers for testing are:

– Database Support: HammerDB supports a wide range of popular database systems, including Oracle Database, Microsoft SQL Server, MySQL, PostgreSQL This flexibility allows users to benchmark and compare the performance of different database platforms under similar conditions.

– Workload Simulation: HammerDB provides the ability to simulate a variety of workloads, configure and customize them and their characteristics, including the number of concurrent users, transaction types, data size, and distribution, to closely mimic real-world scenarios.

– Scalability Testing: HammerDB enables users to evaluate the scalability of their database systems by scaling the number of users and workload intensity. It helps identify potential bottlenecks and performance limitations as the system load increases, allowing users to optimize their infrastructure accordingly.

– Reporting and Analysis: HammerDB provides detailed reports and performance metrics, including transaction throughput, response times, and resource utilization. These insights enable users to analyze the database's behavior under different workloads and make informed decisions to optimize performance.

3 Addressed problem and used features

3.1 Addressed problem

HammerDB supports standard benchmarks both on the transactional side like TPROC-C which fall in the OLTP workload category, and TPROC-H which falls on the analytical side in the OLAP workload category. The OLTP workload category is used in context to applications that have many concurrent users with a lot of requests per second, it's important if the database can handle many queries at the same time usually when this is benchmarked some important numbers that are important to look are related to the performance like TPS (transactions per second) and the query latency. We were more interested in the analytical side of benchmarking, so that's why we used TPROC-H because of certain interests that we have, for example areas such data analysis were certain questions like "what is the 100 worst rated products on the company website in the winter?" these questions often read large parts of the database and do not require a lot of concurrent access to the database (that's why we did not change the parallelism variable on HammerDB), some useful metrics that we can get out of this benchmark are how long it took to run all the queries that are part of the benchmark and how long it took to run each of the queries, benchmarking in OLAP might take a more time than OLTP to complete, but the performance is still important nonetheless that's why we increased the number of queries and virtual users.

3.2 Features of PostgreSQL

PostgreSQL offers a wide range of features that make it a compelling choice for both small-scale applications and enterprise-level deployments. Some of the key features we will leverage in the benchmarking process include:

– Extensibility: PostgreSQL allows users to define custom data types, operators, and functions, enabling seamless integration with specialized data processing requirements. This flexibility allows us to tailor the database to the specific needs of the TPROC-H workload.

– Query Optimization: PostgreSQL employs a sophisticated query optimizer that analyzes queries and generates optimal execution plans. This feature plays a vital role in improving query performance, especially for complex analytical queries that are typical in data warehousing scenarios.

- Indexing and Performance Tuning: PostgreSQL offers various indexing techniques, including B-tree, Hash, and Generalized Inverted Index (GIN), to enhance query performance. Additionally, it provides advanced configuration options and tuning parameters to optimize resource utilization and concurrency control.
- Parallel Processing: PostgreSQL supports parallel query execution, allowing multiple CPU cores to be utilized efficiently for processing complex queries. This feature can significantly speed up the execution of analytical queries, thereby enhancing overall performance.
- Data Compression: PostgreSQL provides built-in compression mechanisms such as The Oversized-Attribute Storage Technique (TOAST), which automatically compresses large columns, reducing storage requirements and improving query performance for large datasets.

By utilizing these features, we will assess the performance of PostgreSQL when subjected to the TPROC-H benchmark workload. The TPROC-H benchmark simulates complex decision support queries involving large volumes of data. It consists of a set of predefined SQL queries representing various analytical operations typically performed in a data warehousing environment. The benchmark provides a standardized workload that enables fair and consistent comparison of database management systems performance across different machines. Through the benchmarking process, we aim to analyze the query execution times, resource utilization, and scalability of PostgreSQL in handling the TPROC-H workload. This analysis will provide insights into the strengths and limitations of PostgreSQL in supporting complex analytical queries and help identify potential areas for performance improvements.

4 Choice of Database management system

As the preferred database management system that HammerDB supports, we choose PostgreSQL as our used technology some members of the group already had experience with it, so it was a familiar option and besides that, not only is it very stable, but it has also been very competitive on the market in the past 20 years, being used in many web and mobile applications.

5 Methodology

The benchmarking process follows a well-defined methodology to ensure accurate and meaningful results. In this study, we outline the methodology employed for benchmarking PostgreSQL using the TPROC-H benchmark. The following components were considered: machine configurations, database management system configurations, experimental setup, and metrics used for performance evaluation.

5.1 Machines configuration

The benchmarking will be executed on different machine configurations, so we can see if some hardware parameters influence the performance of the database management system. The different machine configuration are the followings:

- configuration 1 (Louis): a system running Microsoft Windows 11 Family. The machine is a Lenovo Yoga Slim 7 Pro X, featured an AMD Ryzen 7 6800HS processor running at 3.20 GHz. It is equipped with 16 GB of RAM, but only 13.7 GB usable.
- configuration 2 (Łukasz): another laptop, featured with a 5th Gen Intel Core i5-5200U processor running at 2.20 GHz. It is equipped with 8.00 GB of RAM.
- configuration 3 (Sebastião): a system running Windows 11 Pro. The machine is a Lenovo ThinkPad E15, featured with a 5th Gen Intel Core i5-10210U processor running at 2.10 GHz. It is equipped with 8.00 GB of RAM, but only 7.78 GB usable.
- configuration 4 (Bogdan): a system running Microsoft Windows 10 Pro. The machine, a Lenovo 82FE, featured an 11th Gen Intel Core i5-1135G7 processor with 4 cores and 8 logical processors. It is equipped with 8 GB of RAM and had a total virtual memory of 14.6 GB.

These specifications should provide suitable environments for running the benchmarking tests.

5.2 Database management system configuration

The database management system used for our different tests is PostgreSQL. We set up the database using the HammerDB configuration and creating window in order to have the suitable parameters for TPROC-H benchmark regarding the memory allocation, concurrency control, and query optimization settings. We initiate the database with 15 users and scale value equal to 1.

5.3 Experimental setup

The project implies using two laptops linked one to another, with the database management system running on one and HammerDB performing the benchmark on in the other, both machines having just those software programs running. The configuration offers the best use and so performance of the machines as no other software is using the resources.
However, because we were running out of time to meet each other, we had to run both PostgreSQL and HammerDB on the same machine for the four of us.

5.4 Used metrics

HammerDB has plenty of metrics that can be used for benchmarks. Amongst them, we decide to focus only on the number of query per hours and the time used for running our tests. To get those metrics, we use the HammerDB functionality, which logs the measured number of query per hour with an associated timestamp.

5.5 Experimental protocol

We considered interesting, testing how different parameters variations react on our 4 machines over than testing different parameters on each of our laptops. As said earlier, the test is performed with a database initialized with 15 users and a scale parameter equal to 1.

The first parameters we decide to study is the number of virtual users using the database. To vary the number of virtual users allows evaluating system scalability, resource utilization, and performance under varying workloads. To test it we set 4 degrees of parallelism, 1 number of update sets, the number of total query sets per user equal to 5, and the user and repeat delay set to 500 ms. Without changing these parameters, we benchmarked the database with the following numbers of virtual users: 1, 5, 10, 20, and 50.
Then we focused on studying the influence of the total query sets per user, which can show the impact of query complexity and diversity on system performance To test it, we kept the previous values of degree of parallelism, number of update sets, the user and repeat delay. We also set the number of virtual users to 10 for these tests. Keeping those values the same, we benchmarked the database with the following total query sets per user values: 2, 5, 10, and 20.

6 Results of the testings

6.1 Obtainment of the results

To get the data for analyzing the queries per hour we add to activate the log options in the HammerDB transactions counter options (Fig. 1), after that we went to the directory to find the file where the output of the queries per hour logs will be. Then we find the hdbtcount file which looks like the figure 2, once all the virtual users are finished we can extract the data.

To extracted usable and viewable results of the log files created by HammerDB we had to process the data. Therefore, we write python scripts to first read the logs and save the relevant values in .csv files as required in the project instructions. Once those .csv files are created, we used another python program to create different plots and compute some statistics. For each laptop configuration, we plot the number of query per user regarding the elapsed time during the test on one graph with all the tested parameters, so we can compare the performance of each tested value to the others. For both studied parameters and for each machine configuration, we have also done a table with calculated statistics.

The influence of the number of virtual users, are visible in the annexes on figures 3 and 4. The influence of the total query sets per user in the annexes on the figures 5 and 6. The influence of the machine hardware configuration are compared on the figure 7. All the mentioned plots are visible and captioned in the annexes at the end of the document.

7 Discussion of the results

What we have observed in increasing the number of users but keeping the number of sets of queries to 5 per users is a substantial increase in time of the execution. One other interesting aspect that we have observed is that as the number of users increase, the number of queries per hour decrease and the execution takes longer to finish. One possible reason behind this process is that as we increase the number of virtual users, more of them remain stuck on hard, complicated queries, which limits the amount of total queries per hour as the degree of parallelism is limited.
Also, one key aspect that we have observed is that depending on which computer we run it, the number of the quarries per hour a system can manage greatly varies.

8 Conclusion of the project

8.1 Technical conclusion

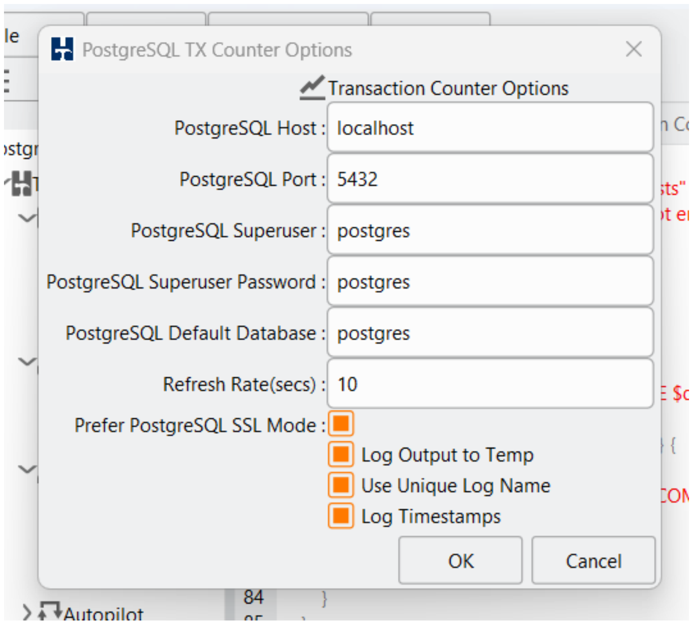
Based on this work, we learned how to use a benchmarking tool to test and observe our database behavior. In this paper, we analyzed what would mean to have a database respecting the TPROC-H benchmarking standard, meaning we have tried to see how a database would face multiple users doing hard analytical queries. We tested how the number of users influence the time and performance of the database, and also how much time each set takes depending on the numbers of queries. We conclude by saying that is a big factor that influences the system you run on, there is a linear increase in the time it takes to execute a set and the number of queries per set and also that the number of users increases exponentially the amount of time taken. One interesting observation we made is that as we increase the number of users, the amount of queries per hour decreases.

8.2 Limitations

Due to time constraints and lack of different machines, our benchmarking analysis was affected. Even though the project was delayed, we still struggled to deliver more quantity of benchmarking since we were having problems with versions of PostgreSQL, some pgAdmin versions specifically. We wanted to tune our database and play with other functionalities we couldn't, but we still gained some practical database benchmarking skills with HammerDB. Because of the short time, we also weren't able to compute the tests on all machines as we would like to.

9 Annexe - Figures

List of Figures

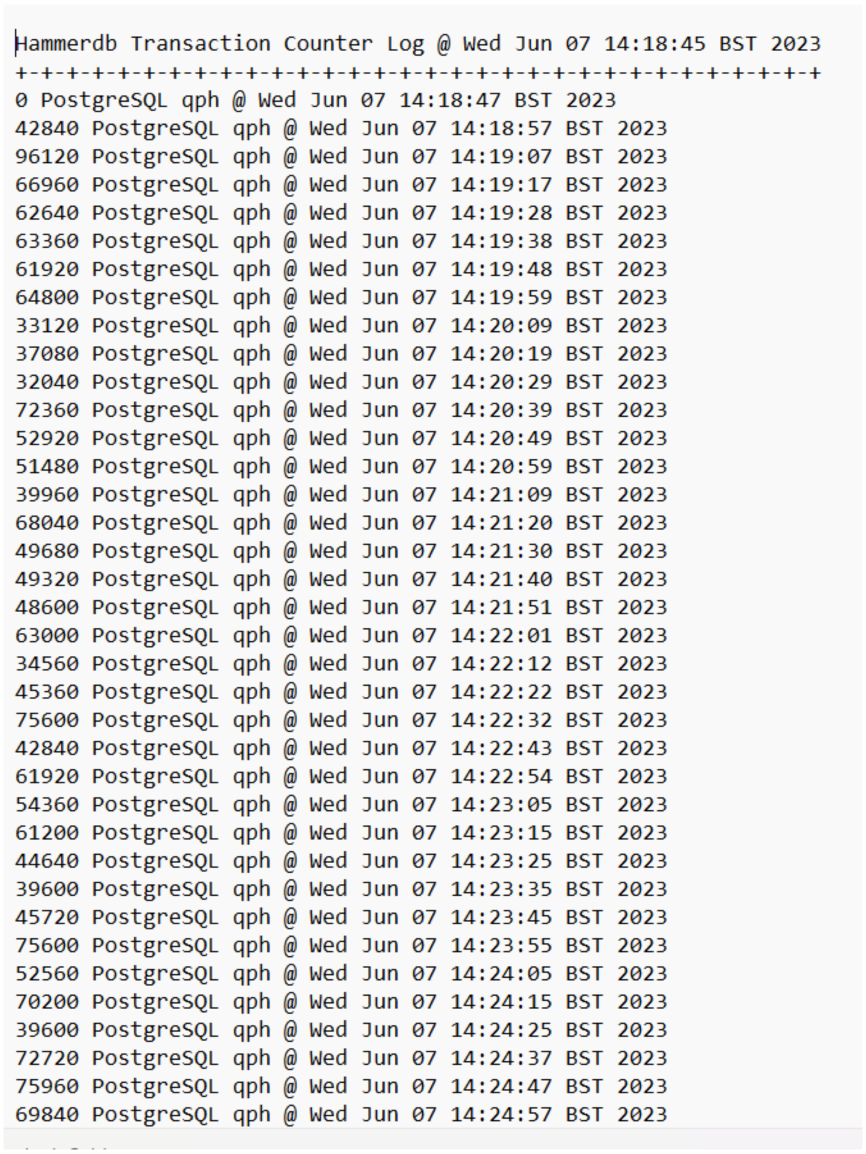


PostgreSQL TX Counter Options

Transaction Counter Options

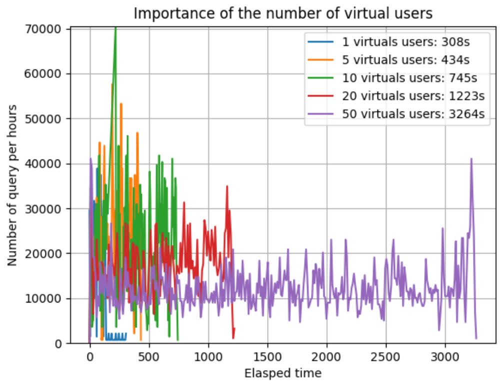
PostgreSQL Host :	localhost
PostgreSQL Port :	5432
PostgreSQL Superuser :	postgres
PostgreSQL Superuser Password :	postgres
PostgreSQL Default Database :	postgres
Refresh Rate(secs) :	10
Prefer PostgreSQL SSL Mode :	☐
	☐ Log Output to Temp
	☐ Use Unique Log Name
	☐ Log Timestamps

Figure 1: Transaction Counter options window

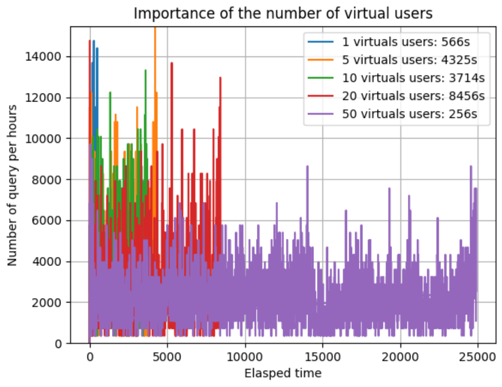


```
Hammerdb Transaction Counter Log @ Wed Jun 07 14:18:45 BST 2023
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
0 PostgreSQL qph @ Wed Jun 07 14:18:47 BST 2023
42840 PostgreSQL qph @ Wed Jun 07 14:18:57 BST 2023
96120 PostgreSQL qph @ Wed Jun 07 14:19:07 BST 2023
66960 PostgreSQL qph @ Wed Jun 07 14:19:17 BST 2023
62640 PostgreSQL qph @ Wed Jun 07 14:19:28 BST 2023
63360 PostgreSQL qph @ Wed Jun 07 14:19:38 BST 2023
61920 PostgreSQL qph @ Wed Jun 07 14:19:48 BST 2023
64800 PostgreSQL qph @ Wed Jun 07 14:19:59 BST 2023
33120 PostgreSQL qph @ Wed Jun 07 14:20:09 BST 2023
37080 PostgreSQL qph @ Wed Jun 07 14:20:19 BST 2023
32040 PostgreSQL qph @ Wed Jun 07 14:20:29 BST 2023
72360 PostgreSQL qph @ Wed Jun 07 14:20:39 BST 2023
52920 PostgreSQL qph @ Wed Jun 07 14:20:49 BST 2023
51480 PostgreSQL qph @ Wed Jun 07 14:20:59 BST 2023
39960 PostgreSQL qph @ Wed Jun 07 14:21:09 BST 2023
68040 PostgreSQL qph @ Wed Jun 07 14:21:20 BST 2023
49680 PostgreSQL qph @ Wed Jun 07 14:21:30 BST 2023
49320 PostgreSQL qph @ Wed Jun 07 14:21:40 BST 2023
48600 PostgreSQL qph @ Wed Jun 07 14:21:51 BST 2023
63000 PostgreSQL qph @ Wed Jun 07 14:22:01 BST 2023
34560 PostgreSQL qph @ Wed Jun 07 14:22:12 BST 2023
45360 PostgreSQL qph @ Wed Jun 07 14:22:22 BST 2023
75600 PostgreSQL qph @ Wed Jun 07 14:22:32 BST 2023
42840 PostgreSQL qph @ Wed Jun 07 14:22:43 BST 2023
61920 PostgreSQL qph @ Wed Jun 07 14:22:54 BST 2023
54360 PostgreSQL qph @ Wed Jun 07 14:23:05 BST 2023
61200 PostgreSQL qph @ Wed Jun 07 14:23:15 BST 2023
44640 PostgreSQL qph @ Wed Jun 07 14:23:25 BST 2023
39600 PostgreSQL qph @ Wed Jun 07 14:23:35 BST 2023
45720 PostgreSQL qph @ Wed Jun 07 14:23:45 BST 2023
75600 PostgreSQL qph @ Wed Jun 07 14:23:55 BST 2023
52560 PostgreSQL qph @ Wed Jun 07 14:24:05 BST 2023
70200 PostgreSQL qph @ Wed Jun 07 14:24:15 BST 2023
39600 PostgreSQL qph @ Wed Jun 07 14:24:25 BST 2023
72720 PostgreSQL qph @ Wed Jun 07 14:24:37 BST 2023
75960 PostgreSQL qph @ Wed Jun 07 14:24:47 BST 2023
69840 PostgreSQL qph @ Wed Jun 07 14:24:57 BST 2023
```

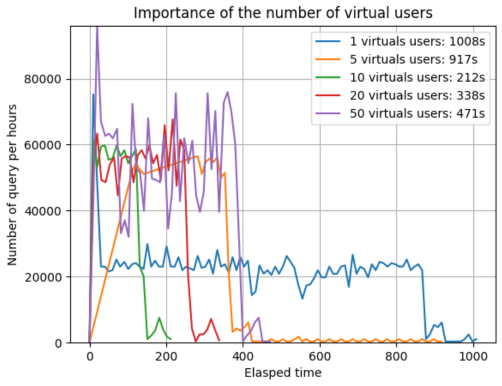
Figure 2: Log file example



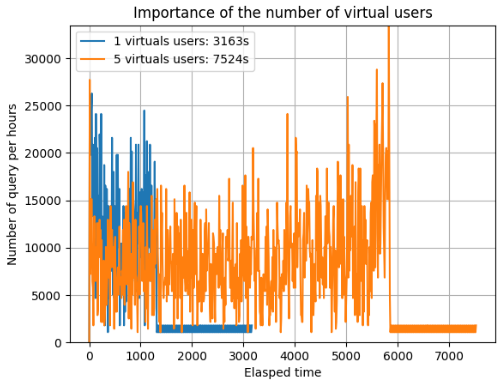
(a) Testing on configuration 1



(b) Testing on configuration 2



(c) Testing on configuration 3



(d) Testing on configuration 4

Figure 3: Plots of number of query per hour regarding time elapsed for different numbers of virtual users

	Mean	Median	Standard deviation	Variance	Time
1 virtual users	10207	1800	12498	156208778	308
5 virtual users	23238	23040	11912	141905244	434
10 virtual users	21466	20160	12009	144232692	745
20 virtual users	17362	16920	6411	41103093	1223
50 virtual users	12343	11160	5235	27407112	3264

(a) Statistics on configuration 1

	Mean	Median	Standard deviation	Variance	Time
1 virtual users	5621	6120	3891	15143548	566
5 virtual users	3872	3600	2594	6731565	4325
10 virtual users	4301	3960	2619	6862135	3714
20 virtual users	3062	2520	2101	4415663	8456
50 virtual users	2143	1800	1268	1608895	256

(b) Statistics on configuration 2

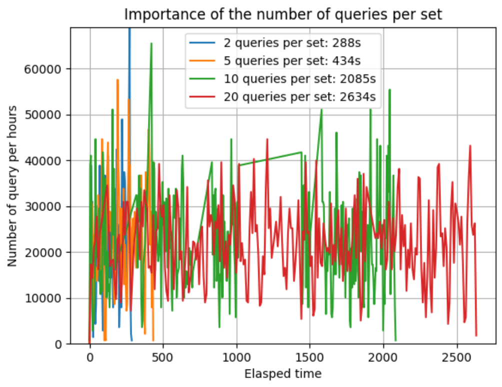
	Mean	Median	Standard deviation	Variance	Time
1 virtual users	20282	23040	10106	102135848	1008
5 virtual users	10346	360	20083	403355803	917
10 virtual users	34265	53820	26616	708463168	212
20 virtual users	40521	53100	24221	586665356	338
50 virtual users	45321	49320	25191	634635614	471

(c) Statistics on configuration 3

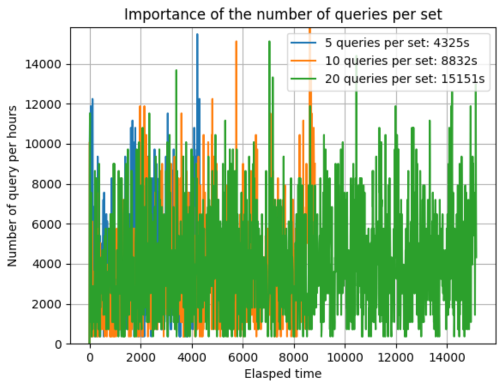
	Mean	Median	Standard deviation	Variance	Time
1 virtual users	5839	1800	6595	43502470	3163
5 virtual users	7374	6840	5473	29954683	7524

(d) Statistics on configuration 4

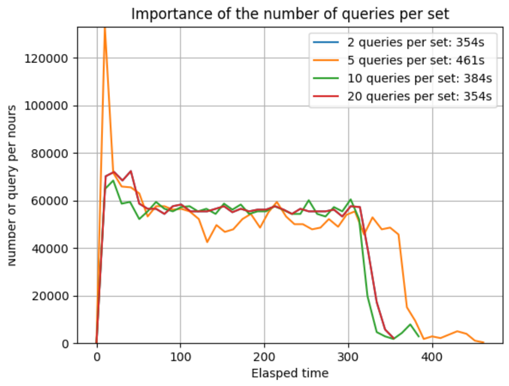
Figure 4: Statistics for each laptop configuration of the different numbers of virtual users



(a) Testing on configuration 1

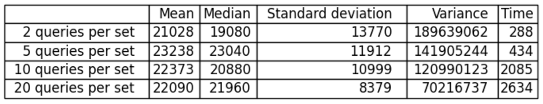


(b) Testing on configuration 2



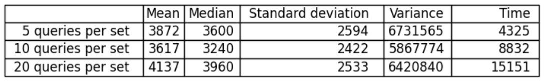
(c) Testing on configuration 3

Figure 5: Plots of number of query per hour regarding time elapsed for different numbers of total query sets per user



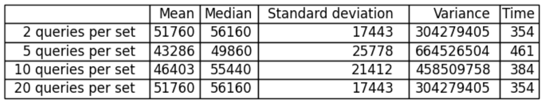
	Mean	Median	Standard deviation	Variance	Time
2 queries per set	21028	19080	13770	189639062	288
5 queries per set	23238	23040	11912	141905244	434
10 queries per set	22373	20880	10999	120990123	2085
20 queries per set	22090	21960	8379	70216737	2634

(a) Statistics on configuration 1



	Mean	Median	Standard deviation	Variance	Time
5 queries per set	3872	3600	2594	6731565	4325
10 queries per set	3617	3240	2422	5867774	8832
20 queries per set	4137	3960	2533	6420840	15151

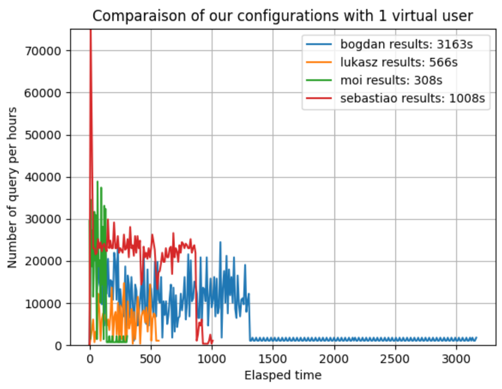
(b) Statistics on configuration 2



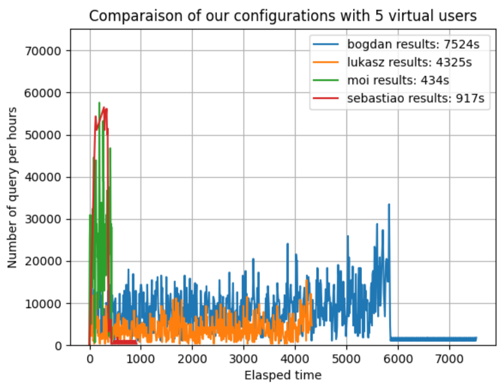
	Mean	Median	Standard deviation	Variance	Time
2 queries per set	51760	56160	17443	304279405	354
5 queries per set	43286	49860	25778	664526504	461
10 queries per set	46403	55440	21412	458509758	384
20 queries per set	51760	56160	17443	304279405	354

(c) Statistics on configuration 3

Figure 6: Statistics for each laptop configuration of the different numbers of total query sets per user



(a) Comparison for 1 virtual user



(b) Comparison for 5 virtual users

Figure 7: Comparison of the machines configuration