ULB UNIVERSITÉ LIBRE DE BRUXELLES

INFO-H-419: Data Warehouses

Report

MariaDB : TPC-DS Benchmark

Authors

Mustapha Ayadi

Soumaya Izmar

Narmina Mahmudova

Nils van Es Ostos

Professor

Esteban Zimányi

2023-2024

Contents

1 Benchmarking MariaDB **2**

1.1 MariaDB . . . 2

1.2 Decision Support System . . . 2

1.2.1 Types of Decision Support System . . . 3

1.2.2 Benchmarking in the development of a Decision Support System . . . 3

1.3 TPC-DS . . . 4

1.3.1 Query Summary . . . 4

1.3.2 Performance Test . . . 5

2 Implementation **6**

2.1 Hardware specification . . . 6

2.2 Scaling . . . 6

2.3 Generating and loading the data . . . 7

2.4 Generating queries . . . 7

2.4.1 Adapting queries to MariaDB . . . 7

2.5 Power test . . . 9

2.6 Throughput test . . . 9

3 Results **11**

3.1 Load test results . . . 11

3.2 Power test result . . . 12

3.2.1 Scale factor 1 . . . 14

3.2.2 Scale factor 3 . . . 15

3.2.3 Scale factor 5 . . . 16

3.2.4 Scale factor 7 . . . 17

3.3 Throughput test result . . . 18

3.4 Comparison between Power test and Throughput test . . . 18

3.5 Query optimization 19
3.5.1 Indexing 19
3.5.2 CTE Materialization 20

4 Conclusions **22**
4.1 Discussion 22

A Appendix **24**
A.1 Implementation of TPC-DS on mariaDB 24

Abstract

When it comes to implementing a data warehouse solution for an enterprise, there are many tools and services available. All these services claim to offer efficient data analysis solutions. Organizations need to discover simple and cost-effective ways of managing their growing data. Data warehouse benchmarking involves performing technical evaluations on data to measure performance, scalability and functionality. It's important to note that the results of these benchmarks depend largely on the specific business applications and analytical objectives chosen for decision support. The aim of this project is to evaluate the performance of a data warehouse system.

For this project, the relatively well-known database MariaDB will be studied. This database system is very similar to MySql. Indeed, in 2009, when MySql was acquired by Oracle, MySQL founder Michael Widenius and a few of his collaborators forked the project due to concerns about Oracle's management. He named the new project MariaDB after his second daughter, Maria. Evaluation of the performance of MariaDB queries using a well-known benchmark tool, TPC-DS. TPC-DS is a powerful metric that gives an overview of the performance of database management systems with different scaling factors.

1. Benchmarking MariaDB

This chapter is dedicated to the introduction of the technologies used for benchmarking.

1.1 MariaDB

MariaDB is an open-source relational database management system (RDBMS) that has become a popular choice for managing enterprise data. It is a fork of the MySQL project and has been developed with the goal of maintaining compatibility with MySQL while offering additional features and performance improvements.

Figure 1.1: Official MariaDB logo.

MariaDB stands out its adaptability and scalability, making it suitable for evolving, high-traffic applications. Despite this, MySQL remains widely more used by enterprises than MariaDB, due to its longevity in the sector.

1.2 Decision Support System

Decision Support Systems (DSS) are a category of information systems designed to support effective decision-making processes within organizations. These systems provide valuable tools and resources for collecting, analyzing, and interpreting data and information to help individuals and groups make informed decisions. One of the key components of a DSS is the Database Management System (DBMS), which stores, manages and retrieves data efficiently. This data can include historical records, current operational data, and external data sources. Notice that we will address this matter at a later stage.

Decision Support Systems helps organizations make well-informed data-driven decisions, reducing guess-working and leading to competitive edge in market. By automating data analysis,

DSS save time and resources in decision-making processes. In addition, DSS can assist in identifying and mitigating risks by analysis certain scenarios.

1.2.1 Types of Decision Support System

There are different types of Decision Support Systems tailored for various levels of decision-making within an organization:

1. Strategic DSS: These support top-level management in making strategic decisions that affect the overall direction of the organization. They often rely on external data and focus on long-term goals.

2. Tactical DSS: Tactical DSS assist middle managers in addressing medium-term challenges. They provide insights into how to implement strategic plans effectively.

3. Operational DSS: Operational DSS are for day-to-day decisions made by front-line employees and supervisors. They focus on routine tasks and immediate issues

1.2.2 Benchmarking in the development of a Decision Support System

Benchmarking is a valuable process in the development of a Decision Support System (DSS) as it helps to assess and improve the DSS's performance and effectiveness. Benchmarking in the context of DSS development involves comparing the DSS's performance and features against industry standards or best practices, as well as against other similar DSS solutions. It aims to identify areas for improvement and enhance the DSS's capabilities. Benefits of Benchmarking in DSS Development:

1. Performance Enhancement: Benchmarking helps identify areas for improvement, allowing developers to enhance the DSS's performance and capabilities.

2. Alignment with Best Practices: It ensures that the DSS adheres to industry best practices and standards, which can improve decision-making processes.

3. Cost Reduction: By optimizing the DSS, organizations can reduce operational costs and achieve a better return on investment.

4. Competitive Advantage: Benchmarking against competitors can help an organization gain a competitive edge by identifying and implementing superior DSS features and functions.

5. User Satisfaction: Focusing on user feedback and satisfaction leads to a DSS that better meets user needs and expectations.

1.3 TPC-DS

TPC-DS (Transaction Processing Performance Council Decision Support) [**?**] [**?**] is a widely recognized benchmark that measures the performance of decision support and data warehousing systems. It simulates real-world business scenarios and data processing workloads, helping organizations assess the efficiency of their database and data warehousing solutions. It was created to provide a standardized way of evaluating the performance of Database Management Systems (DBMS), helping organizations assess the ability of a system to handle complex queries, large datasets, and concurrent users, making it ideal for decision support and analytical workloads.

The benchmarking standard allows different scales factor, ranging from 1 G to 10^4 G. This enables organizations to test system performance under varying data loads. Furthermore, TPC-DS includes multiple query categories, from a wide range of aspects, which permits evaluating the system's performance in different scenarios.

In order to assess the performance of decision support and data warehousesing systems, TPC-DS reports metrics of throughput (queries per hour), query execution time and response times for concurrent user sessions among others. Later on, we will study aspects related to the loading of data, execution time of queries and the tool's ability to handle concurrent users.

1.3.1 Query Summary

TPC-DS includes a set of 99 queries that analyse the performance of DBMS in various scenarios. These queries cover a wide range of analytical and reporting tasks typically encountered in real-world business intelligence and data warehousing applications. Every query of the set belongs to one of the following groups:

1. Reporting queries: These queries aim to retrieve and present data from a data warehouse for reporting and analysis purposes. They are characterized for including aggregation, filtering, grouping and joining.

2. Ad-hoc queries: These queries are generated on-the-fly to answer immediate questions. Their key characteristics include user-driven, interactivity, unpredictable and response time.

3. Iterative OLAP queries: These queries are used for interactive data exploration. Usually characterized by multidimensional, slicing and dicing, pivoting and rolling up.

4. Extraction or data mining queries: These queries focus on extracting valuable insights, patterns, or knowledge from large datasets. They include scalability, data analysis, pattern discovery and algorithms.

1.3.2 Performance Test

The performance test consist in measuring the speed and efficiency with which a system can execute decision queries in a typical business scenario. We will perform different tests to get a complete picture on the subject:

1. Load Test: Defined as all activity required to bring the system to the configuration that immediately precedes the beginning of the Performance Test.

2. Power Test: Consist in running the 99 queries by a single emulated user.

3. Throughput Test: Consists of S_q query sessions each running all 99 queries at the same time. In our case, we will consider 4 query sessions.

2. Implementation

In this chapter, we will go through the various steps involved in the benchmark, as well as the difficulties we experienced.

2.1 Hardware specification

The first thing you have to think when you are about to benchmark any technology where are you going to run it. We come across two options, running TPC-DS workloads in a cloud environment or running them locally on your own infrastructure. As we want to have access to scalable and highly available resources, the best option would be running TPC-DS in the cloud, by means of utilizing cloud service providers like Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure. Unfortunately, this services often follow a pay-as-you-go model, where you pay for the resources you use. Therefore, since this is a mere approach to benchmark MariaDB, we have chosen to conduct a local TPC-DS test. The results of the benchmark have been conducted locally with the following specifications:

Operating System	Windows 11
CPU	Intel Core i7-12700KF
RAM	32 GB
Hard Disk	1 To PCIe NVMe M.2

Table 2.1: Hardware specification

2.2 Scaling

TPC-DS defines default scaling factors, which correspond to the size of the raw data generated for subsequent analysis. The default TPC-DS scale is *1TB,3TB,10TB,30TB,100TB*. Unfortunately, as you can see from the section on hardware specifications, it will be difficult for us to run the benchmark, mainly due to the computational resources this scaling would involve. Therefore, we

have decided to use a smaller scaling factors, *1GB, 3GB, 5GB, 7GB* . This will enable us to see if performance degrades when the scale is larger.

2.3 Generating and loading the data

The TPC-DS file contains a data generating program *dsdgen*. Initially, binaries were build attending at the following steps [1]:

1. Copy Makefile.suite to Makefile
2. Edit Makefile and find the line containing "OS = "
3. Read the comments and append your target OS. For example: "OS = LINUX"
4. Run "make"

After this first step, data was generated by just the command *dsdgen –scale x –dir /tmp* where */tmp* refers to the directory in which it was build and x being the amount of data generated.

In addition, since *dsdgen* generates 200-300GB/hour serially on a 2-3GHz x86 processor, it is useful to run multiple parallel streams when generating large amounts of data. In order to do that, the command *-parallel* was used. Finally, the data generated is loaded into the files by running the loader provided.

2.4 Generating queries

2.4.1 Adapting queries to MariaDB

The *dsqgen* utility is used to transform the query templates into executable SQL for certain "dialects". The program will generate 99 queries into a single file *query_0.sql* and we separated them using a script [2] to split them into 99 different files, named *query_n.sql* where $n \in \{1,...,99\}$. However, MariaDB is not one of the one supported, so we were forced to use the IBM Db2 SQL dialect and fix some problems by hand. In particular, we fixed the queries *2, 5, 14, 16, 18, 21, 22, 23, 27, 30, 36, 40, 41, 48, 49, 51, 67, 70, 77, 80, 86, 94, 95, 97*. Here are the specific changes made:

[1] Using Windows Subsystem for Linux (WSL), it allows you to install an Ubuntu terminal environment in Windows
[2] `https://github.com/Lorenc1o/TPC-DS_Oracle/blob/main/scripts/utils/separate_queries.py`

Adaptation 1: full outer join

MariaDB does not currently support *full outer join*. One solution is to use *Left join* keyword. Here is an example with a part of *query 97*.

Before

```
FROM  ssci FULL OUTER JOIN csci
ON (ssci.customer_sk = csci.customer_sk
AND ssci.item_sk = csci.item_sk)
```

After

```
FROM (
(SELECT DISTINCT customer_sk, item_sk FROM ssci)
AS s LEFT JOIN
(SELECT DISTINCT customer_sk, item_sk FROM csci)
AS c ON s.customer_sk = c.customer_sk
AND s.item_sk = c.item_sk)
```

Adaptation 2: RollUp

At the syntax level, MariaDB does differently for the *rollUp* function. The keyword *WITH ROLLUP* instead of *ROLLUP* is used.

Before

```
GROUP BY ROLLUP (channel, id)
ORDER BY channel,id
FETCH FIRST 100 ROWS ONLY;
```

After

```
GROUP BY channel,id WITH ROLLUP) t1
ORDER BY channel,id
LIMIT 100;
```

Adaptation 3: Rank

The *Rank* function has been replace by *dense_rank* function. Example with a part of query 49.

Before

```
SELECT item,return_ratio,currency_ratio,
RANK() OVER
(ORDER BY return_ratio) AS return_rank,
RANK() OVER
(ORDER BY currency_ratio) AS currency_rank
```

After

```
SELECT item,return_ratio,currency_ratio,
DENSE_RANK() OVER
(ORDER BY return_ratio) AS return_rank,
DENSE_RANK() OVER
(ORDER BY currency_ratio) AS currency_rank
```

Adaptation 4: Grouping

MariaDB does not support *grouping* function which allow to substitute meaningful labels for super-aggregate *NULL* values instead of displaying it directly. One way of replicating this behavior is to use the *coalesce* function from MariaDB. Here is a example with query 70.

Before

```
SELECT SUM(ss_net_profit) AS total_sum,
s_state,s_county,GROUPING(s_state)+
GROUPING(s_county) AS lochierarchy
```

After

```
SELECT SUM(ss_net_profit) AS total_sum,
s_state,s_county,COALESCE(s_state)
+ COALESCE(s_county) AS lochierarchy,
```

N.B : *For some queries, we have also fixed some variable names and some other syntax variations.*

2.5 Power test

The power test implementation is available in our git repository. Here is a simplified view of how we implemented our power test.

```
# Connect to mariaDB
connection = mysql.connector.connect( host=db_host, user=db_user, database=db_name, port=
    db_port)
for query in queries: # iterate on all queries generated
    cursor = connection.cursor() # Generate a cursor in order to interact with the DB
    start = get_time()
    cursor.execute(query) # execute the query into the DB
    end = get_time()
    output end - start # the query execution time
```

Note that ere we are using the library *MySQL Connector*[3] in order to connect to our mariaDB database.
In this way we obtain the execution time for each query.

2.6 Throughput test

The throughput test implementation is also available in our git repository. Here is a simplified view of how we implemented our throughput test.

```
def run_queries(queries):
    # Connect to mariaDB given the parameters
    connection = mysql.connector.connect(
        host=db_host, user=db_user, database=db_name, port=db_port
    )
    for query in queries:
        cursor = connection.cursor()
        cursor.execute(query)
        cursor.close()

# Here we are going to simulate 4 clients
# that connect and performs the 99 queries in parallel
nb_client = 4
queries = [f for f in os.listdir(datadir) if f.endswith(".sql")] # the 99 queries
start = get_time()

with ThreadPoolExecutor(max_workers=nb_client) as executor: # Create 4 clients
    random.shuffle(queries)  # Shuffle the order queries for each client
    executor.submit(run_queries, queries)

duration = get_time() - start
```

[3] https://dev.mysql.com/doc/connector-python/en/

Each client will execute a set of 99 queries (in any order). The throughput test time is the difference between the timestamp taken before the first query of the first client and the timestamp taken after the last query of the last client.

3. Results

This section concerns results and discussions for load test, power test and throughput test.

3.1 Load test results

We can see in left-side figure 3.1 that for the load test at each scale, the loading time is multiply by a factor 2 except for scale factor 7, where this is slightly larger. This means that loading times increase more than linearly which implies that MariaDB does not scale optimally. One thing to consider is that the number of raw data in each scale is much greater than the scale. For example for scale 1, 4.63 GB are actually loaded. The right-side figure 3.1 shows the execution time for each of the estimated full size.

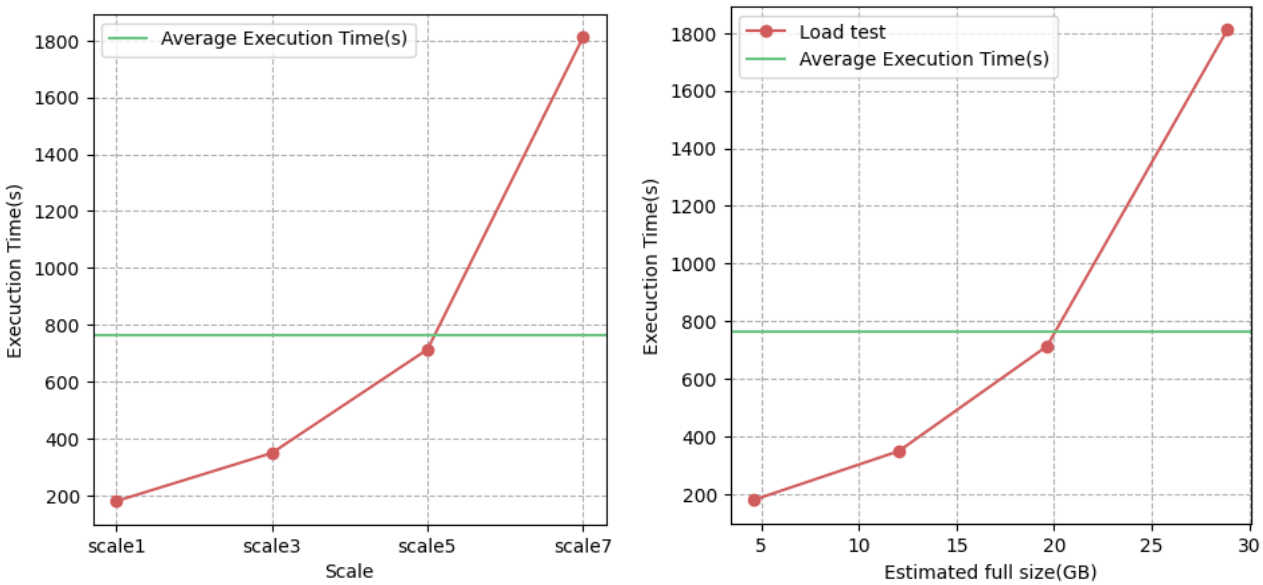

Figure 3.1

3.2 Power test result

For the power test, we can see in figure 3.2 that for the first three scales, it's linear. However, after scale 5, we see that the execution time is multiplied by a factor of 3. This may raise questions about the scalability of MariaDB for the power test. Unfortunately, to confirm that, we should benchmark other scale factor. Furthermore, we are now going to take a look at the execution time for each query and each scale, in order to determine the reason for the difference between the first three scales and the last.

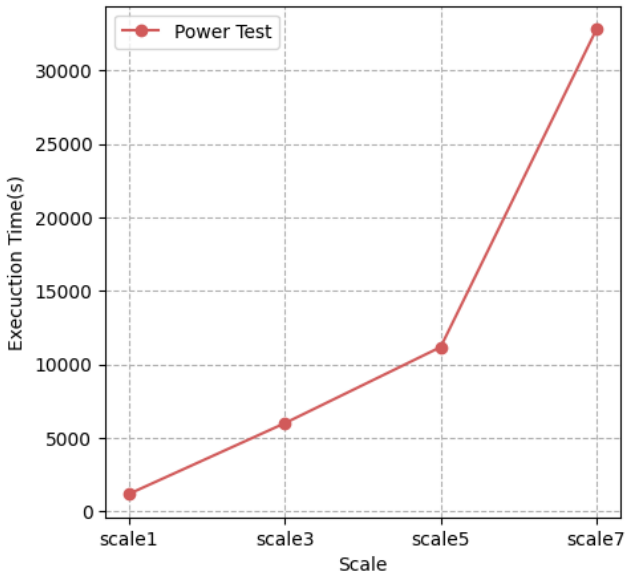

Figure 3.2: Power test total execution time in second for each scale.

Let's have a look at the graphics obtained for the different scales par queries. For display reasons (very large graphics), the graphics will be separated from the text to which they correspond.

Scale 1 We can see in figure 3.3 that there are 6 queries above the average. Especially queries 5 and 14, which take 132 and 146 seconds respectively. These queries are approximately three times higher than the average.

Scale 3 In figure 3.4, we can see that the execution time of query 72 explodes between scale 1 and 3, going from 18 seconds to 2112 seconds! We can also see that queries 5 and 14 also have execution times that have quadrupled. However, when it comes to the total execution time for all 99 queries, query 72 plays a key role.

Scale 5 In figure 3.5,we can see that scale 5 has a similar shape to scale 3, and query 72 continues to have an awful execution time.

Scale 7 In figure 3.6, we can see that query 72 behaves even worse than expected. In fact, we've gone from an execution time of 3576 seconds at scale 5 to 8013 seconds at scale 7! Furthermore, queries 47 and 64 have seen their execution times increase by more than a factor of 10.

We can see that query 72 is the main reason for the big difference between scale 5 and 7. Other queries have their roles and that's why we have decided to optimize them in section 3.5, in order to get better results and understand why MariaDB has so much trouble with these queries.

3.2.1 Scale factor 1

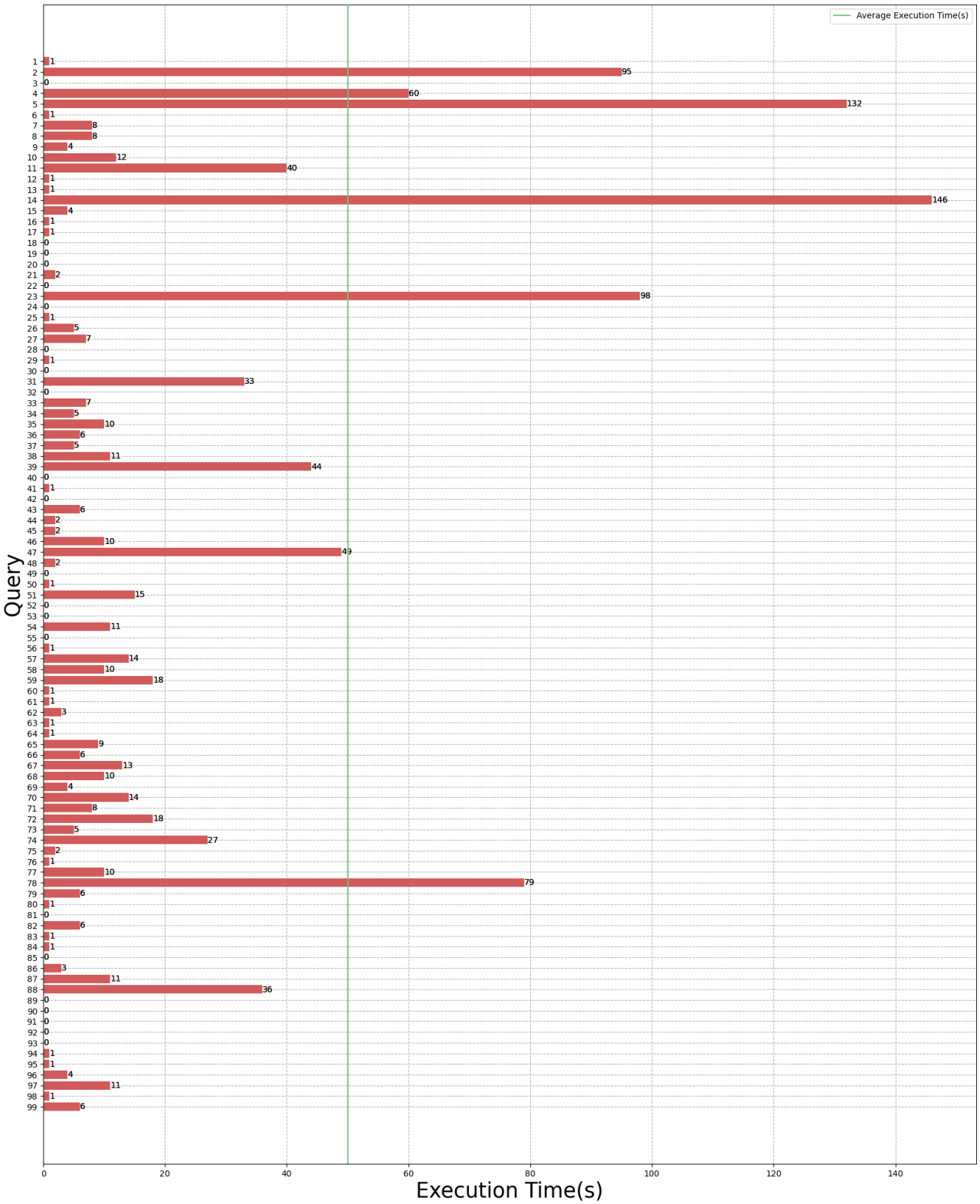

Figure 3.3: Execution time in seconds for scale 1.

3.2.2 Scale factor 3

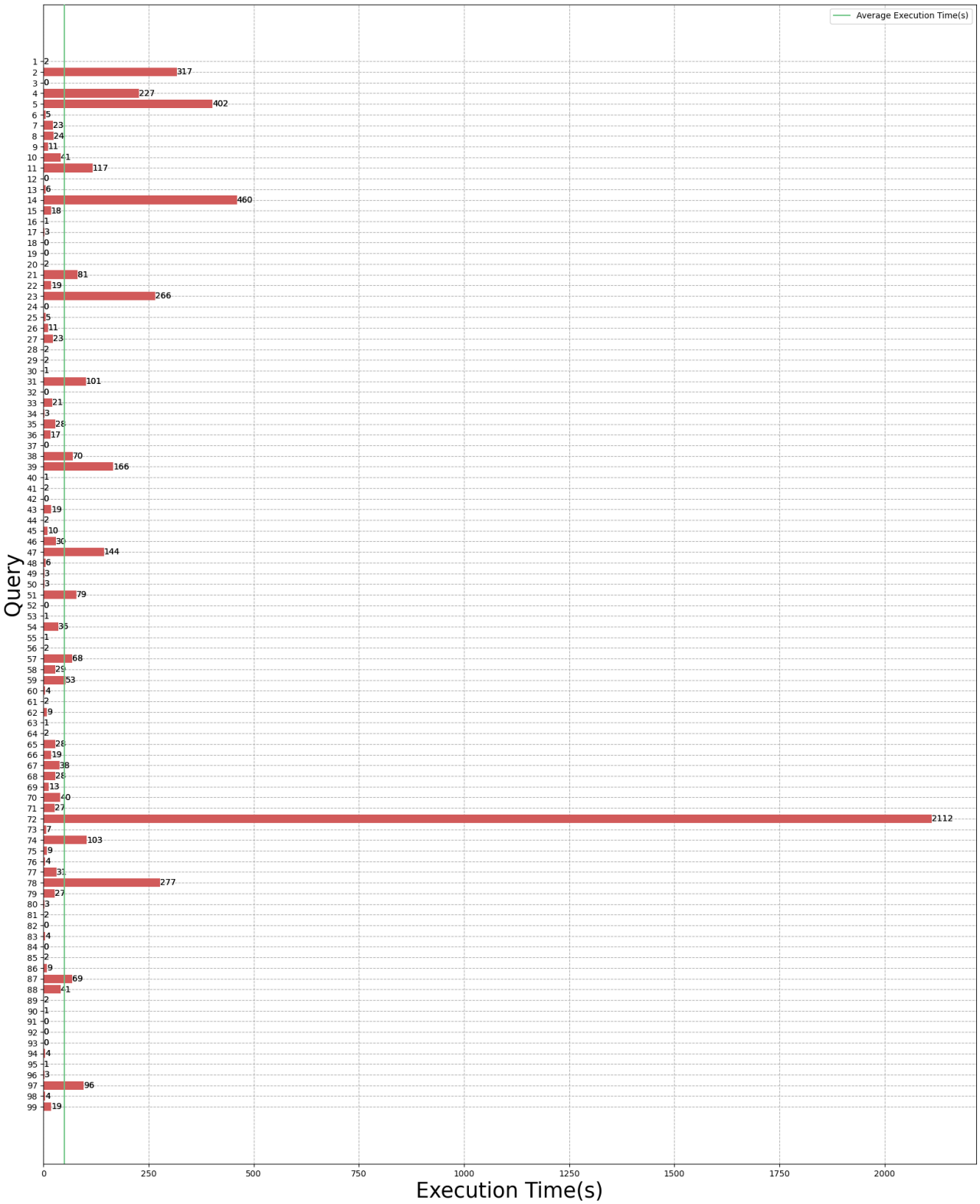

Figure 3.4: Execution time in seconds for scale 3.

3.2.3 Scale factor 5

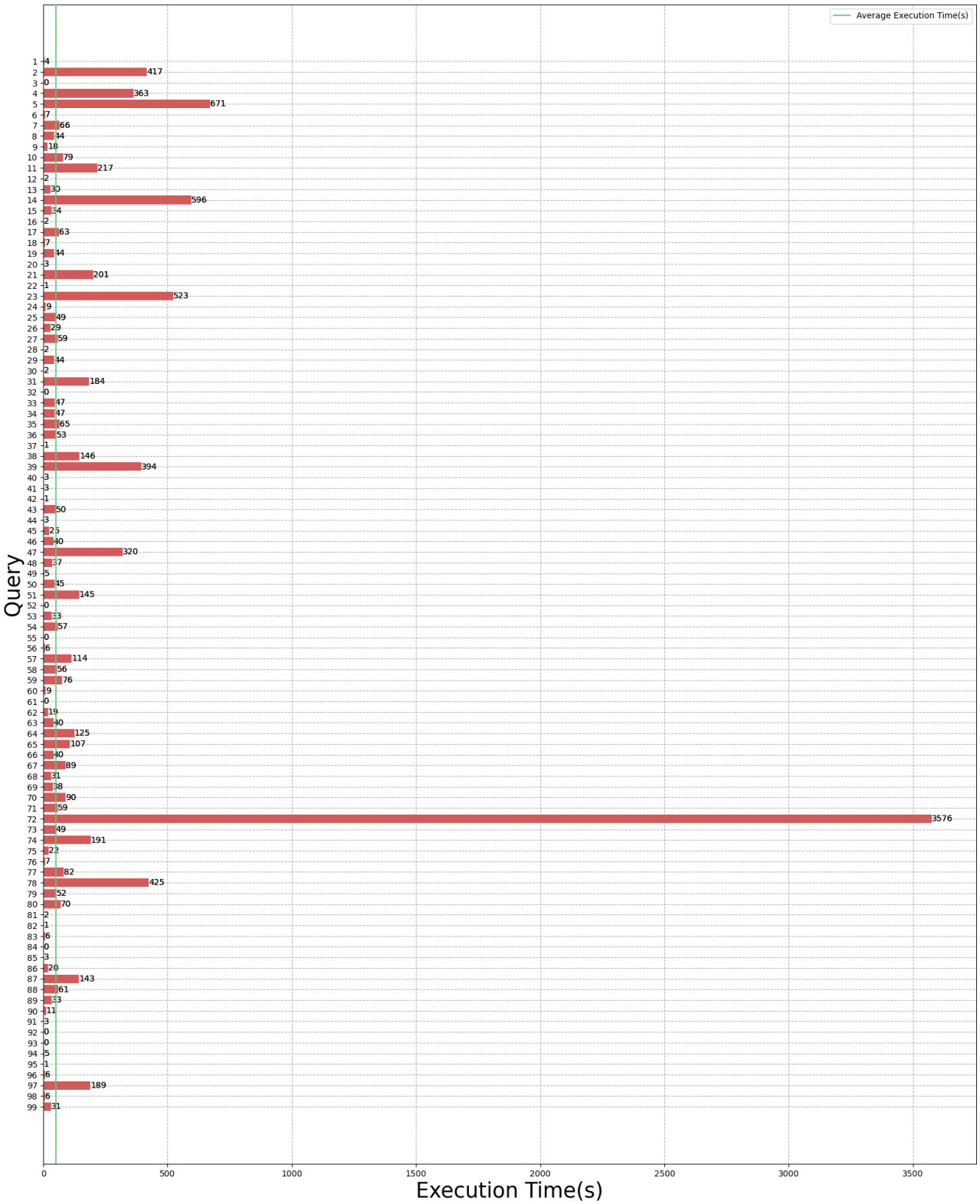

Figure 3.5: Execution time in seconds for scale 5.

3.2.4 Scale factor 7

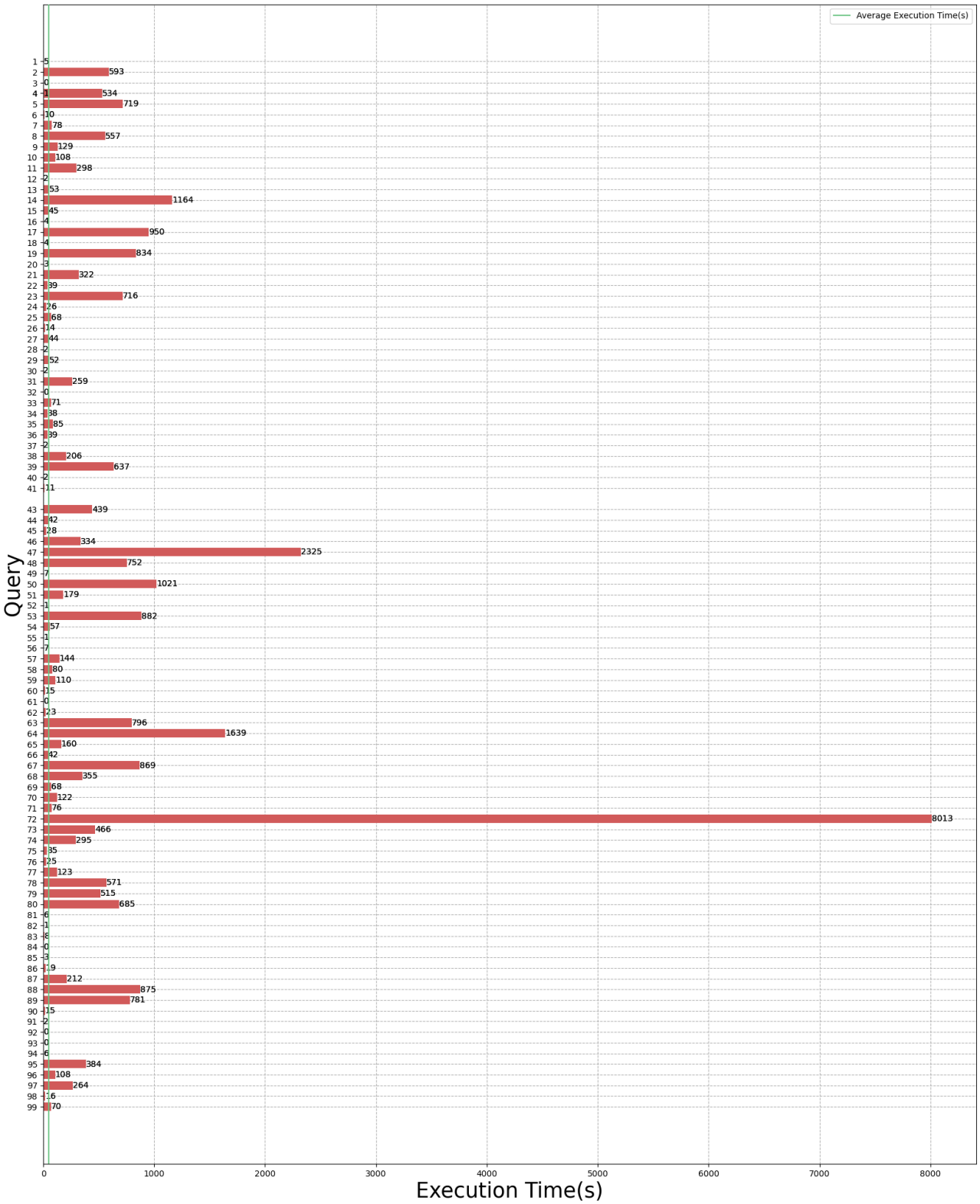

Figure 3.6: Execution time in seconds for scale 7.

3.3 Throughput test result

We can see that the trend of the function is exponential. We might think that MariaDB is not scalable in multi-user scenarios, given that we only have 4 clients for this test. However, we will discuss the scalability of the throughput test compared with that of the power test in the next section 3.4.

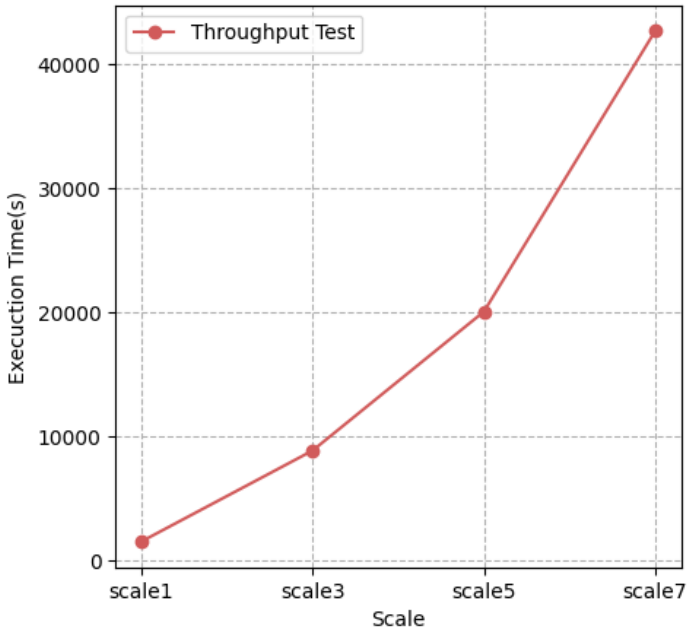

Figure 3.7: Throughput test total execution time in second for each scale.

3.4 Comparison between Power test and Throughput test

Let us overlap Fig.(3.2) and Fig.(3.7), and show the result in Fig.(3.8). The difference between the Power Test results and the Throughput Test results is minimal, which is interesting considering that the Power Test only consider one concurrent query session while the Throughput Test consider seven. This results show that MariaDB is indeed highly scalable, which is a very important aspect of a software technology as it keeps the database from crumbling under increasing traffic.

Another interesting observation in both the Power and Throughput tests is the exponential increase in execution time when the amount of data in the database is increased. This feedback suggests to us that MariaDB may not be a suitable tool for handling massive databases. As a curious fact, we can see that the exponential factor of the throughput test is smaller that the one of the power test. Assuming a non-erratic conduct, a linear behavior should be found from a specific value. However, due to a lack of time, this property has not been studied, so let it be noted for

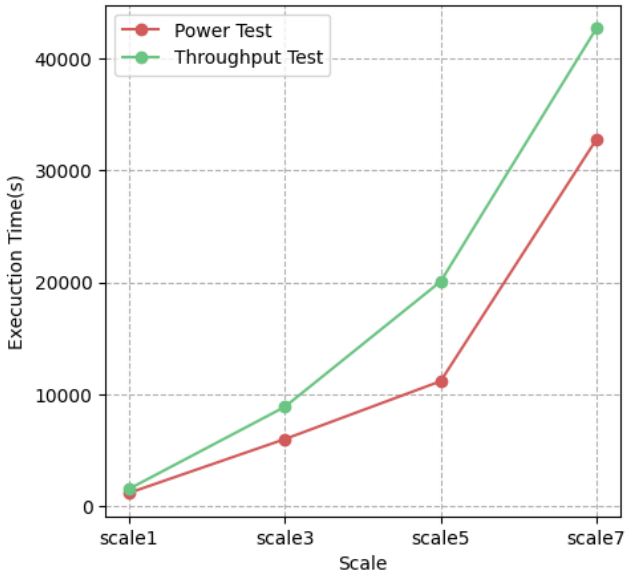

Figure 3.8: Comparison between the total execution time for the power test and the throughput test.

possible future work.

3.5 Query optimization

We can see how there are multiple queries that take very long to execute. This happens due to the way MariaDB manages clauses like JOIN, HAVING or ROLLUP among others. In our case, we decided to optimize *query 72* due to its massive execution time compared to the rest of the queries in scale factor 3, 5 and 7, and *queries 2* and *5* because they are the two queries that take the most in scale factor 1 and they similar execution time. The approach to optimize these queries is presented below.

3.5.1 Indexing

Indexing consists in creating a data structure, the so-called index, that provides a more efficient way to retrieve and access data in a table. We will use as an example of indexing, the ones used for *query 72*:

```
CREATE INDEX idx_week_seq ON date_dim(d_week_seq);
CREATE INDEX idx_cs_sold_date_sk ON catalog_sales(cs_sold_date_sk);
CREATE INDEX idx_cs_item_sk ON catalog_sales(cs_item_sk);
CREATE INDEX idx_inv_item_warehouse ON inventory(inv_item_sk, inv_warehouse_sk);
```

Indexing is a powerful when approaching query optimization. In fact, we can see in Fig.(3.9) the improvement of *query 5* and *query 72*, respectively. Notice that while the improvement in *query*

5 is good enough, the one in *query 72* is massive. The reason for this is that *query 72* makes use of less variables in clauses that *query 5* which enables us to create less indexes and improve more the performance.

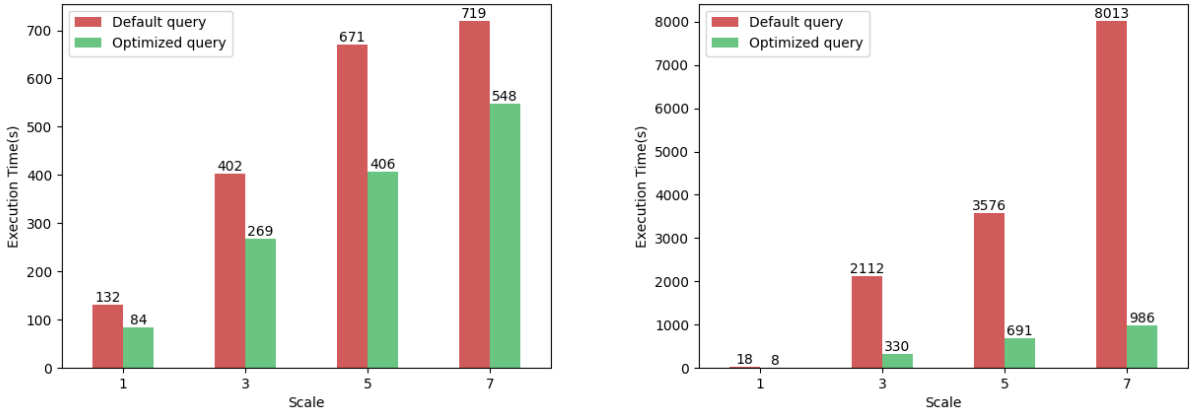

Figure 3.9: Comparison between the running time before and after optimizing for (a) query 5 and (b) query 72.

3.5.2 CTE Materialization

Common Table Expression (CTE) Materialization is usually used in SQL queries to create temporary result sets for complex subqueries or recursive queries, making from it an important aspect of query optimization. In our specific case, we have used CTE materialization to optimize *query_14.sql*. The added CTE's look like the following:

```
WITH store_sales_bucket AS (SELECT *
       FROM store_sales JOIN date_dim d1 ON ss_sold_date_sk = d1.d_date_sk
       WHERE d1.d_year BETWEEN 1999 AND 1999 + 2),
catalog_sales_bucket AS (SELECT *
       FROM catalog_sales JOIN date_dim d2 ON cs_sold_date_sk = d2.d_date_sk
         WHERE d2.d_year between 1999 AND 1999 + 2 ),
web_sales_bucket AS (SELECT *
       FROM web_sales JOIN date_dim d3 ON ws_sold_date_sk = d3.d_date_sk
         WHERE d3.d_year BETWEEN 1999 AND 1999 + 2)
```

For this query, we created 3 'buckets' that contain information about the *store_sales*, *catalog_sales* and *web_sales*, respectively. In the query, this data is multiple times used. Therefore, by creating this CTE, all these calculations will only be done once, improving its performance.

In Fig.(3.10), the improvement in the execution of this query is shown. As we can see, although the difference is not as big as in *query 72*, is still around a 30% quicker for scale factor 1. Notice that for scale factor 3, the optimized query takes even longer than the original one. However, as

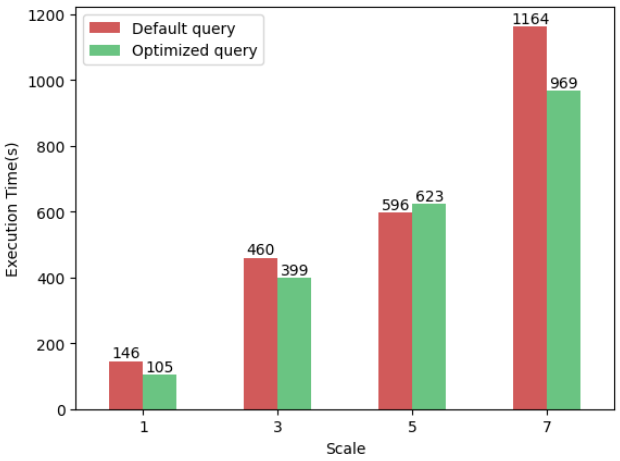

Figure 3.10: Comparison between the running time before and after optimizing for query 14.

we have mentioned before, the reason behind this optimization is improving its performance for scale factor 1.

4. Conclusions

4.1 Discussion

In conclusion, the article presents a comprehensive benchmarking study of MariaDB, an open-source relational database management system. The study covers various aspects, including the load test, power test, and throughput test, conducted at different scaling factors. Key findings from the benchmarking study are as follows:

1. Load Test: Loading times were observed to increase more than linearly, suggesting that MariaDB may not scale optimally for data loading.

2. Power Test: While execution times were linear for the first three scales, they significantly increased for larger scales, raising concerns about MariaDB's scalability in terms of workloads.

3. Throughput Test: The comparison between power and throughput tests revealed minimal differences, indicating MariaDB's high scalability concerning multi-users, which is essential for handling increasing traffic efficiently.

4. Database Size Impact: Both tests showed an exponential increase in execution time as the database size grew, suggesting limitations in handling massive databases.

5. Query Optimization: Query optimization techniques, such as indexing and CTE materialization, were explored to improve the performance of specific queries, resulting in significant execution time reductions.

The observed exponential behavior in execution time as the database size grows raises considerations about the tool's suitability for managing massive datasets. However, it becomes evident that MariaDB excels when employed for smaller databases with a substantial number of concurrent users, making it an attractive option for such scenarios.

Regarding MariaDB's suitability as a database management tool, it proves to be highly capable for specific use cases, particularly where scalability and query optimization are of paramount importance.

Furthermore, the TPC-DS benchmark tool demonstrates its effectiveness in evaluating the perfor-

mance of database management systems, making it a valuable tool for assessing a system's ability to handle complex queries, large datasets, and concurrent users, thus aiding in informed decision-making.

A. Appendix

A.1 Implementation of TPC-DS on mariaDB

The implementation of the different scripts that has been used for the realisation of this report can be found in Github. Please follow the steps provided in the *README* file if you want to reproduce what has been done in this report.

Bibliography

[1] Transaction Processing Performance Council (TPC). TPC BENCHMARK DS - Standard Specification. 3.2.0. https://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp

[2] Menzler, Julian. "A Summary of TPC-DS." Medium, Hyrise, 19 June 2019. URL: https://medium.com/hyrise/a-summary-of-tpc-ds-9fb5e7339a35

[3] Vaisman, Alejandro, and Esteban Zimányi. Data Warehouse Systems: Design and Implementation. Springer, 2022.

[4] Yahya Bakkali. "Ybakkali/Info-H419-Project." GitHub, October, 2021. URL: https://github.com/ybakkali/INFO-H419-Project/blob/main/Part%201/report/Report.pdf

[5] Jose Antonio Lorencio Abril. "Lorenc1o/TPC-DS ORACLE: Repository for Development of the First Project of the Course Data Warehouses (Info-H-419) from the Program BDMA, Fall 2022." Github, November, 2022. URL: https://github.com/Lorenc1o/TPC-DS_Oracle/blob/main/TPC-DS_Oracle_Report.pdf