Data Warehouses

INFO-H419

Part 1 : TPC-DS benchmark on Impala

BAKKALI Yahya (000445166)
FALLAHI Amirmohammad (000460073)
HAUWAERT Maxime (000461714)
LIBERT Alexandre (000435755)

October 2021

Université Libre de Bruxelles (ULB)

Contents

1 Introduction

Databases are inseparable parts of programming and today's technologies. As the quantity of data are increasing constantly, using an efficient database is the main key to overcome the problem of analyzing the data and getting answers in reasonable amounts of time.

Since there exist a lot of different databases, choosing one which is efficient in answering the queries related to a specific domain is a hard task. Therefore, database **benchmarks** can be used in order to compare efficiency and performance of different databases. A database benchmark is a repeatable experimentation on performance of database in various fields such as memory occupation or execution time.

TPC is one of the corporation which focused on developing databases benchmark standards. One of the benchmarks that TPC provided is **TPC-DS** benchmark. TPC-DS is a decision support benchmark that simulates numerous components of a decision support system, such as queries and data maintenance.

In this paper the details of the TPC-DS experiment performed on **Impala** will be seen. Apache Impala is the open source, native analytic database for Apache Hadoop.

2 Hardware specification

In order to do the benchmark experimentation, the *Dell xps-15-9560* computer has been used. The following table show the hardware specification of this computer.

Operating system	Ubuntu 20.04.3 LTS
System type	64 bits
CPU	Intel®Core™i7-7700HQ
CPU frequency	2.8 GHz up to 3.8 GHz
RAM capacity	16 GB
RAM type	DDR4
RAM speed	2,400 MT/s
Hard disk	PM981 NVMe Samsung
Hard disk capacity	256 GB

Table 1: Device information

3 Software specification

In this section, details about Impala, its installation and its functioning will be seen. In order to facilitate reproduction of the experimentation, some scripts have been written, which will be explained in the upcoming sections. It is also necessary to mention that for this experimentation Impala **version 4.0.0** has been used.

3.1 Apache Impala

Apache Impala is an open-source SQL query engine which uses massively parallel computing [1] for data stored in an Apache Hadoop distribution. It provides SQL queries with great speed and minimal latency. When working with medium-sized datasets is required, and real-time responses are necessary, Impala is the ideal solution.

3.2 Impala's prerequisites

In order to work with impala, knowing the basics of Apache Hadoop, HDFS (Hadoop Distributed File System) commands and basic knowledge of SQL is required. HDFS is the Hadoop significant component, in charge of storing enormous datasets of structured or unstructured data across several nodes and as result of that, maintaining metadata in log files format. The following table shows some HDFS commands, their description, and their *UNIX* equivalence (if it exists.)

HDFS command	Description	UNIX equivalent
ls	List all files	ls
lsr	Same as ls but with recursive perspective	ls -R
mkdir	Create a directory	mkdir
touchz	Create an empty file	touch
cat	Print file contents	cat
copyFromLocal (put)	Copy files or directories from OS to hdfs store	∅
copyToLocal (get)	Copy files or directories from hdfs store to OS	∅
moveFromLocal	Move file from OS to hdfs	∅
cp	Copy files	cp
mv	Move files	mv
rmr	Delete a file from HDFS recursively	rm -r
du	Give the size of each file in directory	ls -s

Table 2: HDFS basic commands

3.3 Installing Impala

Impala can be installed by using the source code of their official GitHub repository[2]. Once the repository had been cloned, the following instructions can be used to install Impala on the system[3].

```
$ git clone https://gitbox.apache.org/repos/asf/impala.git ~/Impala
$ cd ~/Impala
$ export IMPALA_HOME=‘pwd‘
$ ./bin/bootstrap_system.sh
$ source ./bin/impala-config.sh
$ ./buildall.sh -noclean -notests -format -start_minicluster -start_impala_cluster
```

The last step took at least one hour to complete.

[1] Massively Parallel computing is a way of computing in which numerous computers (CPUs) work simultaneously in parallel to execute a group of computations

[2] https://github.com/apache/impala

[3] https://cwiki.apache.org/confluence/plugins/servlet/mobile?contentId=65146982#content/view/65146982

This script first initializes all the environments variables, then it starts following services:

- Apache Hadoop : consisting of a storage part; HDFS (Hadoop Distributed File System), a processing part and the MapReduce. Hadoop divides files into large blocks and distributes them on cluster nodes.
- Apache YARN (Yet Another Resource Negotiator) : which is the Hadoop cluster manager.
- Apache Kudu : is the Hadoop ecosystem columnar data-store optimized to use on memory-intensive hardware.

Finally, it starts an Impala Cluster running with 3 nodes (3 coordinators, 3 executors).

3.4 Launching Impala

Once the impala build has been successfully performed, the impala cluster will be started and connected automatically. However, after each reboot of the computer, the following commands must be executed to start the cluster and establish the connection with impala. Before launching it, some important variables must be set. The script **impala-config.sh** can be used for this aim.

```
$ cd ~/Impala
$ source bin/impala-config.sh
$ testdata/bin/run-all.sh
$ bin/start-impala-cluster.py
```

3.5 Interacting with Impala

There exists a lot of ways to interact with Impala. The easiest method, which was used in this project, is to simply use the **Impala-Shell** program. The shell can be launched by using this command.

```
$ bin/impala-shell.sh
```

The **Impala-Shell** program can also execute a query directly given in the command.

```
$ bin/impala-shell.sh -f <SQL file> -d <database name>
```

3.6 TPC

TPC is a corporation which establishes and develops objective transaction processing and databases benchmarks. In this part of the project, TPC-DS is used to verify performance and efficiency of Impala. TPC-DS benchmark illustrates decision support systems that :

- Analyze enormous amounts of data

- Provide solutions to real-world business problems
- Execute queries with a variety of operational needs and complexity
- Are marked by a high CPU and IO load.
- Are regularly with source OLTP databases by using maintenance functions.
- Launch on **Big Data** solutions, like RDBMS and also Hadoop/Spark based systems.

4 Implementation

In this section, details of implementation and experimentation will be seen. This section shows how the input of the experimentation has been provided and also explains how the situation of the experimentation has been facilitated for users.

4.1 TPC-DS kit

In order to launch the experimentation, TCP-DS tool was downloaded from TCP official website[4].

4.1.1 Data generation

The TPC-DS kit provides a data population generator to generate experimental data for different scaling factors. The command line executed for this purpose is as follows:

```
$ ./dsdgen -scale <volume of datae> -dir <output destination> -parallel 4 -child 1 &
$ ./dsdgen -scale <volume of data> -dir <output destination> -parallel 4 -child 2 &
$ ./dsdgen -scale <volume of data> -dir <output destination> -parallel 4 -child 3 &
$ ./dsdgen -scale <volume of data> -dir <output destination> -parallel 4 -child 4
```

4.1.2 Queries generation

To perform the benchmark and obtain the results of the experimentation, the TPC-DS model queries have been adapted to obtain executable `.sql` queries on Impala. The TPC-DS tool has dialects to convert the model queries into SQL queries, the "netezza" dialect is the closest to Impala, however the dialects given with the tools do not work until the following line is added at the end of these files "define _END = "";". The following bash codes show how to generate the 99 queries.

```
$ for i in {1..99};
> do
> ./dsqgen -template query$i.tpl -directory <query_templates directory of TPC-DS tool> -output
    <output destination> -dialect netezza -scale <volume of data>;
```

[4] http://tpc.org/tpc_documents_current_versions/current_specifications5.asp

```
> mv <output destination/query_0.sql> <output destination/query_$i.sql>;
> done
```

4.2 Scripts

The following scripts have been implemented in order to make the experimentation simplier.

4.2.1 create_tables.sql

This script was provided by TPC corporation in order to initialize the experimentation by creating table. As the experimentation have been performed on Impala, this script has been changed a little bit in order to be compatible with Impala; It is a simple script which creates tables with the aim of collecting the experimentation data.

4.2.2 execute_queries.sh

As there are 99 queries, it would be time-consuming to execute the queries one by one; So, a script has been implemented in order to automate the execution of TPC queries and starts the examination. For each execution, its output will be stored in a distinct file and its execution time will be appended to a file which records execution time.

```
$ ./execute_queries.sh <query_path_older> <answer_path_folder>
```

4.2.3 load_data.sh

This script loads data in the tables that we created by `create_tables.sh`; It uses `moveFromLocal` command of *HDFS* in order to move files from local path (OS) to HDFS (Hadoop Distributed File System). The following box shows the code that have been executed for each table :

```
$ hdfs dfs -moveFromLocal <data_source>/<table_name>.dat <data_destination>/<table_name>;
```

4.3 Queries adaptation

As Impala is different a little bit from original *SQL*, some of the queries have been updated in order to be compatible with impala. These adaptions are as follows:

4.3.1 Type A: Date and INTERVAL keyword

In this type of adaptation, the date syntax errors have been handled. The time syntax in the template queries is not valid for Impala. It was therefore necessary to use vendor specific syntax for date expressions and date casting in some cases.

Before

```
where date_sk = d_date_sk and d_date
    between cast('[SALES_DATE]' as date)
    and (cast('[SALES_DATE]' as date) +
    14 days) and store_sk = s_store_sk
```

After

```
where date_sk = d_date_sk and d_date
    between cast('[SALES_DATE]' as date)
    and (cast('[SALES_DATE]' as date) +
    INTERVAL 14 days) and store_sk =
    s_store_sk
```

4.3.2 Type B: ORDER BY instruction followed by CASE WHEN keyword

The aliases after keyword "CASE WHEN" which has been implemented in an "ORDER BY" instruction, are unknown to Impala; Therefore, in this adaptation type, these kind of aliases have been replaced by their initial values.

Before

```
 [_LIMITA] select [_LIMITB]
   ...
  ,grouping(i_category)+grouping(i_class)
      as lochierarchy
  ...
 from
  ...
 where
  ...
 group by rollup(i_category,i_class)
order by
   lochierarchy desc
  ,case when lochierarchy = 0 then
     i_category end
  ,rank_within_parent
  [_LIMITC];
```

After

```
 [_LIMITA] select [_LIMITB]
   ...
  ,grouping(i_category)+grouping(i_class)
      as lochierarchy
  ...
 from
  ...
 where
  ...
 group by rollup(i_category,i_class)
order by
   lochierarchy desc
  ,case when
      grouping(i_category)+grouping(i_class)
      = 0 then i_category end
  ,rank_within_parent
  [_LIMITC];
```

4.3.3 Type C: Nested table expressions

As Impala does not allow to use nested table expressions without giving name to interior expressions, this adaption type have been considered in order to settle this problem.

Before

```
with wscs as
( select sold_date_sk, sales_price
   from ( select ws_sold_date_sk
       sold_date_sk, ws_ext_sales_price
       sales_price
           from web_sales
           union all
           select cs_sold_date_sk
               sold_date_sk,
               cs_ext_sales_price
               sales_price
           from catalog_sales )),
```

After

```
with wscs as
( select sold_date_sk, sales_price
   from ( select ws_sold_date_sk
       sold_date_sk, ws_ext_sales_price
       sales_price
           from web_sales
           union all
           select cs_sold_date_sk
               sold_date_sk,
               cs_ext_sales_price
               sales_price
           from catalog_sales ) AS tmp),
```

4.3.4 Type D: Reserved keywords

As the templates queries sometimes used some reserved keywords of *SQL*, this adaption type have been defined; The character "`" have been used around these words in order to transform them to normal aliases

Before

```
select s_store_id, sum(sales_price) as
    sales, sum(profit) as profit,
    sum(return_amt) as returns,
    sum(net_loss) as profit_loss
```

After

```
select s_store_id, sum(sales_price) as
    sales, sum(profit) as profit,
    sum(return_amt) as `returns`,
    sum(net_loss) as profit_loss
```

4.3.5 Type E: Alternatives

The queries number 10 and 35 could not be adapted to perform by Impala, therefore alternative version of these queries provided by *TPC-DS*, have been used.

4.3.6 All the adaptions in one sight

The following table shows all the queries that have been modified according to one of the types above;

Adaption type	Queries number
Type A	5, 12, 16, 20, 21, 32, 37, 40, 72, 77, 80, 82, 92, 94, 95, 98
Type B	36, 70, 86
Type C	2, 14, 23, 49
Type D	5, 77, 80, 99
Type E	10, 35

Table 3: Queries adaption

5 Benchmark results and analysis

In this section, the different benchmarks done as well as their results and some analysis will be seen.

It has been decided to set the size of the biggest data-set to 50 GB due to limited storage capacity, which represents the reference scale factor 1.0. Three other scale factors have been chosen, 0.02, 0.2 and 0.5, which give three datasets of size 1, 10, 25 GB in addition to the 50 GB one.

For each of the scale factor, the queries have been re-generated as described in the queries generation section.

In order to have more precise and reliable result, for each data-set the queries have been executed six times and the displayed result is the average of the last five executions. Except for the 50 GB data execution, where the queries have been run only once due to time related issues.

The four following figures 1, 2, 3 and 4 show the execution time of each query on the four datasets. It was chosen to show these results using this kind of chart instead of the typical histogram used in this type of situation, as they are more appropriate.

5.1 Experiment 1 : 1 GB dataset

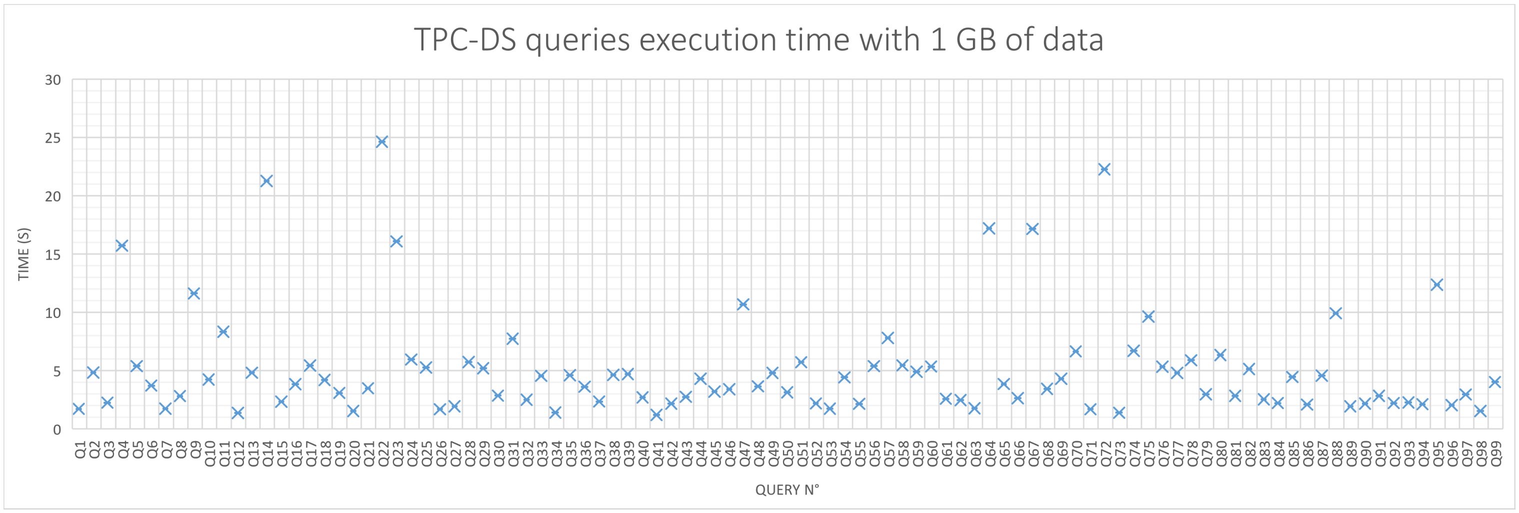

Figure 1: Execution time with 1 GB of data

It can be seen that the execution time of the 99 queries vary between 1.22 and 24.64 secs, the average execution time is equal to 5.12 secs. Some queries take a lot of time, up to 4 or 5 times more than the average ones. The overall execution time of all the queries is equal to 506.61 s.

5.2 Experiment 2 : 10 GB dataset

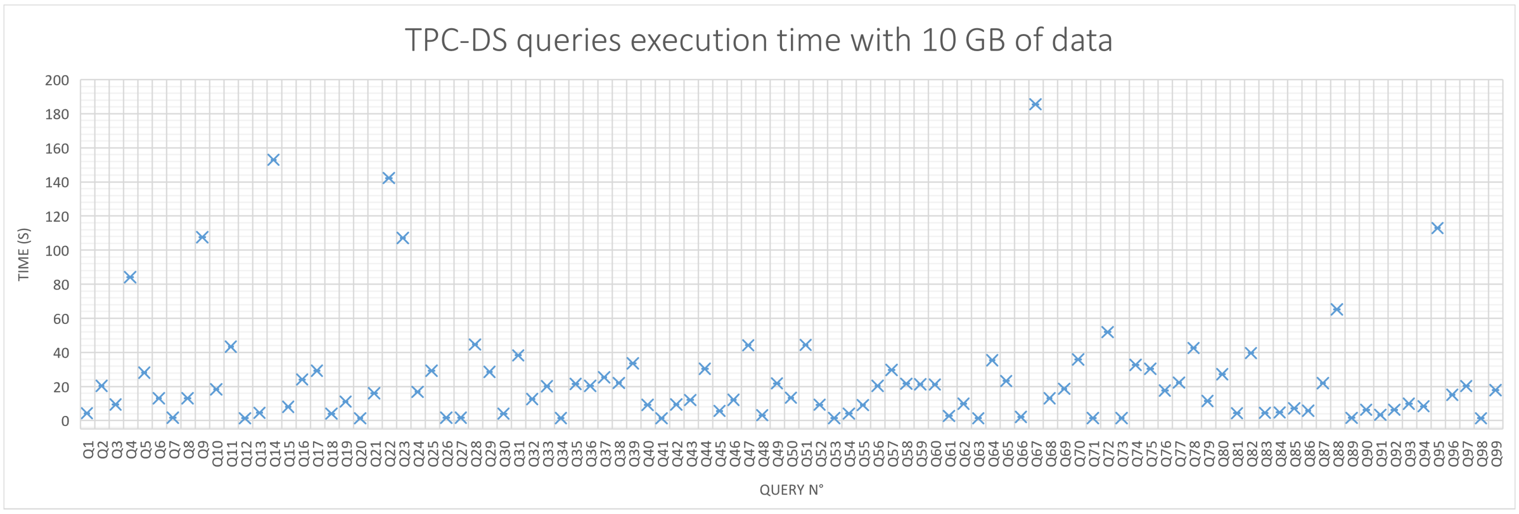

Figure 2: Execution time with 10 GB of data

It can be seen that the execution time of the 99 queries vary between 1.38 and 185.54 secs, the average execution time is equal to 24.63 s. Some queries take a lot of time, up to 4 or 5 times more than the average ones. The overall execution time of all the queries is equal to 2438.59 s.

5.3 Experiment 3 : 25 GB dataset

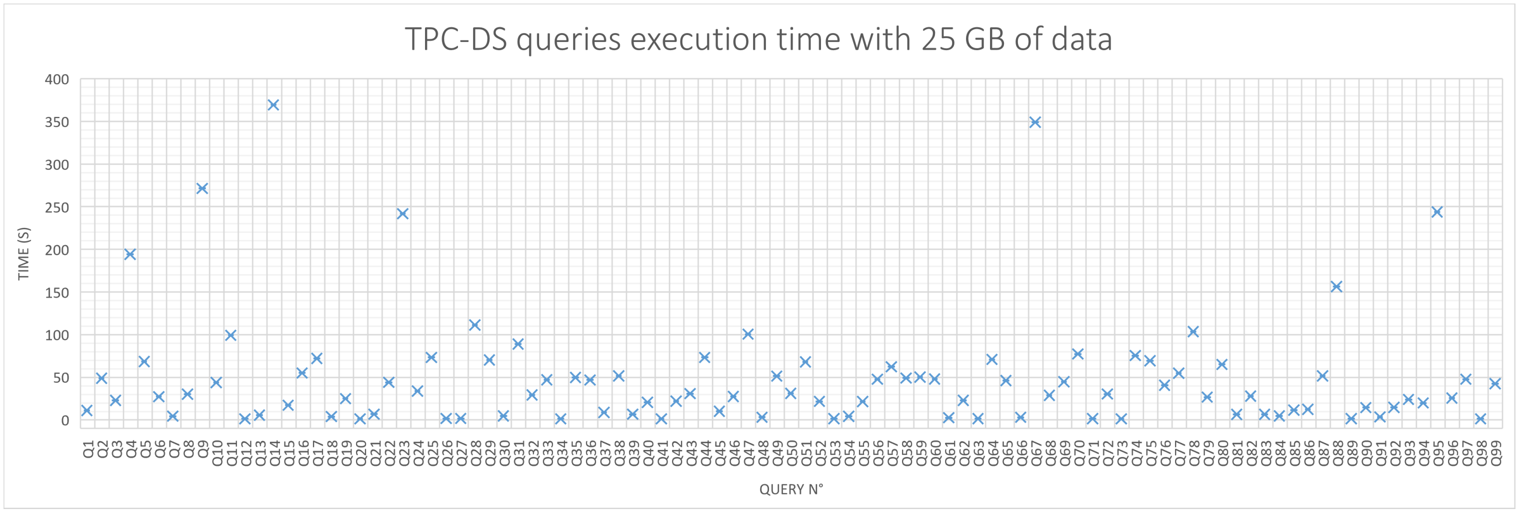

Figure 3: Execution time with 25 GB of data

It can be seen that the execution time of the 99 queries vary between 1.29 and 369.31 secs, the average execution time is equal to 49.28 secs. Some queries take a lot of time, up to 4 or 5 times more than the average ones. The overall execution time of all the queries is equal to 4878.49 s.

5.4 Experiment 4 : 50 GB dataset

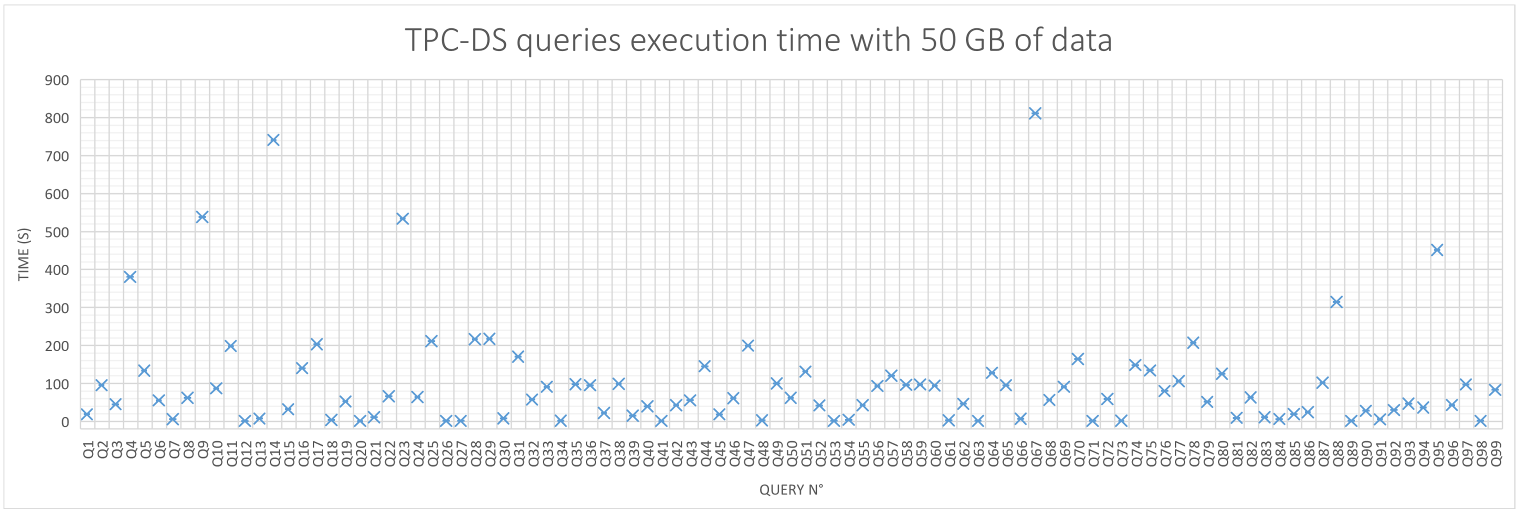

Figure 4: Execution time with 50 GB of data

It can be seen that the execution time of the 99 queries vary between 1.32 and 811.05 secs, the average execution time is equal to 100.38 secs. Some queries take a lot of time, up to 6 or 8 times more than the average ones. The overall execution time of all the queries is equal to 9937.49 s.

5.5 Execution time summary

In the two following figures 5 and 6, it can be seen more clearly the impact of the size of the datasets on the execution time of each query.

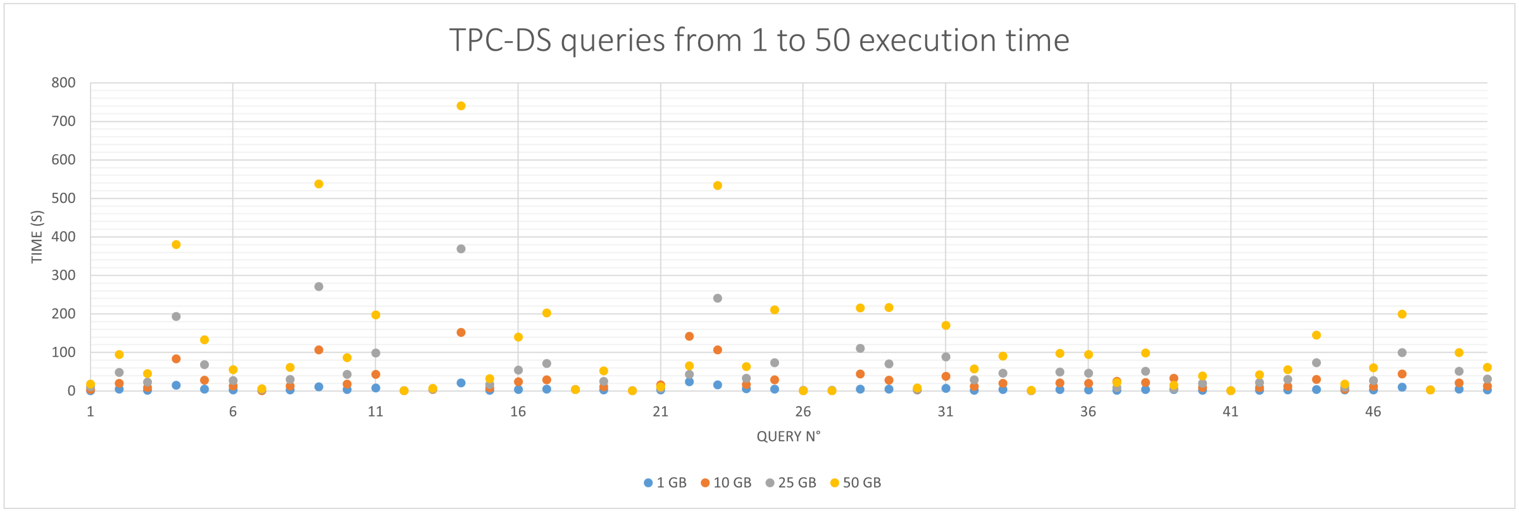

Figure 5: Execution time of queries from 1 to 50

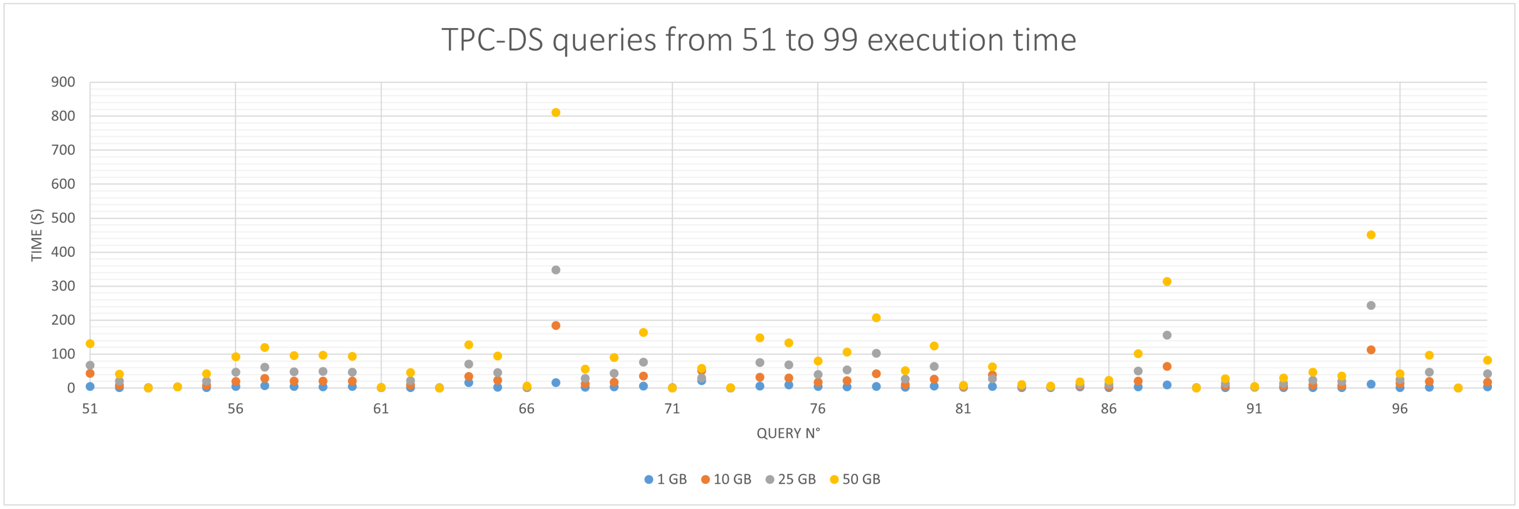

Figure 6: Execution time of queries from 51 to 99

The queries where only the yellow bullet can be seen, are queries where the size of the database has a negligible impact.

The results of the query 22 seem to be inconsistent, as it takes longer for it to complete with 10 GB of data than all the other scale. It can be explained as the queries are generated for a specific scale factor as described in TPC-DS documentation.

Some queries are highly dependent on the size of the data they operate on. Especially the following ones :

- Query 4 : It has a lot of **union all**'s. Which means that more than one table is included on the operations, which adds a non-negligible execution time.
- Query 9 : It has a lot of cases. The same operation has been repeated with different parameters, which means that the used table has been read many times.
- Query 14 : It has a lot of **intersect**'s, indeed the operation becomes more costly depending on the dataset size.
- Query 23 : It has a lot of **union all**'s and multiple nested **select**'s. Thus, the operation is more complex and a possible query optimization can improve the time of execution.
- Query 67 : It has a nested **select**'s with depth of 3.
- Query 95 : It has more than one **select** in a where statement. As a consequence, the **where** clause becomes more costly to verify, thus doing that twice is time-consuming.

5.6 Execution time growth

The following graph 7 shows the growth of execution time of all the queries compared to the size of the datasets. Notice that the execution time is calculated as the overall sum of the time execution of each query independently, and not the real time needed to execute all of them on the computer. These two values can be different due to different I/O operations used in the script.

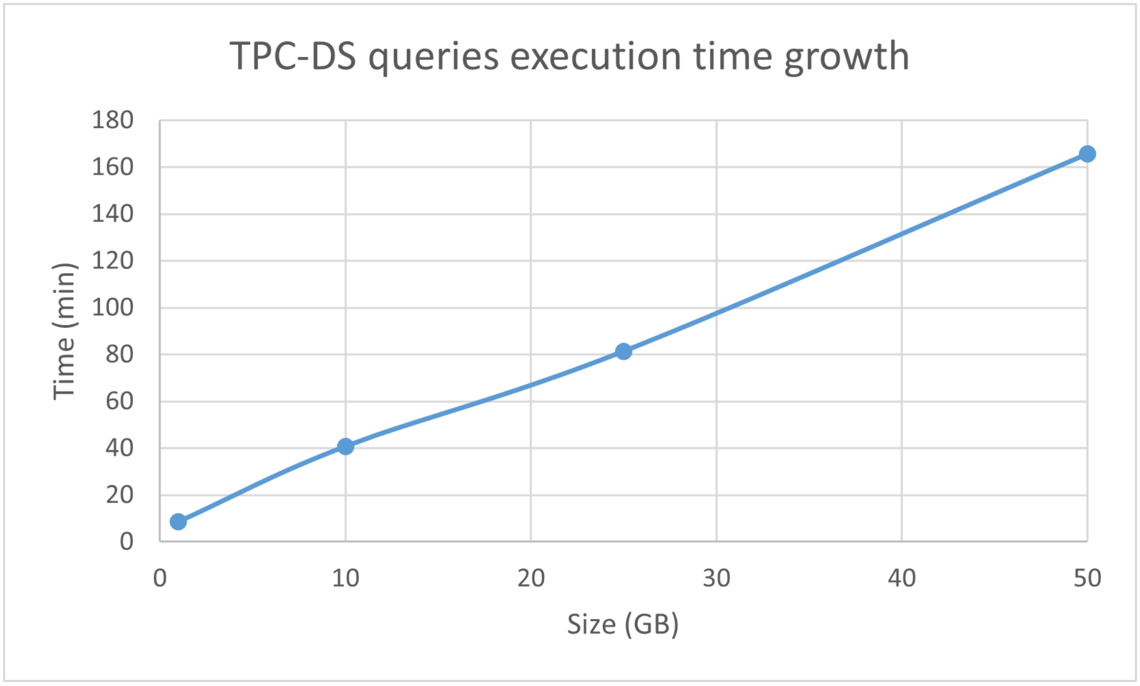

Figure 7: Execution time growth

It can be seen in the graph 7, the growth of the execution time of all the queries compared to the size of the datasets, is linear. It means that Impala handles well queries which ran on large amount of data. Furthermore, a line can be traced which pass through all the points. This line can be used to extrapolate higher values of data sizes. As the points are not perfectly aligned, the method of least square has been used to get its equation.

Here is its equation :

$$y = 3.169x + 5.859$$

The function can be seen in the following graph 8 with the dotted line.

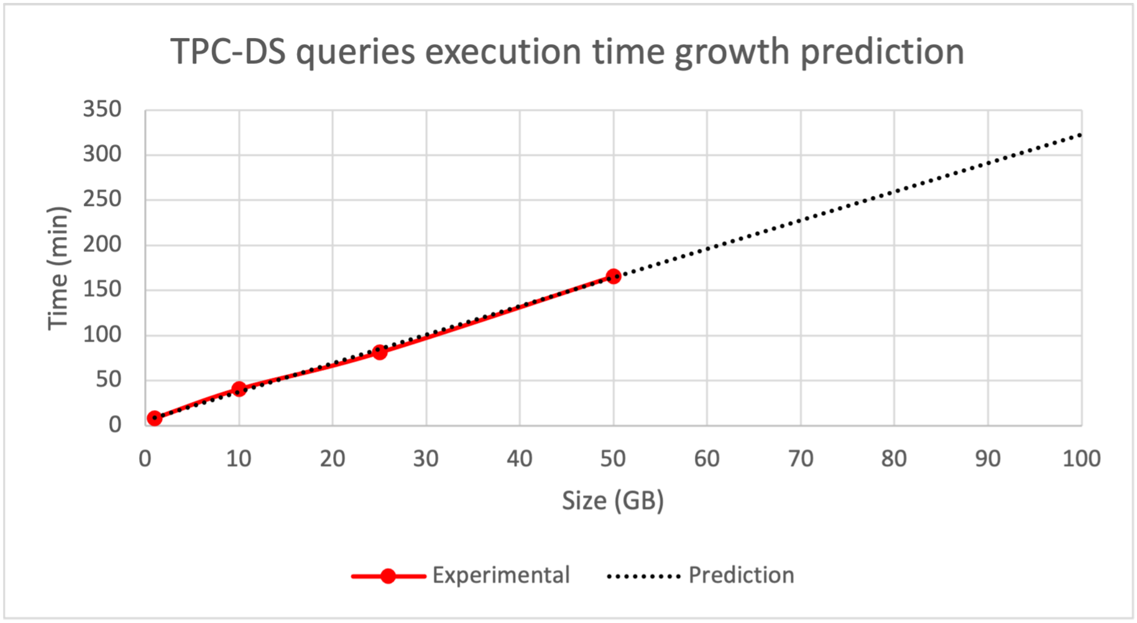

Figure 8: Execution time growth prediction

It means that for 100 GB of data, it is expected to get an execution time of 322.759 minutes.

6 Conclusion

In this project, *TPC-DS* tools have been used in order to get a benchmark of Apache Impala. In this report, all the necessary steps were described in order for anyone to be able to reproduce the obtained results.

The first parts of this project were centered on the definitions of the different tools used. Additionally, all the steps needed for anyone to be able to install, launch and use Impala were fully detailed.

After that, the directives used to get the required materials, which were composed of the queries and the datasets, were seen. As all queries of *TPC-DS* were not directly compatible with Impala, some of these queries have been syntactically altered and some were replaced by official alternatives one, so that they all could be executed via Impala.

As the diagrams of previous chapters show, it can be seen that the execution time increases with the size of the data, not exponentially but linearly. It can be assessed that Impala is indeed designed for large data warehouses, as it is capable of holding terabytes of data.

To sum up, it seems that Impala results are satisfying, even though very large datasets have not been tested because of the lack of resources.

References

[1] *Transaction Processing Performance Council (TPC), TPC BENCHMARK™ DS Standard Specification — Version 3.2.0, June 2021*