Analysis of the Influence of Modeling, Data Format and Processing Tool on the Performance of Hadoop-Hive Based Data Warehouse

Beatriz F. P. de Oliveira, Aline S.O. Valente, Marcio Victorino, Edward Ribeiro, Maristela Holanda

Universidade de Brasília, Brazil
{beatrizfra, asoliveirav}@gmail.com, {mcvictorino, edwardribeiro, mholanda}@unb.br

Abstract. With the emergence of Big Data and the continuous growth of massive data produced by web applications, smartphones, social networks, and others, organizations began to invest in alternative solutions that would derive value from this amount of data. In this context, this article evaluates three factors that can significantly influence the performance of Big Data Hive queries: data modeling, data format and processing tool. The objective is to present a comparative analysis of the Hive platform performance with the snowflake model and the fully denormalized one. Moreover, the influence of two types of table storage file types (CSV and Parquet) and two types of data processing tools, Hadoop and Spark, were also comparatively analyzed. The data used for analysis is the open data of the Brazilian Army in the Google Cloud environment. Analysis was performed for different data volumes in Hive and cluster configuration scenarios. The results yielded that the Parquet storage format always performed better than when CSV storage formats were used, regardless of the model and processing tool selected for the test scenario.

Categories and Subject Descriptors: H.3 [**Information Storage and Retrieval**]: Evaluation of retrieval results

Keywords: Data Format, Data Warehouse, Hadoop, Hive, Modeling, Spark

1. INTRODUCTION

Data Warehouse (DW) is a subject-oriented, integrated, time-variable, and non-volatile collection of data to support the management of decision-making processes [Inmon 2005]. *Integrated*, data is gathered in the data warehouse from multiple disparate sources and consolidated into a healthy, consistent environment. These sources can be internal organizational systems, external systems, spreadsheets, etc. *Time Variable*, every datum in the DW is identified by a specific period. This period can be a day, month, year, term, etc. *Non-volatile*, in a DW, data are stable. Furthermore, they are only added. It means they should not be removed. In addition, data will follow the same temporal version, indefinitely.

In DW, it is common to have a centralized architecture to facilitate data analysis. However, the volumes of analytics data are reaching critical sizes [Jacobs 2009], requiring different solutions for the processing and management of Big Data. Therefore, it is possible to have distributed DW with different structures [Mohanty et al. 2013]. In this context, companies have developed a new ecosystem of platforms that uses not only the data from the traditional DW implemented, but also hybrid DW architectures, called Big Data Warehouses (BDW) [Mohanty et al. 2013]. Big Data Warehouses differ substantially from traditional DW as their schema must be based on new and more flexible logical models than the relational models [Di Tria et al. 2014].

Apache Hadoop and Apache Spark are open-source frameworks. The former allows distributed

Copyright©2022 Permission to copy without fee all or part of the material printed in JIDM is granted provided that the copies are not made or distributed for commercial advantage, and that notice is given that copying is by permission of the Sociedade Brasileira de Computação.

processing of large datasets in a computer cluster; the latter runs in a cluster for large-scale data processing, built with a focus on performance. Currently, different tools facilitate the work of extracting value and knowledge from Big Data. Therefore, it is necessary to compare them to know which ones have the best performance for each specific situation. In the BDW scenario, Hive is a Data Warehouse system for Big Data contexts that organizes data into tables, partitions, and buckets. Therefore, Hive is a very popular system because it uses the HIVEQL language, similar to SQL. In Hive, there is no single format in which data should be stored. It supports text files with comma-separated values (CSV), tab-separated values (TSV), or other formats such as Parquet and Optimized Row Columnar (ORC). Parquet is an open source file format available for projects in the Hadoop ecosystem, which is designed for a columnar storage format [Weintraub et al. 2021], [Rodrigues et al. 2019], as opposed to row-based files such as CSV. Hive has been used to implement Big Data Warehouse architectures.

This article is an extension of the article published in the proceedings of the 2021 Brazilian Symposium on Databases (SBBD), [de Oliveira et al. 2021]. In this extended version, we broadened the test scenarios with new experiments. Hence, the present work aims to analyze the performance of Hive using two different data models. The first employs a fully denormalized table (FDT), and the second follows the dimensional model (DM). The influence of the table data format on query execution time was also analyzed, comparing the CSV and Parquet formats. In addition, all simulations were run on the Apache Hadoop and Apache Spark platforms, to compare their different types of data processing. Thus, simulations were made with different workloads, and then the results were analyzed and compared.

This work is structured as follows: Section 2 presents the concepts and theory involved in the article; Section 3 describes the most relevant related work; Section 4 presents the case study; Section 5 presents the results; Section 6 presents the discussion of the results and Section 7, the conclusion and future work.

2. BACKGROUND

2.1 Apache Hadoop and Apache Spark

Apache Hadoop is an open-source framework, written in Java, for the storage and distributed processing of large datasets in computer clusters. It uses simple programming models, organized in a master node and several workers, and each worker machine provides local processing and storage. This design provides scalability and high availability as it is designed to detect and handle failures to provide the most available service possible. The main modules of Hadoop are as follows:

(1) Hadoop Distributed File System: It divides the data into blocks and stores them in nodes over the distributed cluster architecture. It allows the storage of distributed large data sets. HDFS is the primary distributed storage used by Hadoop applications.
(2) Hadoop Common: It contains libraries and common files required by all Hadoop modules.
(3) Hadoop YARN: It is the Hadoop cluster resource management system. It is Hadoop's cluster resource management system. It separates the resource management layer from the processing layer and enables distributed computing frameworks (MapReduce, Spark) to run as YARN applications on the cluster compute layer (YARN) and the cluster storage layer (HDFS) [White 2015].
(4) Hadoop Map Reduce: Programming model used in the distributed data processing.

Over the years, new components were incorporated into the architecture to solve specific problems. The Apache Hadoop architecture can be defined as follows [White 2015]:

—Data Storage Layer: Hadoop Distributed File System (HDFS);
—Data Processing Layer: YARN and MapReduce; and

—Data Access Layer: Tools such as Apache Hive, for abstraction from the SQL language as an interface to the MapReduce commands.

Apache Spark is a cluster-computing platform for large-scale data processing. Although Spark does not use MapReduce as an execution engine, it has many parallels with MapReduce in terms of API and runtime. Unlike Apache Hadoop, Spark runs on RAM and not on hard drives. It caches the data required for computation in the memory of the cluster nodes. This capability to keep large datasets in memory between jobs requires Spark to outperform the equivalent MapReduce workflow, making it very fast [White 2015].

Spark supports interactive queries and iterative algorithms and can be used with real-time data analytics. It can run on YARN and an existing Hadoop cluster. Concerning file access, Spark handles data from a variety of storage systems, such as cloud storage systems, Apache Cassandra, and HDFS [Chambers and Zaharia 2018].

When you have an existing Hadoop cluster, it is convenient to run Spark on YARN. The objective is to provide good integration with the other Hadoop components from its ecosystem, such as Pig, Flume and Hive.

2.2 Apache Hive

Apache Hive is an open-source DW infrastructure built on top of Hadoop, which facilitates queries and management of large volumes of data stored in a distributed environment [Cassavia et al. 2014] and [Sandoval 2015]. It has been used to implement Big Data Warehouse architectures. Data processing performance in Hive depends on factors such as: software configuration parameters, query performed, data volume, and data modeling, among other characteristics.

When performing a query or command, the Driver compiles the input, optimizes the required computation, and performs the necessary steps, usually with MapReduce jobs in Hadoop; however, the Execution Engine can be MapReduce, Tez, or Spark. Figure 1 presents this architecture with more details.

2.3 Types of Models and Data File Formats

Dimensional modeling in a DW plays a decisive role in the performance of queries to be performed. When modeling a DW, descriptive, textual data is divided from numeric data. The first make up the tables known as dimensions and the latter, tables known as facts. Fact tables are also populated with foreign keys. The two most well-known models/schemes are the Star and the Snowflake. The Star

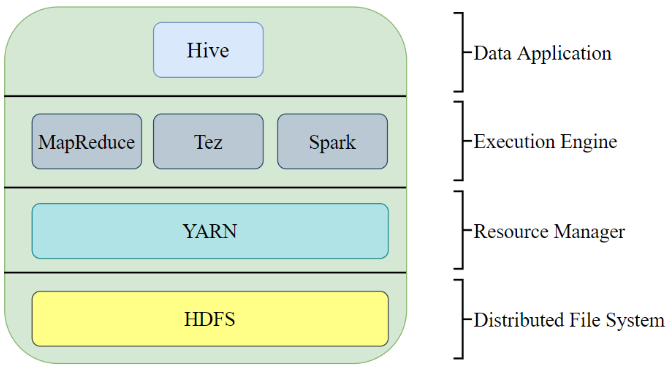


Fig. 1. Hive Architecture.

schema is an optimized structure that uses one centralized fact table indexing one or many dimensional tables.

The Snowflake schema is a more complex model and seeks to reduce storage redundancy. It is characterized by dimensions of a schema that are detailed and have a hierarchical relationship, so the child tables have multiple parent tables. Its name is due to the design that resembles a snowflake. This type of schema makes data navigation more difficult.

The total denormalized model employs a single large table to define all existing relationships. It is not widely used due to its large size, generated by data redundancy; however, it can be an excellent choice for certain situations. As mentioned in [Vajk et al. 2013], when the total data denormalization occurs, it is possible to improve query performance.

The types of data file formats used in this article are comma-separated values (CSV) and Parquet.

CSV files contain comma-separated data records. It is a row-based file format, which means that each line is a new record and each row in the file will be a row in the Hive table. Its structure usually contains a header row that provides a structure to the file. One of the advantages of this file format is the ease of reading and understanding the contents of file data.

Apache Parquet is an open-source columnar-storage format that enables greater efficiency in terms of file size and query performance.

In Parquet, file sizes are usually smaller than in CSV format, and its structure allows a very efficient encoding. In terms of query performance, its improvement occurs because the query engine can skip columns that are not required to answer a query [White 2015]. All projects in the Hadoop ecosystem support Parquet.

3. RELATED WORKS

In the literature, there are different works with the theme of DW and Hive for Big Data such as [Costa et al. 2017], [Santos and Costa 2016], [Plase et al. 2017], [Luckow et al. 2015] and [Rodrigues et al. 2019].

[Costa et al. 2017] investigated the impact of data organization and modeling on the processing times of BDWs implemented in Hive, benchmarking multidimensional star schema and fully denormalized tables with different Scale Factors (SFs). In [Plase et al. 2017], the authors evaluated the performance of data queries, comparing file formats such as Avro and Parquet with text formats, using Apache Hadoop.

[Luckow et al. 2015] investigated processing and execution management frameworks and higher-level frameworks for SQL and advanced analytics. The authors used a customized benchmark with five queries and evaluated the suitability of Hive (with Tez) and Spark-SQL using different storage formats: Text, Parquet, and ORC.

[Santos and Costa 2016] showed that the data model can change according to the DW requirements and highlighted the importance of choosing the most second model for each need. [Rodrigues et al. 2019] evaluated different Big Data processing tools such as Drill, HAWQ, Hive, Impala, Presto, and Spark, using the Transaction Processing Performance Council (TPC-H) benchmark.

Differently from the works mentioned before, this article compares the performance of two data models in Hive and brings two more contributions:

(1) analysis of the influence of table data format on the query execution time, comparing CSV and Parquet formats; and
(2) data processing on Apache Hadoop and Apache Spark platforms, to compare the execution times of the related queries.

All the cited articles analyzed some of the factors mentioned, but not altogether, as in the present work.

4. CASE STUDY

This section presents the infrastructure used to process the queries, the dataset, queries' characteristics, the test scenarios, and the method applied to obtain the results. In this article, the data sources will be HDFS files for Hadoop and Spark processing.

4.1 Dataset and Queries

This work used an open data set of the Brazilian military service, with records referring to the military enlistment. The choice of this data set did not occur due to the great exercise of it in the context only, but also of the country. The objective of this work is to evaluate the performance of different data models, comparing the dimensional model with a fully denormalized model. In addition, a performance comparison of the queries is made, varying the way the data is stored, Parquet and CSV, and varying the data processing between Hadoop and Spark.

The complete DW, represented by the snowflake dimensional model, consists of 1 FACT table and 23 dimensions, with several columns ranging from 4 to 86. Some tables will be explained below, but briefly. The data in the tables are related to the enlisted citizen and contain information such as registration number, age, biometric data, city and level of education; in addition, it presents if he wishes to serve, if he has an occupation, among others. It is important to highlight that the queries processed in this work analyzed the FACT table plus fourteen dimensions. The name and amount of columns of the tables used by the queries of this article are presented in Table I. Moreover, the number of columns of the fully denormalized table is 268, because it was constructed by joining all the columns of all the DW tables.

The Fact table is CITIZEN_EVENT_FACT. It has 11 columns and stores data such as enlistment number, dates, geographic and military codes, and age, among others. Hence, the analyzed dimensions are:

—Geography_Dim, geographic dimension that stores geographic data related to the citizen, such as city, state, country of birth and residence, and geographic data of the military enlistment organization, among others;
—Time_Dim, time dimension stores data related to time measurements and their variations such as YEAR, WEEK, DATE, and DAY;
—Armed_Forced_Dim describes the Armed Force to which the data is related;
—Mil_Wish_Serve_Dim describes which Armed Force the citizen wishes to serve or if he does not wish to serve;
—Mobiliz_Dest_Dim, this dimension presents the cities to which the incorporated citizens will be mobilized;
—Resid_Zone_Dim describes the type of residential zone the enlisted citizen lives;
—Reserve_Class_Dim describes the type of reserve class the citizen will be classified;
—Behavior_Dim describes the classification of the soldier's behavior;
—Mil_Sit_Dim describes the military situation of the citizen, for example, if he was excluded, enlisted, capable, incapable, among others;
—Occupation_Dim describes all the occupations/jobs the citizens may have when enlisting;
—JSM_Dim describes all the information related to the military service board;
—Spec_Rank_Grad_Dim describes all the information related to the military's rank or graduation;

Table I. Characteristics of the tables processed

Table Name	Number of Columns
Citizen_Event_Fact	11
Geography_Dim	86
Time_Dim	23
Armed_Forced_Dim	4
Mil_Wish_Serve_Dim	4
Mobiliz_Dest_Dim	4
Resid_Zone_Dim	4
Reserve_Class_Dim	4
Behavior_Dim	4
Mil_Sit_Dim	4
Occupation_Dim	4
JSM_Dim	22
Spec_Rank_Grad_Dim	11
Citizen_Dim	48
Mil_Sv_Event_Dim	4
FDT	268

—Citizen_Dim, one of the most important dimensions, as it contains all the personal data of the enlisted citizen; and

—Mil_Sv_Event_Dim, table that contains the code and description of military events related to military service.

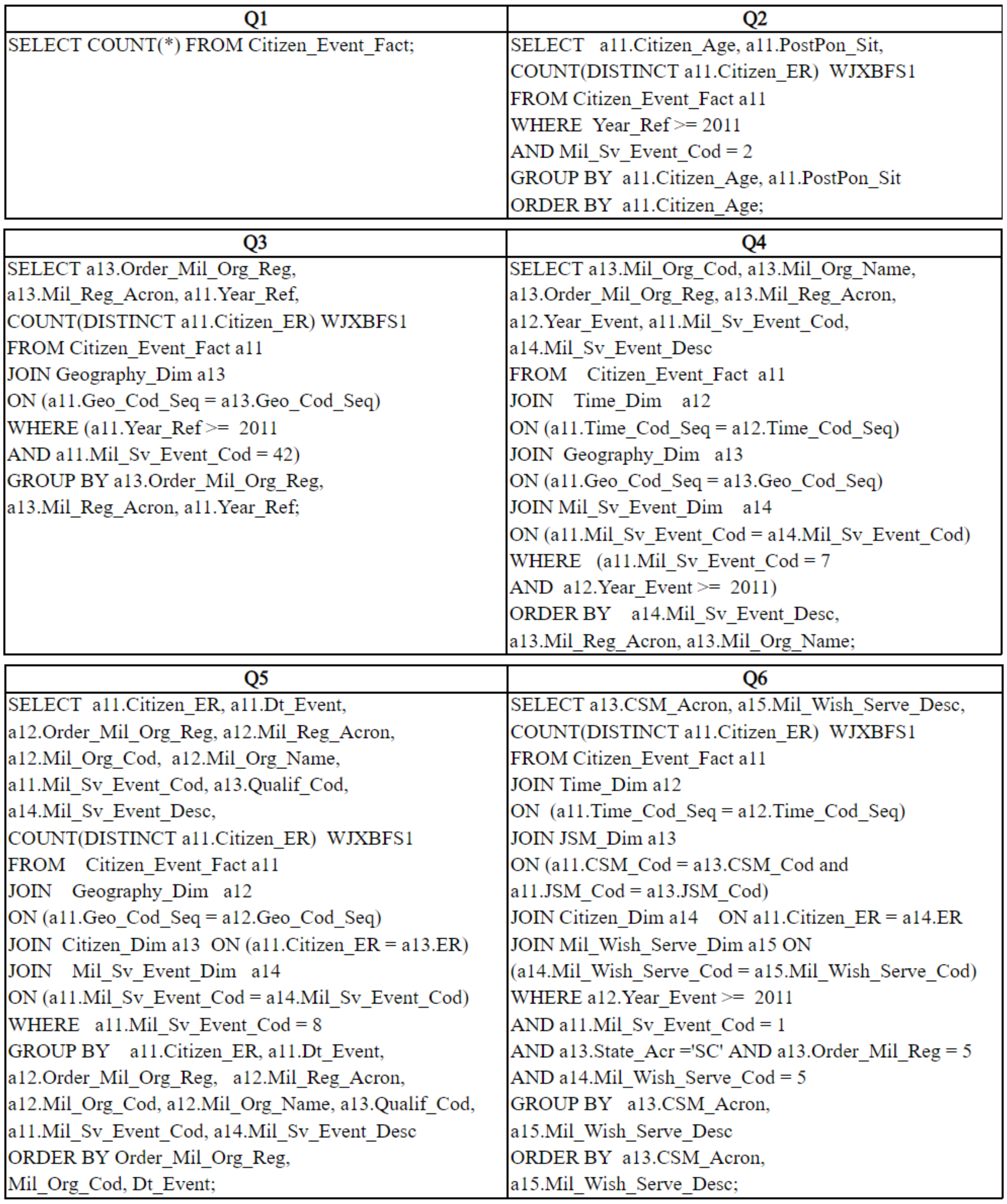
This work comprises eight queries. Figures 2 and 3 show each specific query. Subsequently, Table II shows the complexity of each query analyzed. And also to help the better understanding of the aforementioned queries, Table III shows the queries summary, ordering them by increasing complexity.

In addition, Table IV presents the loading time of the fully denormalized table with CSV data format, in three specific years, using Hadoop and Spark. Table V presents the cardinalities of the attributes involved in the GROUP BY and ORDER BY clauses, to support the results presented.

Considering that the dataset used was not an established benchmark such as the TPC-H, the eight generated queries sought to follow the pattern of the queries profile used in the TPC-H benchmark. Therefore, the queries explored different factors such as the number of JOINS, filters, the number of attributes analyzed and the presence or absence of GROUP BY and ORDER BY. This heterogeneity of query profiles favors the strengthening of the results of this research.

Table II. Queries' complexity

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Joins	0	0	1	3	3	4	7	12
Filters	0	2	2	2	1	5	3	3
Group by	0	2	3	0	8	2	13	0
Order by	0	3	0	3	3	2	0	0
Attribute(s)	1	5	5	7	9	8	14	15



Q1	Q2

SELECT COUNT(*) FROM Citizen_Event_Fact;	SELECT a11.Citizen_Age, a11.PostPon_Sit,
	COUNT(DISTINCT a11.Citizen_ER) WJXBFS1
	FROM Citizen_Event_Fact a11
	WHERE Year_Ref >= 2011
	AND Mil_Sv_Event_Cod = 2
	GROUP BY a11.Citizen_Age, a11.PostPon_Sit
	ORDER BY a11.Citizen_Age;

Q3	Q4

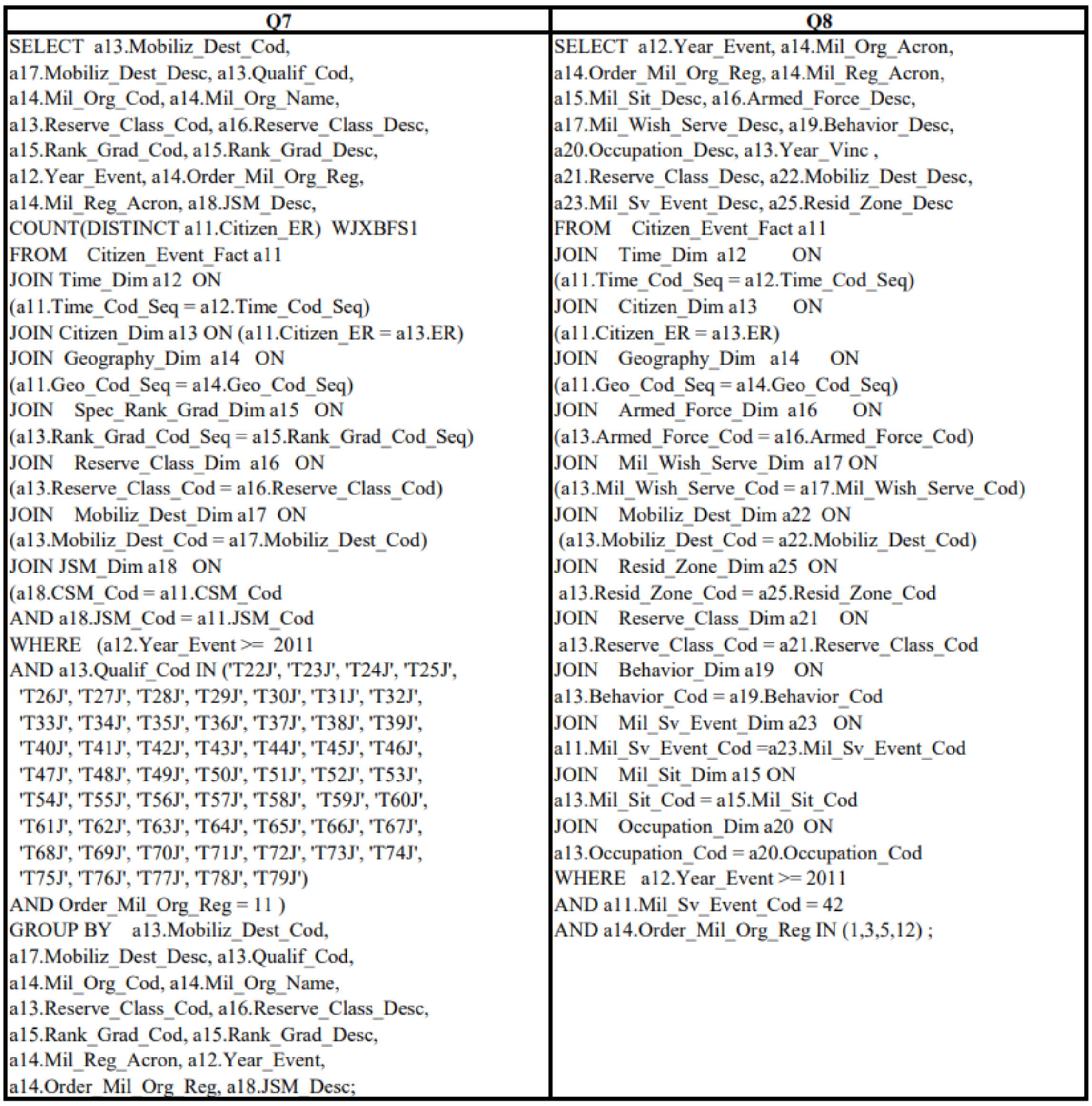
SELECT a13.Order_Mil_Org_Reg,
a13.Mil_Reg_Acron, a11.Year_Ref,
COUNT(DISTINCT a11.Citizen_ER) WJXBFS1
FROM Citizen_Event_Fact a11
JOIN Geography_Dim a13
ON (a11.Geo_Cod_Seq = a13.Geo_Cod_Seq)
WHERE (a11.Year_Ref >= 2011
AND a11.Mil_Sv_Event_Cod = 42)
GROUP BY a13.Order_Mil_Org_Reg,
a13.Mil_Reg_Acron, a11.Year_Ref;
	SELECT a13.Mil_Org_Cod, a13.Mil_Org_Name,
	a13.Order_Mil_Org_Reg, a13.Mil_Reg_Acron,
	a12.Year_Event, a11.Mil_Sv_Event_Cod,
	a14.Mil_Sv_Event_Desc
	FROM Citizen_Event_Fact a11
	JOIN Time_Dim a12
	ON (a11.Time_Cod_Seq = a12.Time_Cod_Seq)
	JOIN Geography_Dim a13
	ON (a11.Geo_Cod_Seq = a13.Geo_Cod_Seq)
	JOIN Mil_Sv_Event_Dim a14
	ON (a11.Mil_Sv_Event_Cod = a14.Mil_Sv_Event_Cod)
	WHERE (a11.Mil_Sv_Event_Cod = 7
	AND a12.Year_Event >= 2011)
	ORDER BY a14.Mil_Sv_Event_Desc,
	a13.Mil_Reg_Acron, a13.Mil_Org_Name;

Q5	Q6
SELECT a11.Citizen_ER, a11.Dt_Event, a12.Order_Mil_Org_Reg, a12.Mil_Reg_Acron, a12.Mil_Org_Cod, a12.Mil_Org_Name, a11.Mil_Sv_Event_Cod, a13.Qualif_Cod, a14.Mil_Sv_Event_Desc, COUNT(DISTINCT a11.Citizen_ER) WJXBFS1 FROM Citizen_Event_Fact a11 JOIN Geography_Dim a12 ON (a11.Geo_Cod_Seq = a12.Geo_Cod_Seq) JOIN Citizen_Dim a13 ON (a11.Citizen_ER = a13.ER) JOIN Mil_Sv_Event_Dim a14 ON (a11.Mil_Sv_Event_Cod = a14.Mil_Sv_Event_Cod) WHERE a11.Mil_Sv_Event_Cod = 8 GROUP BY a11.Citizen_ER, a11.Dt_Event, a12.Order_Mil_Org_Reg, a12.Mil_Reg_Acron, a12.Mil_Org_Cod, a12.Mil_Org_Name, a13.Qualif_Cod, a11.Mil_Sv_Event_Cod, a14.Mil_Sv_Event_Desc ORDER BY Order_Mil_Org_Reg, Mil_Org_Cod, Dt_Event;	SELECT a13.CSM_Acron, a15.Mil_Wish_Serve_Desc, COUNT(DISTINCT a11.Citizen_ER) WJXBFS1 FROM Citizen_Event_Fact a11 JOIN Time_Dim a12 ON (a11.Time_Cod_Seq = a12.Time_Cod_Seq) JOIN JSM_Dim a13 ON (a11.CSM_Cod = a13.CSM_Cod and a11.JSM_Cod = a13.JSM_Cod) JOIN Citizen_Dim a14 ON a11.Citizen_ER = a14.ER JOIN Mil_Wish_Serve_Dim a15 ON (a14.Mil_Wish_Serve_Cod = a15.Mil_Wish_Serve_Cod) WHERE a12.Year_Event >= 2011 AND a11.Mil_Sv_Event_Cod = 1 AND a13.State_Acr ='SC' AND a13.Order_Mil_Reg = 5 AND a14.Mil_Wish_Serve_Cod = 5 GROUP BY a13.CSM_Acron, a15.Mil_Wish_Serve_Desc ORDER BY a13.CSM_Acron, a15.Mil_Wish_Serve_Desc;

Fig. 2. Queries' description - Q1 to Q6.

4.2 Methodology

To evaluate the performance of the two proposed models, a cluster was built on Google Cloud with five nodes. The metric selected to measure the performance was the query execution time in seconds(s). The queries were executed ten times and then the average of the measurements was calculated.



Q7

```
SELECT  a13.Mobiliz_Dest_Cod,
a17.Mobiliz_Dest_Desc, a13.Qualif_Cod,
a14.Mil_Org_Cod, a14.Mil_Org_Name,
a13.Reserve_Class_Cod, a16.Reserve_Class_Desc,
a15.Rank_Grad_Cod, a15.Rank_Grad_Desc,
a12.Year_Event, a14.Order_Mil_Org_Reg,
a14.Mil_Reg_Acron, a18.JSM_Desc,
COUNT(DISTINCT a11.Citizen_ER)  WJXBFS1
FROM   Citizen_Event_Fact a11
JOIN Time_Dim a12  ON
(a11.Time_Cod_Seq = a12.Time_Cod_Seq)
JOIN Citizen_Dim a13 ON (a11.Citizen_ER = a13.ER)
JOIN  Geography_Dim  a14   ON
(a11.Geo_Cod_Seq = a14.Geo_Cod_Seq)
JOIN    Spec_Rank_Grad_Dim a15   ON
(a13.Rank_Grad_Cod_Seq = a15.Rank_Grad_Cod_Seq)
JOIN    Reserve_Class_Dim  a16   ON
(a13.Reserve_Class_Cod = a16.Reserve_Class_Cod)
JOIN    Mobiliz_Dest_Dim a17  ON
(a13.Mobiliz_Dest_Cod = a17.Mobiliz_Dest_Cod)
JOIN JSM_Dim a18   ON
(a18.CSM_Cod = a11.CSM_Cod
AND a18.JSM_Cod = a11.JSM_Cod
WHERE   (a12.Year_Event >=  2011
AND a13.Qualif_Cod IN ('T22J', 'T23J', 'T24J', 'T25J',
  'T26J', 'T27J', 'T28J', 'T29J', 'T30J', 'T31J', 'T32J',
  'T33J', 'T34J', 'T35J', 'T36J', 'T37J', 'T38J', 'T39J',
  'T40J', 'T41J', 'T42J', 'T43J', 'T44J', 'T45J', 'T46J',
  'T47J', 'T48J', 'T49J', 'T50J', 'T51J', 'T52J', 'T53J',
  'T54J', 'T55J', 'T56J', 'T57J', 'T58J',  'T59J', 'T60J',
  'T61J', 'T62J', 'T63J', 'T64J', 'T65J', 'T66J', 'T67J',
  'T68J', 'T69J', 'T70J', 'T71J', 'T72J', 'T73J', 'T74J',
  'T75J', 'T76J', 'T77J', 'T78J', 'T79J')
AND Order_Mil_Org_Reg = 11 )
GROUP BY    a13.Mobiliz_Dest_Cod,
a17.Mobiliz_Dest_Desc, a13.Qualif_Cod,
a14.Mil_Org_Cod, a14.Mil_Org_Name,
a13.Reserve_Class_Cod, a16.Reserve_Class_Desc,
a15.Rank_Grad_Cod, a15.Rank_Grad_Desc,
a14.Mil_Reg_Acron, a12.Year_Event,
a14.Order_Mil_Org_Reg, a18.JSM_Desc;
```

Q8

```
SELECT  a12.Year_Event, a14.Mil_Org_Acron,
a14.Order_Mil_Org_Reg, a14.Mil_Reg_Acron,
a15.Mil_Sit_Desc, a16.Armed_Force_Desc,
a17.Mil_Wish_Serve_Desc, a19.Behavior_Desc,
a20.Occupation_Desc, a13.Year_Vinc ,
a21.Reserve_Class_Desc, a22.Mobiliz_Dest_Desc,
a23.Mil_Sv_Event_Desc, a25.Resid_Zone_Desc
FROM    Citizen_Event_Fact a11
JOIN    Time_Dim  a12        ON
(a11.Time_Cod_Seq = a12.Time_Cod_Seq)
JOIN    Citizen_Dim a13       ON
(a11.Citizen_ER = a13.ER)
JOIN    Geography_Dim   a14      ON
(a11.Geo_Cod_Seq = a14.Geo_Cod_Seq)
JOIN    Armed_Force_Dim  a16       ON
(a13.Armed_Force_Cod = a16.Armed_Force_Cod)
JOIN    Mil_Wish_Serve_Dim  a17 ON
(a13.Mil_Wish_Serve_Cod = a17.Mil_Wish_Serve_Cod)
JOIN    Mobiliz_Dest_Dim a22  ON
 (a13.Mobiliz_Dest_Cod = a22.Mobiliz_Dest_Cod)
JOIN    Resid_Zone_Dim a25  ON
 a13.Resid_Zone_Cod = a25.Resid_Zone_Cod
JOIN    Reserve_Class_Dim a21    ON
 a13.Reserve_Class_Cod = a21.Reserve_Class_Cod
JOIN    Behavior_Dim a19    ON
a13.Behavior_Cod = a19.Behavior_Cod
JOIN    Mil_Sv_Event_Dim a23    ON
a11.Mil_Sv_Event_Cod =a23.Mil_Sv_Event_Cod
JOIN    Mil_Sit_Dim a15 ON
a13.Mil_Sit_Cod = a15.Mil_Sit_Cod
JOIN    Occupation_Dim a20  ON
a13.Occupation_Cod = a20.Occupation_Cod
WHERE   a12.Year_Event >= 2011
AND a11.Mil_Sv_Event_Cod = 42
AND a14.Order_Mil_Org_Reg IN (1,3,5,12) ;
```

Fig. 3. Queries' description - Q7 and Q8.

The data used refers to the years 2011 to 2018. We simulate the DW construction by inserting data into the fact table sorted by date in yearly batches, increasing the table cardinality. We use reference names A1 to A8 to identify the fact table with increasing cardinalities.

Table VI shows the time interval in years of the loaded data batches, their reference names, the respective number of rows (totaling 35,073,178 rows), and the number of rows returned for queries from Q1 to Q8.

4.3 Test Scenarios

The Google Cloud environment consists of a Hadoop cluster with five nodes, configured as one master (YARN Resource Manager) and four workers (YARN Node Managers). The master node configuration was named by name: Optimized for Compute C2 Standard, with four CPUs with 16 GB of memory

Table III. Queries summary

Query	Fact	Dimension(s)	Attribute(s)	Condition(s)	Group by	Order by
Q1	Citizen_Event_Fact	-	count(*)	-	-	-
Q2	Citizen_Event_Fact	-	Citizen_Age, PostPon_Sit, count(distinct Citizen_ER), Year_Ref, Mil_Sv_Event_Cod	Year_Ref >= 2011, Mil_Sv_Event_Cod = '2'	Citizen_Age, PostPon_Sit	Citizen_Age
Q3	Citizen_Event_Fact	Geography_Dim	Order_Mil_Org_Reg, Mil_Reg_Acron, Year_Ref, count(distinct Citizen_ER), Mil_Sv_Event_Cod	Year_Ref >= 2011, Mil_Sv_Event_Cod = '42'	Order_Mil_Org_Reg, Mil_Reg_Acron, Year_Ref	-
Q4	Citizen_Event_Fact	Time_Dim, Geography_Dim, Mil_Sv_Event_Dim	Mil_Org_Cod, Mil_Org_Name, Order_Mil_Org_Reg, Mil_Reg_Acron, Year_Event, Mil_Sv_Event_Cod, Mil_Sv_Event_Desc	Year_Event >= '2011', Mil_Sv_Event_Cod = '7'	-	Mil_Sv_Event_Desc, Mil_Reg_Acron, Mil_Org_Name
Q5	Citizen_Event_Fact	Citizen_Dim, Geography_Dim, Mil_Sv_Event_Dim	Citizen_ER, Dt_Event, Order_Mil_Org_Reg, Mil_Reg_Acron, Mil_Org_Cod, Mil_Org_Name, Mil_Sv_Event_Cod, Mil_Sv_Event_Desc, count(distinct Citizen_ER)	Mil_Sv_Event_Cod = '8'	Citizen_ER, Dt_Event, Order_Mil_Org_Reg, Mil_Reg_Acron, Mil_Org_Cod, Mil_Org_Name, Mil_Sv_Event_Cod, Mil_Sv_Event_Desc	Order_Mil_Org_Reg, Mil_Org_Cod, Dt_Event
Q6	Citizen_Event_Fact	Citizen_Dim, Mil_Wish_Serve_Dim, Time_Dim, JSM_Dim	CSM_Acron, State_Acr, Mil_Wish_Serve_Desc, count(distinct Citizen_ER), Year_Ref , Mil_Sv_Event_Cod, Mil_Wish_Serve_Cod, Order_Mil_Reg	Year_Event >= 2011, Mil_Sv_Event_Cod = '1', State_Acr ='SC', Mil_Wish_Serve_Cod ='5', Order_Mil_Reg = '5'	CSM_Acron, Mil_Wish_Serve_Desc	CSM_Acron, Mil_Wish_Serve_Desc
Q7	Citizen_Event_Fact	Time_Dim,Citizen_Dim, JSM_Dim, Geography_Dim, Spec_Rank_Grad_Dim, Reserve_Class_Dim, Mobiliz_Dest_Dim	Mobiliz_Dest_Cod, Mobiliz_Dest_Desc, Qualif_Cod, Mil_Org_Cod, Mil_Org_Name, Reserve_Class_Cod, Reserve_Class_Desc, Rank_Grad_Cod, Rank_Grad_Desc, Year_Event, Order_Mil_Org_Reg, Mil_Reg_Acron, JSM_Desc, count(distinct Citizen_ER)	Year_Event >= '2011', Qualif_Cod in ('T22J', 'T23J', 'T24J', 'T25J', 'T26J', 'T27J', 'T28J', 'T29J', 'T30J', 'T31J', 'T32J', 'T33J', 'T34J', 'T35J', 'T36J', 'T37J', 'T38J', 'T39J', 'T40J', 'T41J', 'T42J', 'T43J', 'T44J', 'T45J', 'T46J', 'T47J', 'T48J', 'T49J', 'T50J', 'T51J', 'T52J', 'T53J', 'T54J', 'T55J', 'T56J', 'T57J', 'T58J', 'T59J', 'T60J', 'T61J', 'T62J', 'T63J', 'T64J', 'T65J', 'T66J', 'T67J', 'T68J', 'T69J', 'T70J', 'T71J', 'T72J', 'T73J', 'T74J', 'T75J', 'T76J', 'T77J', 'T78J', 'T79J'), Order_Mil_Org_Reg = '11'	Mobiliz_Dest_Cod, Mobiliz_Dest_Desc, Qualif_Cod, Mil_Org_Cod, Mil_Org_Name, Reserve_Class_Cod, Reserve_Class_Desc, Rank_Grad_Cod, Rank_Grad_Desc, Year_Event, Order_Mil_Org_Reg, Mil_Reg_Acron, JSM_Desc	-
Q8	Citizen_Event_Fact	Time_Dim, Citizen_Dim, Geography_Dim, Armed_Force_Dim, Mil_Wish_Serve_Dim, Mobiliz_Dest_Dim, Resid_Zone_Dim, Reserve_Class_Dim, Behavior_Dim, Mil_Sit_Dim, Mil_Sv_Event_Dim, Occupation_Dim	Year_Event, Mil_Org_Acron, Order_Mil_Org_Reg, Mil_Reg_Acron, Mil_Sit_Desc, Armed_Force_Desc, Mil_Wish_Serve_Desc, Behavior_Desc, Occupation_Desc, Year_Vinc, Reserve_Class_Desc, Mobiliz_Dest_Desc, Mil_Sv_Event_Desc, Resid_Zone_Desc, Mil_Sv_Event_Cod	Year_Event >= '2011', Mil_Sv_Event_Cod = '42', Order_Mil_Org_Reg in (1,3,5,12)	-	-

Table IV. Loading time of the FDT with CSV data

Years	Hadoop	Spark
A1 (2011)	1,464	956
A5 (2011 to 2015)	8,715	2,735
A8 (2011 to 2018)	16,721	5,748

and a 500 GB primary disk of HDD type. Finally, each worker node in this cluster has two CPUs with 7.5 GB of memory and a primary disk with 300 GB of HDD disk type.

Four scenarios were compared, and all used the five-node cluster. In scenarios one and two (C1 and C2), data processing was performed on disk by Apache Hadoop, while in scenarios three and four (C3

Table V. Cardinalities of the attributes involved

Attribute	Cardinality
Citizen_Age	76
CSM_Acron	34
JSM_Desc	5423
Mil_Org_Cod	3869
Mil_Org_Name	3869
Mil_Reg_Acron	11
Mil_Sv_Event_Cod	66
Mil_Sv_Event_Desc	66
Mil_Wish_Serve_Desc	7
Mobiliz_Dest_Cod	10
Mobiliz_Dest_Desc	10
Order_Mil_Org_Reg	14
PostPon_Sit	23
Qualif_Cod	215
Rank_Grad_Cod	47
Rank_Grad_Desc	47
Reserve_Class_Cod	7
Reserve_Class_Desc	7
Year_Event	1 to 8
Year_Ref	1 to 8

and C4), the same was performed by Apache Spark. In scenarios one and three (C1 and C3), the dimensional model and the fully denormalized table were compared and the data were stored as CSV. Scenarios two and four (C2 and C4) were as C1 and C3, but using tables stored as Parquet. Table VII show the scenarios' characteristics with its respective name.

5. RESULTS

All results are shown in Tables from VIII to XV. Each table shows the performance of a specific query over the years, from A1 to A8, comparing the model, table file format, and execution engine.

The first scenario evaluated was C1 using the years A1 to A8, and the results are shown according to each query performed. In the graph of query Q1, shown in Figure 4, the performance of the fully denormalized table (represented in dark blue color) was so superior to that of the dimensional model

Table VI. Lines analyzed by period

Years	Reference	Number of Lines of the Fact Table	Number of Lines Returned							
			Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
2011	A1	3,645,403	1	16	13	127	28	1	24	195
2011 a 2012	A2	9,458,748	1	16	26	308	80	1	43	380
2011 a 2013	A3	14,621,794	1	27	39	532	138	1	69	626
2011 a 2014	A4	18,206,124	1	16	52	899	185	1	133	847
2011 a 2015	A5	22,191,384	1	31	65	1468	226	1	162	1040
2011 a 2016	A6	26,059,430	1	32	77	4475	277	1	200	1262
2011 a 2017	A7	30,515,025	1	48	90	4766	2660	1	231	1504
2011 a 2018	A8	35,073,178	1	57	103	5296	7666	1	254	1619

Table VII. Scenarios' characteristics

Data format/Execution Engine	Haddop	Spark
CSV	C1	C3
Parquet	C2	C4

(represented in light blue color), which is not visible in the graph. It was also possible to observe that the query time in the fully denormalized model did not vary much as the number of rows grew, that is, over the years. This behavior occurred due to an existing parameter in Hadoop Hive called hive.fetch.task.*, which enables Fetch Task instead of MapReduce Job for simple queries, including select count (*) from the table. This task is very efficient and makes Hive access the file directly to get the result, making the query time much shorter, since it avoids the overhead of starting a MapReduce job. When running the query, Hive compares the table size under analysis with a threshold value. If the threshold value is higher than the table size, Hive will execute the fetch task; otherwise, it will execute the MapReduce job. As the table of the fully denormalized model is larger than the FACT table, the behavior should be the same for both models, but there is another factor that influences the performance of the query, which is the way the table is created. Although both tables present data in

Table VIII. Results of query Q1

Years	Hadoop - C1/C2				Spark - C3/C4			
	CSV		Parquet		CSV		Parquet	
	DM	FDT	DM	FDT	DM	FDT	DM	FDT
A1	32.12	0.085	0.112	0.080	8.83	5.85	0.078	0.356
A2	43.03	0.073	0.064	0.072	16.37	12.92	0.084	0.400
A3	43.16	0.077	0.060	0.067	25.89	19.47	0.096	0.588
A4	41.52	0.088	0.086	0.083	31.25	34.46	0.113	0.804
A5	57.28	0.081	0.074	0.078	40.76	67.02	0.113	0.811
A6	57.47	0.104	0.065	0.076	49.81	86.42	0.128	0.890
A7	58.79	0.097	0.064	0.071	56.18	157.48	0.135	0.954
A8	60.13	0.112	0.071	0.076	105.86	161.13	0.140	1.019

Table IX. Results of query Q2

Years	Hadoop - C1/C2				Spark - C3/C4			
	CSV		Parquet		CSV		Parquet	
	DM	FDT	DM	FDT	DM	FDT	DM	FDT
A1	32.52	20.40	1.56	3.24	7.35	10.86	3.78	4.93
A2	57.76	49.73	2.88	5.39	15.27	37.31	6.21	9.59
A3	58.65	70.01	7.88	5.55	22.41	65.16	15.80	14.78
A4	60.46	121.60	3.59	6.50	29.05	78.83	13.84	17.22
A5	78.27	119.07	7.62	8.30	34.15	95.27	11.20	20.06
A6	78.78	125.28	16.09	9.26	39.27	166.06	11.92	24.98
A7	76.24	155.88	17.81	11.69	53.69	145.16	19.25	22.32
A8	79.05	162.83	16.18	19.77	75.44	177.54	9.77	24.10

Table X. Results of query Q3

Years	Hadoop - C1/C2				Spark - C3/C4			
	CSV		Parquet		CSV		Parquet	
	DM	FDT	DM	FDT	DM	FDT	DM	FDT
A1	39.09	10.25	13.59	16.90	5.89	9.63	5.03	3.96
A2	48.30	55.66	11.31	12.85	10.86	35.83	4.08	7.23
A3	53.00	84.60	16.08	15.42	15.87	63.32	5.15	14.66
A4	62.62	91.81	15.87	16.09	20.80	79.75	8.09	12.12
A5	77.05	111.72	16.08	23.10	22.82	111.96	4.42	13.93
A6	68.18	134.58	13.80	19.66	27.42	170.23	6.25	18.35
A7	72.42	150.45	13.57	13.37	36.44	201.95	5.45	21.80
A8	78.64	162.76	10.74	15.82	40.24	235.65	3.73	21.47

Table XI. Results of query Q4

Years	Hadoop - C1/C2				Spark - C3/C4			
	CSV		Parquet		CSV		Parquet	
	DM	FDT	DM	FDT	DM	FDT	DM	FDT
A1	35.50	16.84	6.03	4.48	9.30	19.68	4.91	5.60
A2	57.98	47.03	5.88	4.27	20.66	82.50	3.41	9.26
A3	56.06	72.53	7.78	5.96	29.58	126.57	3.02	13.74
A4	51.20	204.39	6.54	8.12	38.26	156.29	2.33	17.03
A5	63.69	108.06	13.86	14.31	45.41	190.75	2.27	21.70
A6	58.91	130.96	17.50	11.53	57.60	237.77	2.42	24.70
A7	66.05	145.99	17.69	13.74	73.85	277.03	4.52	30.53
A8	68.11	254.62	16.81	16.45	80.29	306.39	6.74	36.64

Table XII. Results of query Q5

Years	Hadoop - C1/C2				Spark - C3/C4			
	CSV		Parquet		CSV		Parquet	
	DM	FDT	DM	FDT	DM	FDT	DM	FDT
A1	62.20	9.31	41.03	5.98	8.83	5.85	6.53	0.73
A2	75.69	46.89	36.58	13.88	16.37	12.92	1.93	0.88
A3	97.94	82.31	39.44	13.02	25.89	19.47	1.88	0.79
A4	97.98	93.53	40.25	12.08	31.25	34.46	2.17	0.88
A5	123.61	105.53	47.96	21.51	40.76	67.02	2.28	1.05
A6	147.91	124.96	50.65	21.11	49.81	86.42	7.68	1.07
A7	137.12	149.95	51.96	27.06	56.18	157.48	6.62	2.83
A8	172.77	168.15	56.75	28.83	105.86	161.13	6.06	4.15

Table XIII. Results of query Q6

Years	Hadoop - C1/C2				Spark - C3/C4			
	CSV		Parquet		CSV		Parquet	
	DM	FDT	DM	FDT	DM	FDT	DM	FDT
A1	57.87	18.43	32.95	15.75	12.42	9.06	4.77	4.73
A2	86.40	46.61	36.79	9.43	21.23	35.00	4.90	7.40
A3	105.29	72.05	46.36	10.56	30.90	63.44	7.24	12.10
A4	109.10	90.29	42.34	15.93	42.99	80.45	6.46	13.82
A5	126.76	113.15	50.55	18.27	47.11	95.66	9.09	17.95
A6	128.70	115.46	57.40	20.32	51.98	150.44	8.15	21.23
A7	142.97	143.30	64.72	17.50	84.33	146.65	12.48	29.51
A8	159.66	161.20	59.18	22.52	78.28	263.10	9.08	26.23

Table XIV. Results of query Q7

Years	Hadoop - C1/C2				Spark - C3/C4			
	CSV		Parquet		CSV		Parquet	
	DM	FDT	DM	FDT	DM	FDT	DM	FDT
A1	81.22	19.19	23.27	17.97	13.17	8.98	3.92	3.98
A2	92.11	52.22	22.38	18.87	24.61	34.58	5.74	6.52
A3	142.30	82.42	22.65	14.22	35.27	62.59	5.82	10.99
A4	148.40	97.31	23.56	18.74	47.88	47.85	6.78	14.87
A5	164.38	126.09	21.77	20.89	56.48	100.53	13.90	17.58
A6	198.60	123.86	29.71	24.69	62.03	124.70	14.55	21.55
A7	209.18	158.57	29.05	20.78	96.08	131.64	13.11	22.73
A8	244.53	172.77	35.05	27.58	146.20	226.34	21.20	27.70

Table XV. Results of query Q8

Years	Hadoop - C1/C2				Spark - C3/C4			
	CSV		Parquet		CSV		Parquet	
	DM	FDT	DM	FDT	DM	FDT	DM	FDT
A1	86.30	14.11	42.23	2.65	14.26	9.46	17.72	5.16
A2	110.43	32.28	62.70	13.49	25.42	41.57	61.64	8.53
A3	132.05	82.74	72.70	6.63	36.93	74.62	11.13	12.62
A4	142.13	95.79	78.12	10.35	44.06	79.68	9.99	15.27
A5	157.42	110.76	87.24	18.51	53.13	96.44	10.97	18.94
A6	153.32	156.77	90.33	21.60	65.00	130.84	13.22	20.23
A7	158.24	151.61	96.32	22.86	75.77	150.50	15.42	23.10
A8	182.54	172.16	95.76	26.77	91.74	158.32	17.79	26.56

CSV format, they were created in different ways. Therefore, when Hive executes the query execution plan of the fact table, it cannot access the size of the table to compare with the threshold; thus, it switches to the MapReduce job. Regarding the fully denormalized table, as it was created differently, its size is accessible by Hive; hence, the Fetch Task is chosen in the query execution plan. For query Q5, as can be seen in Figure 5, the performance of the fully denormalized table was slightly higher in almost all simulations, being exceeded only in A7.

In the C2 scenario, data processing was performed by Apache Hadoop (Tez), and the data was stored in Parquet format. The results of queries Q1 and Q5 are shown in Figures 6 and 7, in an exemplary way. According to Figure 6, in C2, the Q1 query execution times for the dimensional model and the fully denormalized table were very similar for most years, except for A1. This behavior was quite different from the result in C1, where the fully denormalized table presented a result much superior to the dimensional model. For example, in query Q5, as can be seen in Figure 7, the performance of the fully denormalized table was better in all simulations, different from that observed in scenario C1. Compared to C1, the behavior profile is similar, but the differences in time measurements are much greater. Analyzing EXPLAIN, it was found that in Parquet queries, in all scenarios, Hive performs a scan in the in-memory index, which justifies its superior performance compared to CSV queries.

In the C3 scenario, data processing was performed by Apache Spark, and the data was stored in CSV format. According to Figure 8, in C3, the execution times of the query Q1 for the dimensional model and the fully denormalized table were similar only up to A3; however, as the data volume grew, the dimensional model showed a much better performance than the fully denormalized table. The difference in results between the HadoopxSpark processing for this query is clarified when performing the EXPLAIN, which shows us that the query plan in Spark is different from the one performed in Hadoop and, by default, it does not perform the fetch task. In query Q5, as shown in Figure 9, a behavior similar to query Q1 can be observed, where the performance of the dimensional model and the fully denormalized table were similar up to A4; from that point, with the increase of the

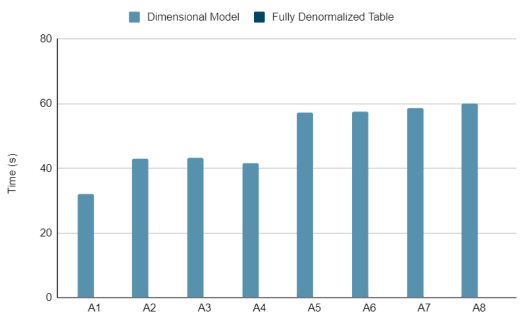


Fig. 4. Results of query Q1 in C1.

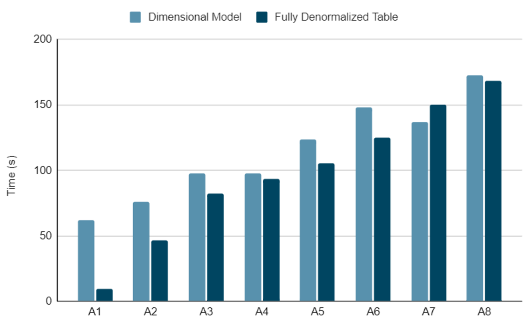


Fig. 5. Results of query Q5 in C1.

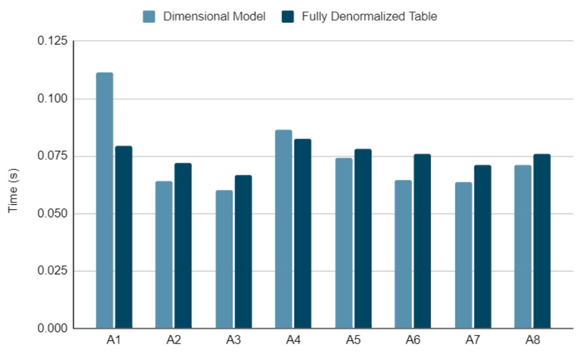


Fig. 6. Results of query Q1 in C2.

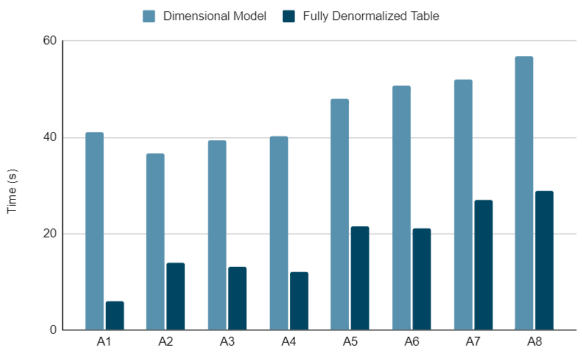


Fig. 7. Results of query Q5 in C2.

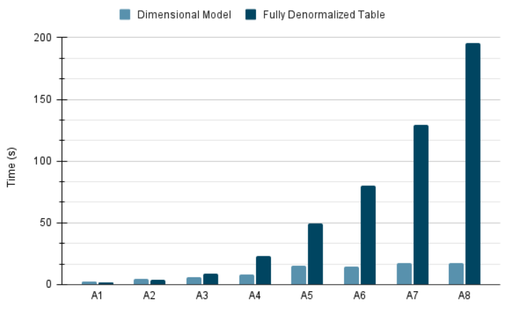


Fig. 8. Results of query Q1 in C3.

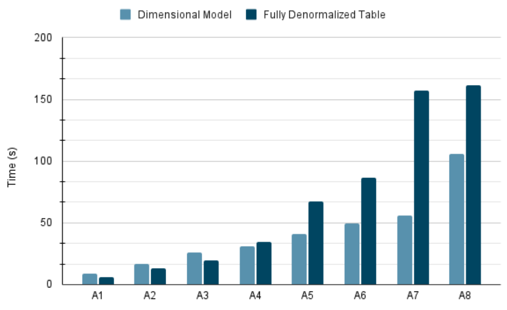


Fig. 9. Results of query Q5 in C3.

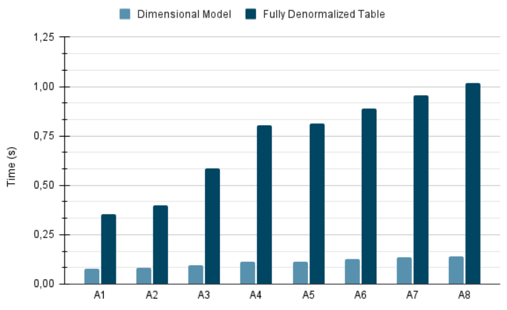


Fig. 10. Results of query Q1 in C4.

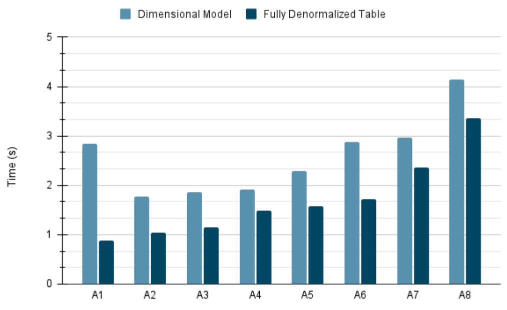


Fig. 11. Results of query Q5 in C4.

data volume for processing, the performance of the dimensional model surpassed that of the fully denormalized table.

In the C4 scenario, data processing was performed by Apache Spark, and the data stored in Parquet format. According to Figure 10, in C4, the execution times of the query Q1 for the dimensional model were much better than those of the fully denormalized table, in all simulations. In query Q5, as shown in Figure 11, it can be seen that the execution times of the two models increased with the addition of rows in the fact table; however, in all simulations, the fully denormalized table obtained superior performance. The other queries followed the profile of query Q1, showing better results for the dimensional model.

Comparison of C1 and C2 – Hadoop - CSV x Parquet: In general, analyzing the execution time of all queries in scenarios C1 and C2, it was verified that the performance of C2 was superior to that of C1. When analyzing the performance of the fully denormalized table alone, C2's results are even better, in terms of query execution time. This is also due to the poor performance of the dimensional model with CSV.

Comparison of C3 and C4 - Spark - CSV x Parquet:

Similarly, the performance of C4, in terms of runtime, was also much better than that of C3, being the biggest gain in the fully denormalized table. An overall decrease in the execution times of all queries can be observed when comparing only the CSV and Parquet formats when performing data processing in Spark.

Comparison of C1 and C3 – Hadoop x Spark both with CSV format:

Analyzing all queries, it was observed that the DM performance improved with Spark processing for queries: Q1, Q5, Q6, Q7, and Q8. FDT, on the other hand, in queries Q2, Q3, Q6, Q7, and Q8, improved performance with Spark until year A5 and worsened as the volume increased.

Comparison of C2 and C4 - Hadoop x Spark both with Parquet format:

Comparing Scenarios C2 and C4, the observations were as follows. For query Q1, scenario C2 (Hadoop + Parquet) outperforms C4, with a query performance greater than 5 times better than C4. When analyzing for example, the FDT separately, this value is greater than 10.

Analyzing all queries, for Parquet format, it was observed that the DM performance improved with Spark processing for queries: Q3, Q4, Q5, Q6, Q7, and Q8. Query Q2 improved its performance on Spark through year A6. On the other hand, the FDT obtained a performance improvement with all volume loads for Q5, a worsening in the other three (Q1, Q2, and Q4) and in the remaining ones, it obtained a result that oscillated with the data volume.

6. DISCUSSION OF RESULTS

First Analysis: Scenarios C1 and C2, using Hadoop

From the performed simulations, it is possible to infer that, for this dataset, running the simulations on Hadoop, the Parquet format favored the FDT. Therefore, in general, on Hadoop, if we consider the average of query times for all years in each scenario, the Parquet format improves query performance. This was explained in Section 5.

Second Analysis: Scenarios C3 and C4, using Spark

For data processing performed in Spark, DM functionedperformed better than the FDT table in CSV format, for most queries, and in Parquet format for almost all queries, except for Q5.

We can infer that Spark in-memory processing favored the DM. A possible explanation would be the much smaller size of the tables executed in the dimensional model in relation to the size of the fully denormalized table, which encompasses all DW tables. That happens because Spark is an execution engine that works both in memory and on disk, so Spark performs operations on disk when the data no longer fits in memory. Then, the big size of the fully denormalized table (which encompasses all DW tables) harmed the performance of this model in relation to the dimensional model. On the other hand, the dimensional model has a much smaller size, relative only to the accessed tables, favoring its complete processing in memory. This behavior of the FDT can also be related to the DW data, and a problem known as Stragglers Tasks. A straggler Task takes an exceptionally high time for completion as compared to the median or average time taken by other tasks belonging to the same stage. In this case, the FDT has a very large size and is not partitioned; thus, the partition by equal sizes associated with the non-uniformity of the distribution of the DW data ends up causing this abnormal increase in time in the processing of the queries, because as the partition of the data directly reflects the data to be processed, the skewed partitions result in a skewed distribution of computation time among tasks assigned for them.

These results complement our previous conclusion in [de Oliveira et al. 2021]: there is no best scenario in general, but for a group of specific queries, as well as data processing tools. Thus, it is necessary to group similar queries and suit not only the data format but the data processing tool as well. For example, it was noted that simple queries with aggregation functions perform better in Hadoop, because of the fetch tasks; however, when it is processed in Spark this optimization thus does not occur by default. Observing the different profiles of queries and their performance on Hadoop and Spark, it is possible to perceive that, in general, the FDT performs better on Hadoop and the DM on Spark.

Another important factor is the long loading time and table size of the FDT model, because to adopt the respective model, there can be no limitations regarding time and especially the storage space of the DW.

If we compare only the Parquet vs. CSV data formats, the Parquet format always performs better. There are two explanations for this superior result, one is relative to the in-memory index used by Hive to scan the data and the other is the nature of the Parquet format, which is a binary columnar format and depending on the query, it will only bring up the columns that were selected in the projection. Also, the compression factor of the Parquet will act so that these columns take up less space (less network traffic, possibility of caching in memory). Therefore, the Parquet format decreases the volume of data to be sent to the cluster to be processed, helping to improve its performance.

From the modeling point of view, in view of the behavior of the two models, it was clear that the choice of model to be employed should be made mainly in conjunction with the queries to be performed and the execution engine to be used.

7. CONCLUSION AND FUTURE WORK

This work presented a performance analysis between DW queries using different models, data formats and execution engines (Hadoop and Spark). Unlike other published works, which only compare the performance of queries for some file formats or the combination of file formats and execution engines, this work presented how the data modeling can interfere in the performance of certain queries, varying the file format and execution engine. Thus, it was possible to observe that independent of the model, Parquet file format made all queries faster, improving their performance significantly, both with Hadoop's disk processing and Spark's in-memory processing. However, when comparing the performance between models, it was observed that the choice of the execution engine has a great influence on the result and should be chosen according to the model used.

Therefore, in Hadoop processing, with the implemented queries, it is generally observed that the FDT outperformed the DM. In Spark processing, the result was the opposite, the DM outperformed the FDT. This work showed the importance of managing the sinergy that exists between these three factors as data modeling, data format and execution engine and the importance of adjusting them to actual existing requirements.

The heterogeneous profile of the queries strengthened the results, showing a trend of better performance of the DM in Spark and the FDT in Hadoop. In future works, there is the possibility to carry out an analysis of the impact of the cache in the queries executed successively and verify the impact of table partitions on the performance of the queries for the two models in question, dimensional and denormalized.

REFERENCES

Cassavia, N., Dicosta, P., Masciari, E., and Saccà, D. Data preparation for tourist data big data warehousing. In *International Conference on Data Management Technologies and Applications*. INSTICC, SciTePress, pp. 419–426, 2014.

Chambers, B. and Zaharia, M. *Spark: The definitive guide.* O'Reilly, 2018.

Costa, E., Costa, C., and Santos, M. Y. Efficient big data modelling and organization for hadoop hive-based data warehouses. In *European, Mediterranean and Middle Eastern Conference on Information Systems*, M. Themistocleous and V. Morabito (Eds.). Springer International Publishing, pp. 3–16, 2017.

de Oliveira, B., Valente, A., Victorino, M., Ribeiro, E., and Holanda, M. Análise da influência da modelagem e formato de dados no desempenho de data warehouse baseado em hadoop-hive. In *Anais do XXXVI Simpósio Brasileiro de Bancos de Dados (SBBD)*. SBC, pp. 271–276, 2021.

Di Tria, F., Lefons, E., and Tangorra, F. Design process for big data warehouses. In *International Conference on Data Science and Advanced Analytics (DSAA)*. pp. 512–518, 2014.

Inmon, W. H. *Building the Data Warehouse.* Wiley, 2005.

Jacobs, A. The pathologies of big data. *Comm. of the ACM* 52 (8): 36–44, 2009.

Luckow, A., Kennedy, K., Manhardt, F., Djerekarov, E., Vorster, B., and Apon, A. Automotive big data: Applications, workloads and infrastructures. *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, 2015.

Mohanty, S., Jagadeesh, M., and Srivatsa, H. *Big data Imperatives: Enterprise Big Data Warehouse, BI Implementations and Analytics.* Apress, 2013.

Plase, D., Niedrite, L., and Taranovs, R. A Comparison of HDFS Compact Data Formats: Avro Versus Parquet. *Mokslas - Lietuvos ateitis* 9 (3): 267–276, 2017.

Rodrigues, M., Santos, M. Y., and Bernardino, J. Big data processing tools: An experimental performance evaluation. *WIREs Data Mining and Knowledge Discovery* 9 (2): e1297, 2019.

Sandoval, L. J. Design of business intelligence applications using big data technology. In *2015 IEEE Thirty Fifth Central American and Panama Convention (CONCAPAN XXXV)*. pp. 1–6, 2015.

Santos, M. Y. and Costa, C. Data warehousing in big data: From multidimensional to tabular data models. In *Ninth International C* Conference on Computer Science Software Engineering.* ACM, pp. 51–60, 2016.

Vajk, T., Fehér, P., Fekete, K., and Charaf, H. Denormalizing data into schema-free databases. In *2013 IEEE 4th International Conference on Cognitive Infocommunications (CogInfoCom)*. pp. 747–752, 2013.

Weintraub, G., Gudes, E., and Dolev, S. Needle in a haystack queries in cloud data lakes. In *EDBT/ICDT Workshops.* CEUR-WS.org, 2021.

White, T. *Hadoop: The definitive guide.* O'Reilly, 2015.