**Journal Pre-proof**

Beyond TPC-DS, a benchmark for Big Data OLAP systems (BDOLAP-Bench)

Roberto Tardío, Alejandro Maté, Juan Trujillo

PII: S0167-739X(22)00055-3
DOI: https://doi.org/10.1016/j.future.2022.02.015
Reference: FUTURE 6406

To appear in: *Future Generation Computer Systems*

Received date : 1 July 2021
Revised date : 11 February 2022
Accepted date : 17 February 2022

Please cite this article as: R. Tardío, A. Maté and J. Trujillo, Beyond TPC-DS, a benchmark for Big Data OLAP systems (BDOLAP-Bench), *Future Generation Computer Systems* (2022), doi: https://doi.org/10.1016/j.future.2022.02.015.

# Beyond TPC-DS, a Benchmark for Big Data OLAP Systems (BDOLAP-Bench)

Roberto Tardío[a,*], Alejandro Maté[b], Juan Trujillo[a,b]

[a] StrateBI Business Solutions Ltd, Madrid, 28020, Spain
[b] Dept. of Software and Computing Systems, Lucentia Lab, University of Alicante, Alicante, 03690, Spain

**Abstract**

Online Analytical Processing (OLAP) systems with Big Data support allow storing tables of up to tens of billions of rows or terabytes of data. At the same time, these tools allow the execution of analytical queries with interactive response times, thus making them suitable for the implementation of Business Intelligence applications. However, since there can be significant differences in query and data loading performance between current Big Data OLAP tools, it is worthwhile to evaluate and compare them using a benchmark. But we identified that none of the existing approaches are really suitable for this type of system. To address this, in this research we propose a new benchmark specifically designed for Big Data OLAP systems and based on the widely adopted TPC-DS benchmark. To overcome TPC-DS inadequacy, we propose (i) a set of transformations to support the implementation of its sales data mart on any current Big Data OLAP system, (ii) a choice of 16 genuine OLAP queries, and (iii) an improved data maintenance performance metric. Moreover, we validated our benchmark through its implementation on four representative systems.

*Keywords:* Big Data OLAP, Benchmarking, Data Modelling, Kylin, Druid

## 1. Introduction

Over the past decade, Online Analytical Processing (OLAP) systems have evolved their architecture and functionality to support Big Data scenarios [1, 2, 3]. Several Big Data OLAP approaches have emerged [4], highlighting open-source tools with Standard Query Language (SQL) support, such as Kylin [5], Druid [6], Pinot or Clikhouse. These systems enable the storage of analytical data models comprising tables of up to tens of billions of rows and terabytes of data while allowing interactive query latencies (milliseconds-seconds). This very high performance when executing analytical queries is the main feature that differentiates them from general-purpose Big Data tools with SQL support, such as Hive, Spark SQL or Drill.

When choosing a Big Data OLAP system, two of the key features to consider are (i) the performance in executing analytical queries to maintain user interactivity in applications that require OLAP support (e.g., dashboards, reports or multidimensional tables), and (ii) system efficiency when loading and updating data. In order to allow for an objective performance comparison between current Big Data OLAP systems, the most appropriate is to apply an industry-standardized benchmark. There are several approaches for benchmarking Data Warehouses and OLAP systems [7, 8, 9, 10, 11, 12, 13], being the TPC-H [9], SBB [10] and TPC-DS [8, 7] benchmarks some of the most recognized and widely accepted by the industry.

However, there is no benchmark that addresses the specific nature of current Big Data OLAP systems. The different architectures and constraints posed by Big Data OLAP systems to support extreme data scenarios, challenge or even prevent the application of existing benchmarks, which are only suitable to be used with Data Warehouses (DW) or traditional OLAP systems. This leads many vendors and practitioners [13, 14, 15] to partially apply benchmarks such as TPC-DS or by modifying their definition, without applying the same criteria in each implementation. This leads to biased results that do not allow for an objective comparison of Big Data OLAP systems from different vendors.

We agree with other authors [16] that one of the main challenges is the lack of flexibility of current benchmarks to implement their proposed data model in the system under test. We consider that the implementation of the data model proposed by the benchmark should not be unique for all systems, since each Big Data OLAP approach proposes a different architecture. While some Big Data OLAP systems [5] are optimized to work with star [1] or snowflake schemas composed of several tables and relationships, others [6] are designed to work with simpler data structures, even with single-table data models.

In order to address the issues identified in current benchmarking approaches, we propose a new benchmark for Big Data OLAP systems. Our approach is based on the TPC-DS benchmark, which has been validated by industry for benchmarking DW systems and even for some general-purpose Big Data SQL systems, such as Hive [17]. However, the inadequacy of its (i) data model, (ii) proposed query set, and (iii) performance metrics to the nature of current Big Data OLAP systems, prevent its implementation on this kind of systems. To overcome these shortcomings, we propose (i) a set of transformations to enable the optimal implementation of its sales data model in any Big Data OLAP system, (ii) a selection of 16 genuine OLAP queries, and (iii) an improved data maintenance performance

*Corresponding author.
*Email address:* roberto.tardio@stratebi.com (Roberto Tardío)

measure.

In order to demonstrate the application of our proposed benchmark, we have chosen Apache Kylin [5] and Apache Druid [6] for its implementation, two modern and representative[1] Big Data OLAP systems [18, 19, 20, 21, 22]. We have chosen these tools as they both promise similar query performance for Big Data scenarios while presenting a well-differentiated architecture that allows us to properly evaluate our proposed methods. Furthermore, in order to evaluate the performance advantages promised by these two Big Data tools specialized in OLAP workloads over general-purpose Big Data SQL systems, we have also implemented our benchmark on Apache Hive (with LLAP) [17] and Spark SQL.

In the next section, we analyze the current benchmarking proposals and their application to Big Data OLAP systems. Then, in section 3, we present our benchmark for Big Data OLAP systems and propose some guidelines for its application. In section 4, we apply our proposed benchmark to the four proposed systems. Then, in Section 5, we analyze the results and then discuss the applicability of our benchmark proposal in Section 6. Finally, in Section 7, we present our conclusions as well as future research directions.

## 2. Related work

Among the benchmarking approaches for OLAP systems, we can highlight the ones standardized by the TPC organization (TPC-H, TPC-DS and TPCx-BB) [7, 8, 9, 11, 12, 13] and the SSB (Star Schema Benchmark) [10]. For all these benchmarks, we have found and analyzed use cases of their application [23, 18, 16, 24, 19, 14, 15, 17] to Big Data OLAP systems.

One of the most widely adopted benchmarks is TPC-H [9]. Created in an era before Big Data systems, due to its workload simplicity, this benchmark is still one of the most used benchmarks today. Despite the advantages of TPC-H, we agree with other authors [10, 18] that the proposed third normal form (3FN) normalized data model, typically used in transactional systems, is not suitable for analytical systems. For the implementation of Data Warehouses (DW) and OLAP systems, denormalized snowflake or star schemas [1] are widely used by the industry due to their performance benefits and ease of maintenance.

Rodrigues et al. [24] have applied the TPC-H benchmark on several representative general-purpose Big Data SQL tools: Presto, Impala, Hive on Tez, Spark SQL, Drill and HAWQ. They examine the set of queries proposed by TPC-H to identify the challenges these queries pose to DW and OLAP systems. Random row retrieval without applying data aggregation, and the use of complex expressions or sub-queries are some of the challenges identified. Despite the advantages of this research, the suitability of using TPC-H for benchmarking Big Data OLAP systems is not analyzed.

The SSB (Star Schema Benchmark) [10] benchmark was designed to make the TPC-H benchmark more suitable for OLAP systems. This benchmark proposes a star schema based on the TPC-H data model and a new set of 13 OLAP queries, excluding TPC-H queries that include subqueries or self-joins since they were considered inappropriate for an OLAP system. Thanks to its ease of use, SSB is being applied by the industry for benchmarking Big Data OLAP systems, such as Kylin[2] or Druid[3], as well as in several researches [19, 18, 16]. However, despite its advantages, the SSB benchmark presents a too simple workload for use on Big Data systems and also a less complete definition compared to other existing benchmarks, such as TPC-DS [13].

In [19] the authors apply the SSB benchmark on Apache Druid [6], testing different implementations of the data model proposed by SSB. Since Druid does not support join operations, they require a denormalized implementation of this data model. In [18], the same authors apply the SSB benchmark to compare Apache Druid against Apache Hive and Presto, implementing for each tool the optimal SSB data model design they identified in their previous research [16, 19]. This fact highlights the need for benchmarks to enable the adjustment of their data model design to the specific nature of the Big Data OLAP system under test. Furthermore, in [25] the authors highlight the need to adapt data models to the nature of each specific Big Data database due to the constraints imposed by their complex architectures and also the usefulness of data modeling techniques to perform such adaptation. However, all these studies [19, 18, 16] do not analyze the suitability of applying the SSB benchmark to Big Data OLAP systems.

The TPC-DS benchmark [7, 8, 13], developed between 2006 and 2012, is also one of the most adopted benchmarks today. This benchmark proposes a snowflake data model, suitable [1] for DW and OLAP systems, and a workload of higher complexity than the other benchmarks analyzed. This workload consists of a set of 99 data queries and a set of metrics for the performance measurement to enable the objective comparison of current systems. Noteworthy, its proposed data generator [26] relies on the use of real patterns, achieving a more realistic distribution of the test data than in the other benchmarks discussed. In addition, in [13] the second version of TPC-DS is presented, proposing several adjustments to support its use with Big Data SQL systems, such as Apache Hive, Impala or Presto. Among the proposed modifications, we highlight the relaxation of its implementation restrictions, the rewriting of some queries and the elimination of row-level update operations.

There are many studies where applications of the TPC-DS benchmark are presented [13, 14, 15, 17]. In [15] TPC-DS is applied to benchmark Drill, HAWQ, Hive, Impala, Presto and Spark SQL. Despite the completeness of this study, the partial implementation of TPC-DS with only 16 queries is not justified and the feasibility of implementing TPC-DS on Big Data SQL tools is not questioned. In [14] Hive, Spark SQL, Impala and Presto systems are benchmarked using TPC-DS, TPC-H and TPCx-BB. An important insight of this study are the limitations identified that prevent the implementation of all 99 TPC-DS

[1] http://kylin.apache.org/community/poweredby.html
https://druid.apache.org/druid-powered

[2] https://github.com/Kyligence/ssb-kylin

[3] https://blog.cloudera.com/sub-second-analytics-hive-druid/

queries in the evaluated tools, but again the suitability of TPC-DS to benchmark these systems is not questioned. In [17] the new features of Apache Hive 3.1 for interactive query execution are presented. Although they used TPC-DS for benchmarking Hive 3.1 and Hive 1.2.1, to evaluate the integration between Hive 3.1 and Druid they chose SSB benchmarking instead of TPC-DS and also denormalised the multi-table star schema into a single table. Despite the advantages of this study, the rationale for using SSB (denormalised) instead of TPC-DS when evaluating the integration between Hive and Druid is not discussed. In [13], the new version of TPC-DS is used to compare two unidentified Big Data SQL systems against two more traditional non-Big Data systems. However, the feasibility of applying TPC-DS to new Big Data systems focused on OLAP support has not been studied.

In 2016, the TPCx-BB benchmark was released [12, 11, 27]. Unlike TPC-H and TPC-DS, this benchmark adds support for several types of processing: OLAP, raw data exploration (semi and unstructured) and machine learning processes. Despite its advantages, due to its focus on general-purpose Big Data systems, the TPCx-BB benchmark is only partially suitable for benchmarking Big Data OLAP tools.

After our analysis of the existing approaches for benchmarking Big Data OLAP systems, we identified two key issues that affect all of them, as listed below:

- The proposed data models [7, 8, 10, 9, 11, 12, 13] present suitability issues for their use in Big Data OLAP systems.
- Some of the proposed queries [7, 8, 9, 11, 12, 13] are not truly OLAP queries. They are more appropriate for benchmarking general-purpose DW or even transactional OLTP systems.

The issues identified are leading Big Data OLAP system vendors to apply benchmarks, such as TPC-DS, in a partial or altered form [14, 15], or even to choose [24, 17] simpler but less complete benchmarks, such as TPC-H or SSB. As a result, published performance reports do not allow for an objective comparison of existing Big Data OLAP systems, often favor the technology of the vendor presenting these reports.

Notice that in our previous work [4] we analyzed the challenges for benchmarking Big Data OLAP systems. As a result, we identified the key characteristics of the OLAP workloads and their related performance metrics that should be evaluated for benchmarking Big Data OLAP tools. However, unlike our new proposal based on TPC-DS, the benchmark proposed in [4] focuses on the comparison of different implementations of the same data model in one chosen Big Data OLAP tool rather than on the performance comparison between different tools. On the other hand, we believe that using TPC-DS as the starting point for the design of our new benchmark provides a higher reliability than designing it from scratch, as TPC-DS has been validated by industry and the scientific community for its use with traditional DW systems or even with some Big Data SQL systems such as Hive.

For all these reasons, we identified the need to develop a specific benchmark that allows the objective comparison of performance between current Big Data OLAP systems. This benchmark must consider the specifics of this kind of systems and enable their implementation in any of them. To this end, in this research we present a new benchmark approach that fulfills the above goals.

## 3. Proposed benchmark

We conclude that the TPC-DS benchmark is the most complete and realistic of all the benchmarks reviewed, allowing benchmarking traditional DW systems and even some Big Data SQL systems with huge volumes of data. However, the partial implementations of the TPC-DS benchmark analysed in the state-of-the-art [14, 15] show that there are still Big Data SQL tools that do not support the implementation of its complex data model and all of its 99 queries.

Furthermore, our proposal focuses on benchmarking a specific kind of Big Data SQL tools specialized in supporting OLAP workloads [4, 28], i.e., supporting the execution of analytical summary queries with interactive response over huge volumes of data. As shown in [14, 15, 17], the response of general-purpose Big Data SQL tools such as Hive or Presto is not interactive in most cases. However, Big Data tools specifically designed to optimize OLAP workloads have emerged, such as Apache Kylin or Druid, which promise sub-second query execution times. In order to achieve this extreme performance, these tools further restrict SQL support for highly complex analytical queries, i.e. queries with a high level of detail, involving a very large number of result rows or a high number of attributes. These constraints prevent the execution of TPC-DS on Big Data OLAP systems. However, highly complex OLAP queries are not commonly required for the implementation of dashboards or OLAP tables. Big Data OLAP systems are therefore the only alternative for the implementation of such applications in Big Data scenarios.

Similarly to how TPC-DS adapted its specification [13] to support general purpose Big Data SQL systems, Big Data OLAP systems deserve to be benchmarked in an objective way. To this end, in this research we propose a new benchmark for Big Data OLAP systems based on TPC-DS. Our benchmark is composed of the following elements:

- **Data model:** We propose to use only the data mart representing sales (store, web and catalog) in the original data model proposed by TPC-DS.
- **Rules for data modelling:** To support the optimal implementation of this data model in any Big Data OLAP system, we propose three transformations that can be optionally applied to obtain equivalent data models. This is one of the main novelties of our proposal compared to existing approaches, in which the structural design of the benchmark data model is pre-fixed, preventing its adaptation to the nature of the systems to be tested.
- **A set of 16 OLAP queries:** From the 99 queries proposed by TPC-DS, we selected 16 queries for our benchmark, based on a set of criteria that we defined to discard those

queries that do not really represent an OLAP type query or those that involve operations constrained by most of the current Big Data OLAP systems.

- **A data generator:** We adopted the synthetic data generator proposed by TPC-DS, DSDGEN (aka MUDD) [26], which allows to set the size of the test data using the Scale Factor (SF) parameter.
- **Performance tests and metrics:** We propose to perform individual performance tests of (i) data querying, with one or more concurrent users, (ii) initial data loading, and (iii) data refreshing. For this data maintenance test, we propose changes to both the process implementation and metrics originally proposed by TPC-DS.

For the design of each of these elements, we have taken into account the particularities of current Big Data OLAP systems, discussed in the following subsection. Subsequently, we describe with more detail each of the elements that are part of our benchmark proposal.

### *3.1. Modern big data OLAP systems*

To support OLAP applications on Big Data scenarios, several approaches such as Apache Kylin [5], Apache Druid [6], Pinot or Clickhouse have emerged in recent years. These systems enable the storage of analytical data models in tables of up to tens of billions of rows and terabytes of data while maintaining interactive query latencies (milliseconds-seconds). This extreme performance in the execution of OLAP workloads is what differentiates these tools from other general-purpose Big Data SQL tools such as Apache Hive, Spark SQL or Presto whose execution times [14, 15, 17, 24] are of a few seconds at best and often up to tens of seconds.

To achieve this very high performance, these tools rely primarily on distributed processing and storage architectures, such as those of Hadoop, as well on the use of aggressive data indexing, pre-combination or pre-aggregation techniques. As a trade-off, these architectures and techniques impose some notable constraints and differences in the way we should use them, especially if we compare these approaches against traditional non-Big Data OLAP systems. To a lesser extent, such constraints can also be found in general-purpose Big Data SQL systems [13]. Below, we enumerate the main constraints imposed by current Big Data OLAP systems:

- **Data model:**Most of these systems impose constraints to the implementation of relationships and join operations between the different fact and dimension tables within an analytical data model. There are even architectures that only allow the use of a single table, thus all data from the fact and dimensions tables has to be joined and stored in this table.
- **Data querying:** These systems mostly support the use of standard query language (SQL). However, they are often partially compliant with the standards (ANSI SQL), focusing on support for OLAP queries that use mainly clauses and operators for data aggregation and filtering. For instance, in some cases these systems do not support data queries that do not apply aggregation, some special operators, such as standard deviation, or even advanced OLAP operations, such as OVER or RANK.
- **Data loading and refreshing:** To maintain high data loading and query performance, these systems often impose the need to refresh data blocks (aka segments) instead of allowing row-level updates.

These constraints imposed by current Big Data OLAP systems to support extreme data scenarios, make unfeasible the application of some of the existing benchmarks such as TPC-DS. Therefore, our benchmark for Big Data OLAP systems must be flexible in its implementation, taking into account the different architectures and functionality of current Big Data OLAP approaches. To this end, in the following sections, we describe the methods proposed to enable that our benchmark can be applied to any current Big Data OLAP system.

### *3.2. Data model*

The data model is one of the key components of our benchmark. It is based on the TPC-DS, but using only its sales data mart (DM). To enable the implementation of this DM in any current Big Data OLAP system, we propose three transformations to it that can be optionally applied.

The original data model proposed by TPC-DS uses a snowflake schema [1], generally suitable for implementation in DW and traditional non-Big Data OLAP systems. This data model represents a retail scenario, composed of 3 sub-models or DMs: sales, returns and inventory. In addition, the sales and returns DMs consist of 3 sales channels: store, catalog and internet. Thus, each of the sales and returns DMs is implemented using 3 fact tables, one per channel, and a set of dimension tables that are mostly shared between the fact tables. Fig. 1 shows an excerpt of the entire schema depicting the store channel in both the sales and returns DMs. For the entire schema, please refer to the v3.0 public release [29].

As we can see, there are two fact tables per channel: one to store product sales (store_sales) and a second one to store potential product returns (store_returns). In these fact tables, the minimum granularity is given by the products and the sales ticket, being able to relate both DMs through the sales ticket number. At the same time, each of these fact tables is related to a set of dimension tables, which are represented in Fig. 1 by the remaining entities, such as the date or customer dimensions. There is also an inventory fact table that only contains data for the catalog and internet channels. Altogether, the TPC-DS data model comprises 24 tables, including 7 fact tables and 17 dimension tables.

However, the constraints imposed by the architectures of current Big Data OLAP systems make it difficult or even prevent the implementation of the TPC-DS data model. To address this issue, we can apply certain techniques such as data denormalization [1] to obtain equivalent data models, i.e., containing exactly the same data and relationships as the original one. Thus,

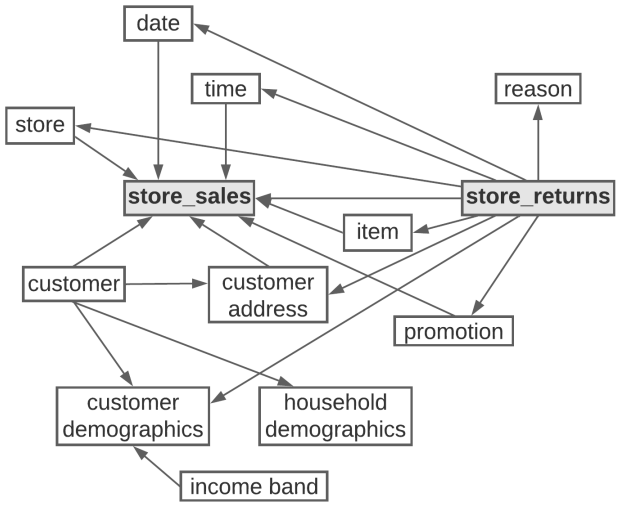



**Fig. 1.** Data mart model for the store channel in TPC-DS benchmark.

by applying these techniques, the benchmark data model can be adapted to the nature of each Big Data OLAP system, overcoming any constraints imposed by these systems. To this end, we propose three transformations that can be applied to the sales data mart (DM) to generate an equivalent data model that allows us to implement our benchmark with the system under test. The application of the three proposed transformations is optional, encouraging a previous study of the architecture and constraints of the system to be tested in order to decide which of these transformations to apply. In the following, we present the 3 proposed transformations, described using set and relational algebra:

- **T1 - Unification of fact tables across the sales channel:** To avoid the execution of UNION operations in queries involving two or more data mart channels, i.e., queries between fact tables. Since DM channels share most of the metrics stored in the fact table columns and also their related dimension tables, we can model the channel as a new degenerate dimension, i.e., stored in a new column on the fact table.

  Let be the fact tables relative to the 3 sales channels: $STORE\_SALES$, $WEB\_SALES$ y $CATALOG\_SALES$. Let $COL_{STORE}$, $COL_{WEB}$ y $COL_{CATALOG}$ respectively be the sets of columns which compose such fact tables. We can then obtain the columns not common to each of these fact tables.

  If we define the set of not common columns to the $STORE\_SALES$ table as $COL_{STORE\_diff} = (COL_{WEB} \setminus COL_{STORE}) \cup (COL_{CATALOG} \setminus COL_{STORE})$, thus the resulting table can be defined as $UNCOMMON_{STORE} = (col_{STORE\_diff\_1}, col_{STORE\_diff\_2}, \ldots, col_{STORE\_diff\_n})$. At this point, we can model the channel dimension as a table $CHANNEL_{STORE} = (channel)$ where $channel \in ('store')$. Then, we can define the fact table $STORE_{CN}$ as a combination of tables: $[STORE_{CN} = ]\Pi_{col1,col2,\ldots,coln}(STORE\_SALES \times UNCOMMON_{STORE} \times CHANNEL_{STORE})$.

  We repeat the above steps to obtain the $WEB_{CN}$ and $CATALOG_{CN}$ tables. Finally, to build the unified fact table by the sales channel dimension, we can apply the operation proposed in Equation 1.

$$[SALES =] \quad STORE_{CN} \cup WEB_{CN} \cup CATALOG_{CN} \tag{1}$$

- **T2 - Denormalization of dimensions composed of more than one table:** In the data model proposed by TPC-DS a dimension can be composed of several tables. For example, in Fig. 1 the dimension representing the customer is composed of the customer table, which is related to the fact table store_sales, but also to three dimension tables: customer_address, household_demographics and customer_demographics. Considering that some Big Data OLAP systems constrain the use of dimensions composed of multiple tables, in these cases we can apply denormalization techniques [1] to reduce the number of dimension tables and thus reduce the number of relationships between tables.

  $A = (\underline{col_1}, col_2, \ldots, col_{n\_b}, col_n)$ and $B = (\underline{col_1}, col_2, \ldots, col_n)$ are two dimension tables composed of $\underline{col_1}$ primary key columns, $col_n$ columns representing the attributes of the dimension instances, and $col_{n\_b}$ foreign key columns in table A that allow to relate this table to B through its primary key. These columns hold values for each row (aka tuples) that compose the table. In these cases, we can apply denormalization between A and B tables, to reduce them to a single table C. In Equation 2, we define this denormalization operation using relational algebra ($|x$ means left join):

$$[C =]A \quad |x_{A.col_{n\_b}=B.\underline{col_1}} \quad B \tag{2}$$

  This transformation can be applied recursively until all dimensions composed of more than one table are reduced into a single table. As a result, we will convert the original snowflake schema into a pure star schema, more suitable for OLAP systems [10, 1, 4].

- **T3 - Full denormalization:** Some Big Data OLAP systems further constrain the complexity of the data models they can implement by not allowing any JOIN operations between tables. In these cases, the only solution is the complete denormalization of all tables into a single big table that combines all facts and dimensions.

  Assuming the fact table $SALES = (\underline{col_{1\_1}}, \underline{col_{1\_2}}, \underline{col_{1\_n}}, \ldots, col_n)$ obtained by the application of T1, and all the dimension tables $DIM_i = (\underline{col_1}, col_2, \ldots, col_n)$ denormalized by the application of T2, the fully denormalized table $SALES_{DEN}$ can be obtained by the operation defined in Equation 3.

This operation combines the fact table SALES with all the previously denormalized dimension tables.

$$[SALES_{DEN} =] \quad (((SALES|x_{SALES.\underline{col_{1\_1}}=DIM_1.\underline{col_1}}DIM_1) |x_{SALES.\underline{col_{1\_2}}=DIM_2.\underline{col_1}}DIM_2)\ldots|x_{SALES.\underline{col_{1\_n}}=DIM_n.\underline{col_1}}DIM_n) \tag{3}$$

By the selective application of one or more of our proposed transformations, we can obtain a data model equivalent to the original one proposed by TPC-DS. Thus, enabling the implementation of the benchmark data model in any Big Data OLAP system. Moreover, these transformations can help optimize the data model to the nature of their particular architecture.

However, although the application of T1 allows us to reduce the number of fact tables from 7 (store_sales, store_returns, web_sales, web_returns, catalog_sales, catalog_returns, inventory) to 3 (sales, returns, inventory), there will still be queries involving UNION or JOIN operations between fact tables. These types of queries are not optimized or, in the worst case, supported by most current Big Data OLAP engines.

It is possible to apply denormalization between the sales and returns fact tables, to combine (JOIN) them in a single table using the ticket number column. Although this design is feasible, we must be aware that the return of a product does not usually occur at the same time of its purchase, but after a few days. However, since row-level updates are not supported in most Big Data OLAP systems or, if supported, random row updates are not recommended in the fact tables [1], if we apply the above solution, it will be necessary to update the entire fact table every time we add or refresh data in it.

The TPC-DS benchmark, in addition to testing the performance of data querying, also tests the performance of the initial (full) load and data maintenance. For these tests, TPC-DS supports the use of different data set sizes, also called Scale Factors (SF). Thus, if only full loads of the fact table can be performed, the data maintenance test would become meaningless. Moreover, the time required for a full fact table reloading, especially with high SF factors with up to hundreds of terabytes of data, could take up to several days. This fact would prevent the availability of recent data in the system, greatly reducing its usefulness due to the loss of value of the data stored on it for decision-making processes.

To overcome the above constraints and their implications, we propose to use only the sales DM in our benchmark. We consider that this DM, consisting of 3 channels (store, web and catalog) related to all dimension tables, represents a data model complex enough to evaluate Big Data OLAP systems. We believe that in these systems, rather than focusing on the structure of the supported data model, the focus of benchmarking should be on the volume of data or rows that these systems allow to be efficiently updated and queried. In fact, the sales DM is where the TPC-DS benchmark maintains most data, as sales is an event that occurs much more frequently than returns and also generates more data than updating the product inventory.

To exemplify the application of the proposed transformations, in Fig. 2 we show a valid data model generated by the application of T1 and T2 transformations. This model includes all the entities used in our benchmark.

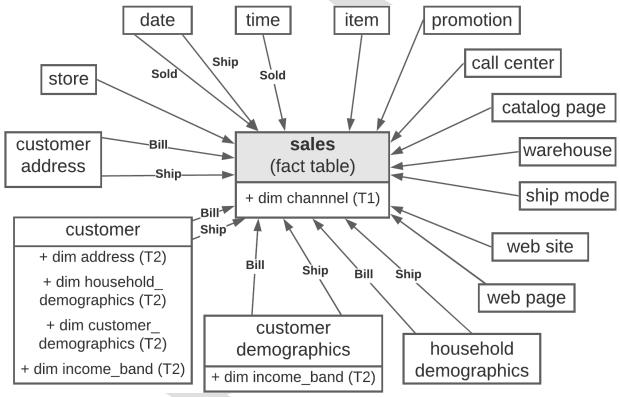



**Fig. 2.** Instance of the data model proposed for the benchmark, obtained by applying the T1 and T2 transformations.

### *3.3. Benchmark query set*

The TPC-DS benchmark proposes a set of 99 queries, in SQL language, which are classified into 4 types: Reporting, Ad-Hoc, OLAP and Data Mining. Based on their type, we might think that at least the queries classified as OLAP, or even reporting, are suitable for OLAP systems. However, the benchmark specification [29] does not indicate the type of each of the 99 queries, thus not allowing us to identify those queries that are truly suitable for benchmarking OLAP systems. Furthermore, we observe that many of the queries proposed by TPC-DS are highly complex OLAP queries. That is, queries with a high level of detail, involving a very high number of result rows, a high number of attributes or sub-queries with several levels of nesting. These queries are rare in OLAP systems, because the whole structure of the data model is designed to facilitate data aggregation and summarization operations.

To determine which TPC-DS queries are suitable for benchmarking Big Data OLAP systems and therefore should be part of our benchmark, we first analysed the query capabilities and constraints posed by the most current Big Data OLAP systems, as well as the characteristics of the OLAP queries that we identified in our previous research [4]. Based on the results, we have defined several criteria to discard those queries that are not suitable for benchmarking this kind of systems. Then, we have applied these criteria to review the 99 queries originally proposed by TPC-DS. As a result, we have selected the following 16 queries that compose our benchmark query set: 2, 3, 26, 33, 42, 43, 52, 55, 58, 58, 60, 62, 63, 76, 88, 90 and 95. We use the original TPC-DS numbering for referring to them.

In the following, we describe the process of defining these criteria. First, in Table 1 we have identified those types of queries and features that may pose significant issues for current Big Data OLAP systems. In addition, we have evaluated them for a Big Data OLAP system that is quite flexible in terms

**Table 1**
Capabilities analysis of two current Big Data OLAP systems

| Feature | Apache Kylin | Apache Druid |
|---|---|---|
| SQL syntax | Extensive, close to ANSI SQL:2003.<br>Not supporting ANSI:89 style joins | Limited, ANSI standard not indicated. |
| Join operations | Limited:<br>- Optimizes inner and left joins.<br>- Very low performance in all other join types. | Not supported. |
| Queries with low or no data aggregation | Limited, with poor performance | Fully Supported |
| Queries between dimension tables (without using fact table) | Limited, with poor performance | Not supported |
| Complex sub-queries | Limited (poor performance) | Limited (no joins supported) |
| Known unsupported functions and operators | Standard deviation | Rank and Over operators |

of data model complexity and supported SQL queries, Apache Kylin, against a system that is more restrictive in these aspects, Apache Druid. Note that some general-purpose Big Data SQL systems, Hive and Spark SQL, already support in their latest versions the implementation of the 99 queries proposed by TPC-DS. However, as these systems do not specifically aim to improve the performance of OLAP workloads, they have not been taken into account for the definition of our proposed query selection criteria.

The queries selected for our benchmark must be able to run on all current Big Data OLAP systems to enable the full implementation of our benchmark on any of them. From the constraints identified in Table 1, we next identify the criteria for discarding those TPC-DS queries that are not suitable for benchmarking Big Data OLAP systems and therefore should not be part of our benchmark:

- **No data aggregation:** The main goal Big Data OLAP systems is to boost the performance of data summarization queries at extreme data scenarios (Big Data). To this end, these systems constrain row-level data query features, or even prevent queries that do not use aggregation operators (e.g., SUM, MAX,...). Thus, we will discard all queries that do not use aggregation operators.

- **Joins and unions between fact tables:** This kind of operation is not optimized or even supported by current Big Data OLAP systems. Therefore, in the previous section, we decided to use just the sales DM and discard the use of returns and inventory DMs. We also proposed a transformation (T1) to unify the 3 channels of the sales DM into a single fact table, in order to avoid JOIN operations between the fact tables of these channels. By applying this criterion, we will discard all queries that involve the returns or inventory DMs.

- **Self joins:** Queries involving a join of a table with itself are not common in OLAP systems [10], being the execution performance of these queries very low in those Big Data OLAP systems that support them. For this reason, we will discard all queries that include self-join operations.

- **Analytical queries on dimensions:** Another identified query pattern that is rare in OLAP analysis is analytical queries across dimensions, without involving a fact table. These queries are not optimized or even supported in Big Data OLAP systems, so we must discard them.

- **Unsupported functions and operators:** Some functions and operators that are less frequently used in OLAP analysis, such as the standard deviation (STDEV) or the RANK and OVER operators, are not supported by all current Big Data OLAP systems. As our goal is to ensure that our benchmark is generally supported, we will remove all queries containing these unsupported operators.

- **Complex subqueries:** Since this type of queries is less common [10] in OLAP analysis, Big Data OLAP systems usually present limitations or performance issues when executing them. For this reason, we will discard all those queries that present one or more of the following subquery patterns: (i) join operations between a subquery and a parent query, (ii) more than one level of nesting, i.e., subqueries containing other subqueries or, (iii) subqueries that meet any of the other discard criteria (e.g., self join or no data aggregation). This also includes WITH expressions presenting any of the above circumstances.

However, we will consider all those queries that, even matching the criteria defined, can be rewritten to remove the problematic casuistry while maintaining their original statements, ensuring that the response data are exactly the same as those returned by the original query. This applies to queries affected by the data model unification and denormalization performed as a result of the application of one or more of the 3 proposed transformations.

For instance, in Fig. 3 we show query 60 in its original form (left), presenting a UNION ALL operation between fact tables, thus matching one of the discard criteria identified. However, by the application of T1 (and T2) we have obtained the data model shown in Fig. 2, where this UNION ALL operation is no longer necessary. Hence, we can rewrite the original query (left) to remove all instances of this operation, thus obtaining

an equivalent query (right) that returns exactly the same data as the original one.

```
with ss as (
select
  i_item_id,
  sum(ss_ext_sales_price) total_sales
from store_sales, date_dim,
   customer_address, item
where
   i_item_id in
            (select i_item_id
             from item
             where
               i_category in ('Jewelry'))
   and  ss_item_sk = i_item_sk
   and  ss_sold_date_sk = d_date_sk
   and  d_year  = 1998
   and  d_moy = 11
   and  ss_addr_sk = ca_address_sk
   and  ca_gmt_offset = -5
group by i_item_id),
 cs as (...
# Same as ss expression but changing
store_sales to catalog_sales.
...)
ws as (...
# Same as ss expression but changing
store_sales to web_sales.
...)
select  i_item_id,
        sum(total_sales) total_sales
from  (select * from ss
       union all select * from cs
       union all select * from ws) tmp1
group by i_item_id
order by i_item_id, total_sales
limit 100;
```

⇨

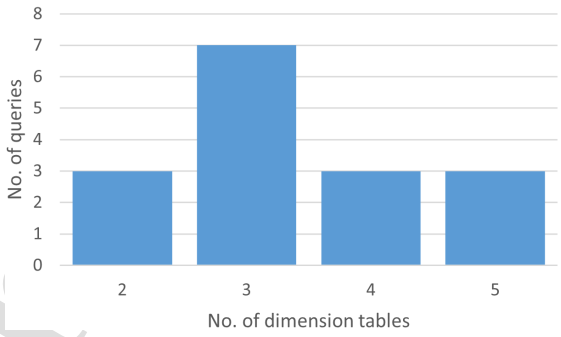
```
select
  i_item_id,
  sum(s_ext_sales_price) total_sales
from sales
  join date_dim on
     s_sold_date_sk = d_date_sk
  join customer_address on
     s_bill_addr_sk = ca_address_sk
  join item on
     s_item_sk = i_item_sk
where
    d_year=1998
    and d_moy=11
    and i_item_id in
              (select i_item_id
               from  item
               where
                 i_category in
('Jewelry'))
    and ca_gmt_offset=-5
group by i_item_id
order by i_item_id  , total_sales
limit 100;
```

**Fig. 3.** Query 60. On the left is shown the original version proposed by TPC-DS. Right is shown the query rewritten for the equivalent data model generated by application of the proposed transformations (T1-T3) and shown in Fig. 2.

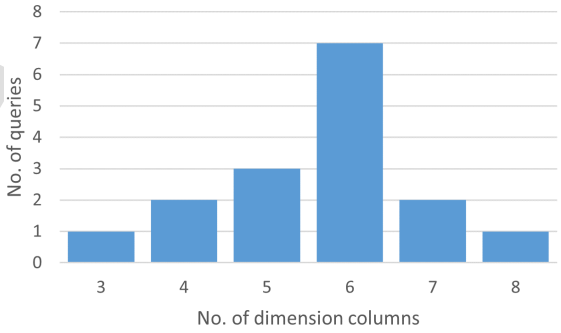
In Table 2, we show the 16 queries included in our benchmark, selected by applying the proposed discard criteria on the 99 original TPC-DS queries. In addition, we specify those queries that have been selected even not meeting some of the defined criteria. These are queries 63 and 95, which are especially representative for the benchmark due to their high computational requirements. To overcome all their matching discard criteria, we have applied minor modifications to these query statements. Even after this modification, these queries still represent a very high computational workload for the system to be tested.

There are some more of the 99 queries proposed by TPC-DS that might meet the criteria and therefore could be accepted. Nevertheless, we have performed an analysis of the variability and computational load represented by the 16 selected queries, concluding that this proposed set of queries is enough representative of a real Big Data OLAP workload. Moreover, as highlighted by some studies [13], it is not so important the number of queries, but rather the variety of queries included in the complete set allows for testing the different features provided by current Big Data OLAP systems.

In order to analyze the variability and computational load of our 16 proposed queries, we first studied how many different dimension tables and columns of these tables are involved in the proposed queries. The higher the number of different dimension tables and columns involved, the more JOIN, GROUP BY or WHERE filtering operations are needed, thus implying a higher computational load for the Big Data OLAP system tested. As we can observe in Fig. 4a, most of the selected queries make use of 3 or more dimensions. At the same time, in Fig. 4b we observe that 67% of the queries use at the same time 6 or more dimension attributes. These results show that most of the proposed queries pose a large computational demand on the Big Data OLAP system tested.




**(a)** Number of different dimension tables simultaneously used.




**(b)** Number of different dimension columns simultaneously used.

**Fig. 4.** Analysis of concurrent use of dimension tables and their columns in the proposed query set.

Another factor that has a significant impact on the computational load posed by the query is the number (cardinality) of distinct values in the dimension columns. High or very high cardinality values in columns used in GROUP BY clauses will considerably increase the number of combinations between facts and dimensions in queries. This problem is also known as the curse of dimensionality, whose effects on Big Data OLAP systems we analyzed in our previous research [4].

In [4], we differentiate between high cardinality (HC, >1,000 instances) and ultra-high cardinality (UHC, >300,000 instances) dimensions, which generate a very high computational load to the system. Applying this classification to our proposed queries, we observe that all of them use at least one HC dimension and two of them also use UHC dimensions. In addition,

**Table 2**
Set of 16 queries of our proposed benchmark, selected by applying the discard criteria defined on the 99 original queries proposed by TPC-DS.

| Query | Decision | Not met criteria | Changes | New statement |
|---|---|---|---|---|
| 2, 3, 26, 33, 42, 43, 52, 55, 58, 60, 62, 76, 88, 90 | Accepted | - | - | Unchanged |
| 63 | Accepted with changes | Unsupported operators | Suppressing OVER operation | For a given year calculate the ~~monthly~~ total and average sales of items of specific categories, classes and brands that were sold in stores and group the results by store manager. |
| 95 | Accepted with changes | Join between fact tables using a WITH expression | Suppression of the JOIN clause with the returns fact table | Produce a count of web sales and total shipping cost and net profit in a given 60-day period to customers in a given state from a named web site ~~for returned orders shipped from more than one warehouse.~~ |

there are 14 queries that use two or more HC dimensions, along with multiple low cardinality dimensions.

From the results of the analysis performed, we can conclude that the 16 OLAP queries selected thanks to the application of our defined criteria simulate a high computational load for the system to be tested. Therefore, they are suitable for benchmarking Big Data OLAP systems.

### *3.4. Load testing and data maintenance*

To test the Big Data OLAP system using the proposed queries and data model, we have to load into this data model a volume and variety of data enough to simulate a real Big Data scenario [2]. The TPC-DS benchmark proposes the use of the DSDGEN (aka MUDD) synthetic data generator [26], which applies real distribution patterns from the retail domain, such as sales seasonality. Furthermore, DSDGEN allows us to configure the data volume by means of a parameter named Scale Factor (SF), having as possible values multiples of 10 that correspond approximately to the volume of data generated in GB.

For the above reasons, we consider the use of the DSDGEN data generator suitable for our benchmark. We also consider appropriate the proposed SQL statements to implement the data maintenance test using the generated data. However, if we apply any of the proposed transformations (T1-T3) to adapt the data model design to the nature of the Big Data OLAP system under test, we will need an additional data transformation process to enable the test data loading into this system. For the same reasons, we will need to adapt the proposed SQL statements for the data maintenance processes.

In Fig. 5, we describe the processes and data flows that we propose to implement the initial data load (5a) and the data updates (5b). In both cases, the first step is the generation of data using DSDGEN, specifying the SF value and the type of load to be performed. If the load is of the initial type, all the historical data for the initial load test (5a) will be generated. If the load is of update type, only the data required for the data maintenance test (5a) will be generated. The times required for data generation using DSDGEN and for storing this data without any transformation in the target file system (e.g., HDFS) are not measured, thus they are not taken into account in the performance metrics.

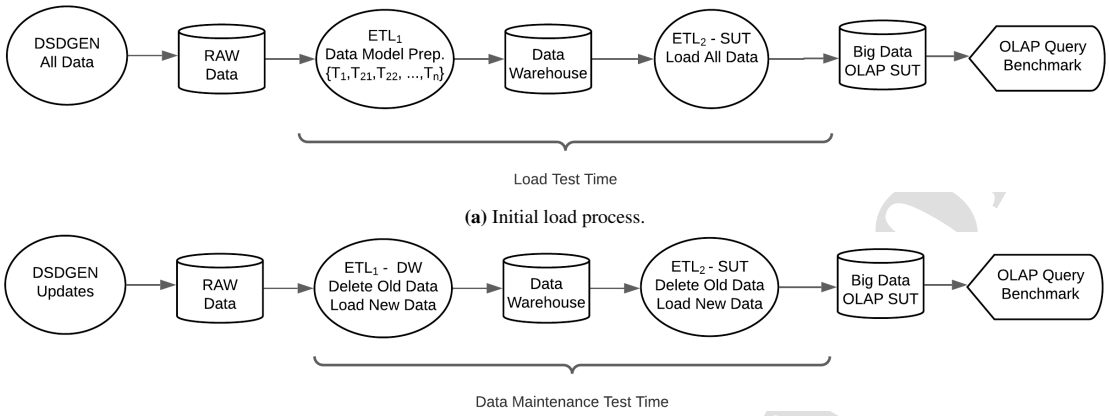
If we have applied any of the proposed transformations (T1-T3), to implement the initial data loading, it will be necessary to transform the data sets generated by DSDGEN to the new data model design. However, Big Data OLAP systems generally do not support the implementation of data transformation processes, e.g., using SQL statements or other programming languages. In these cases, it is necessary that the data extraction, transformation and loading process, $ETL_1$, be implemented and executed using an external tool. Also, it is necessary to store the resulting data in a Data Warehouse (DW) tool, that will be used as a data source of the Big Data OLAP system.

To implement these processes and the repository, we propose to use a Big Data SQL technology, such as Apache Hive [17], that (i) supports efficient distributed processing and storage, (ii) can be used as a data source for current Big Data OLAP systems, and (iii) facilitates the implementation of SQL data update statements. If such supplementary tools are needed, they will be part of the tested system along with the Big Data OLAP technology evaluated.

In addition to $ETL_1$, a second $ETL_2$ process is required to load all the historical data from the DW into the tested Big Data OLAP system. This process can be implemented using the data loading capabilities of the Big Data OLAP system itself.

For the implementation of the data maintenance data flow shown in Fig. 5b, the processes required are very similar to those required for the initial load. For each execution of the data update test, DSDGEN generates 3 data intervals that include 2 contiguous days. The data updates to be performed only affect the fact tables of the sales DM, including data from the store, web and catalog channels.

To implement the $ETL_1$ process for data maintenance, we must carry out the update of the data stored in the DW used as a staging repository. For the implementation of this process, we will have to adapt the SQL statements proposed by TPC-DS to fit the data model obtained by applying our proposed transformations (T1-T3). Again, Big Data DW tools such as




**(a)** Initial load process.

**(b)** Data maintenance process.

**Fig. 5.** Proposed processes and data flows to implement the initial load and data maintenance tests. The bracket below each image indicates those processes whose execution time has to be measured as part of our proposed benchmark tests.

Apache Hive facilitate the data update tasks due to their support for row-level updates [17].

In contrast, Big Data OLAP systems typically do not allow row-level updating, restricting updating processes to blocks of data grouped into temporary partitions (aka segments). The granularity or size of these segments is a user design decision that impacts data loading, update, and query performance. It is generally not desirable to maintain small data segments (e.g., at the hourly or customer level). However, working with larger data segments will often force us to refresh more data than we need to update.

Thus, the $ETL_2$ process required for data maintenance has to replace in the Big Data OLAP system those segments that match or otherwise contain the three 2-day intervals whose data have to be updated in that test run. For example, in Table 3, we show the correspondence between 3 data intervals generated by TPCDS-GEN (SF=10) for the sales fact table update test and the 3 possible data segments that include these data intervals in the Big Data OLAP system tested.

As in the initial load test, both data refresh processes, $ETL_1$ and $ETL_2$, are measured for the TPC-DS data integration test. The value of this test is calculated for TPC-DS using the metric $T_{DI} = T_{DataMaintenance1} + T_{DataMaintenance2}$ ,i.e., by the sum of two complete data maintenance tests. According to our definition of the data loading process shown in Fig. 5b, this data maintenance performance is calculated as $T_{DataMaintenance1} = ETL_1 + ETL_2$.

However, as shown in Table 3, some of the segments to be replaced in the Big Data OLAP system are much larger than the data sets whose update is required in the test. In these cases, the update processes will require much more computational resources, and therefore may considerably penalize the performance results obtained.

However, in a real data scenario, updates on more recent data periods in time are much more frequent than those affecting older data periods. To optimize these cases, Big OLAP systems allow defining smaller data segments for more recent data and larger ones for older data. By applying this strategy, we can improve the performance of most data update processes without degrading query performance, since the number of small segments does not increase significantly.

Considering the above, we propose to modify the $T_{DataMaintenance}$ metric originally defined by TCP-DS in order to give more weight to the performance of processes that affect more recent data periods versus those that affect older periods, since these segments are updated less frequently in a real data scenario. Below, we define the proposed weighting for the execution time of each segment according to the date range to this segment belongs to:

- $DR_1 \in [2002-12-01, 2002-12-31]$: Data updates that affect the last month of the historical data. We weight their execution time with 1 as they are the most frequent updates.
- $DR_2 \in [2002-07-01, 2002-11-30]$: Data updates affecting the 5 months prior to the most recent month. We

**Table 3**
Example of correspondence between the date ranges to be updated in the benchmark and the data segments stored in a Big Data OLAP system.

| TPC-DS GEN Data ranges | | | Segments a in big data OLAP system | | |
|---|---|---|---|---|---|
| Start date | End date | No. of rows | Start date | End date | No. of rows |
| 2002-11-12 | 2002-11-13 | 110,046 | 2002-11-12 | 2002-11-13 | 110,046 |
| 2000-05-20 | 2000-05-21 | 31,884 | 2000-01-01 | 2000-07-01 | 2,897,763 |
| 1999-09-18 | 1999-09-19 | 72,618 | 1999-07-01 | 2000-01-01 | 6,975,700 |

weight their execution time with 0.5 since they are updates that occur less frequently.

- $DR_1 \in [1998-01-01, 2002-06-31]$: Updates affecting the rest of the historical data, i.e., older time periods. We weight their execution time with 0.1 since they are very infrequent updates.

To apply the above proposed weighting, in Equation 4, we propose a new definition of the $T_{DataMaintenance}$ metric to benchmark the data update performance of Big Data OLAP systems.

$$T_{DataMaintenance} = ETL_1 + MAX(ETL_{2_{DR1}}, (ETL_{2_{DR2}} * 0.5), (ETL_{2_{DR3}} * 0.1)) \quad (4)$$

In Equation 4, the $ETL_1$ process will be executed in a DW system, hence not suffering the same constraints as the $ETL_2$ process, which will be executed in the same Big Data OLAP system where the data has to be updated. For this reason, the proposed weighting is applied only on the execution time obtained by each of the three sub-processes in $ETL_2$ that update the data for each of the 3 segments that involves the data maintenance test. Moreover, as Big Data OLAP systems allow the parallel execution of multiple segment update processes, the data update time will be given by the execution time of the slowest sub-process, being its execution time weighted according to the date range to which this segment belongs.

### *3.5. Guidelines for benchmark implementation*

To help with the application of our proposed benchmark, we define below the steps for its implementation in the Big Data OLAP system to be tested:

1. **Data model design review:** Starting from the definition of the TCP-DS sales Data Mart, the first step is to decide whether to apply one or more of the 3 proposed transformation rules (T1-T3). Thanks to their application, we will be able to obtain data models equivalent to the original sales DM in order to adapt its design to the architecture of the Big Data OLAP system to be tested.
2. **Rewriting of the proposed SQL queries:** Depending on the design of the data model generated in the previous step and the SQL syntax supported by the Big Data OLAP system under test, we will have to rewrite, to a greater or lesser extent, the 16 OLAP queries proposed in our benchmark. The rewritten queries must return exactly the same data as the original queries, never altering their statements.
3. **Implementation of data loading and updating processes:** To enable data loading and updating in the Big Data OLAP system under test, we have to implement the processes proposed in Fig. 5. To this end, it is necessary to adapt their functional logic to the data model design we obtained in the step 1. It is also highly recommended to analyze the features of the Big Data OLAP system to select the most appropriate technology for supporting the implementation of the $ETL_1$, $ETL_2$ processes and the staging Data Warehouse.
4. **Benchmark execution:** In order to simplify the application of the proposed benchmark, we propose several individual performance tests. These isolated tests allow a lower-level of comparison between Big Data OLAP systems compared to using the global performance metric proposed in TPC-DS [7]. We propose two classes of performance tests: (i) data querying, and (ii) data loading and updating.
   For query execution performance testing, we propose two tests. First, we will perform sequential execution of the 16 proposed queries by performing 10 complete test runs that randomly alter the order of query execution. Second, to test concurrency in the data querying, we will repeat the above test by simulating multiple users (e.g. 1, 5 and 10 simultaneous users), each executing in parallel their random sequences of the 16 benchmark queries.
   As for the data load and update performance tests, we will first run the initial load test for the different SFs selected. After that, we will run the data maintenance test, measuring the results with the proposed Equation 4.

## 4. Benchmark implementation

In order to validate our benchmarking proposal for Big Data OLAP systems, we have carried out its implementation on two current Big Data OLAP systems: Kylin and Druid. The selection of these systems has been made based on their popularity [18, 19, 20, 21, 22].

Furthermore, in order to study the advantages of these Big Data OLAP tools over general-purpose SQL Big Data systems, we have implemented our benchmark on Apache Hive and Spark.

### *4.1. Apache Kylin*

One of the most mature and powerful Big Data OLAP technologies is Apache Kylin [5], whose architecture we reviewed with greater detail in our previous research [4]. To support sub-second query latencies on data models with tables of up to billions of rows, Kylin relies on the intensive use of data pre-aggregation techniques. By applying such techniques to the data sources, it generates a pre-aggregated data structure named OLAP cube that can be queried with the standard query language (SQL). In addition, Kylin fully supports star and snowflake schemas, the de facto standard for Data Warehouse design [1] as a source for most OLAP applications.

Of the multiple options supported by the Kylin architecture, we will use the following for the implementation of our benchmark:

- Apache Hive as the source Data Warehouse. Thanks to its power and functionality, it allows us to implement both DW storage and the $ETL_1$ process described in Fig. 5.

- Map Reduce as the cube building engine, i.e. for the implementation of the $ETL_2$ process using the Kylin API.

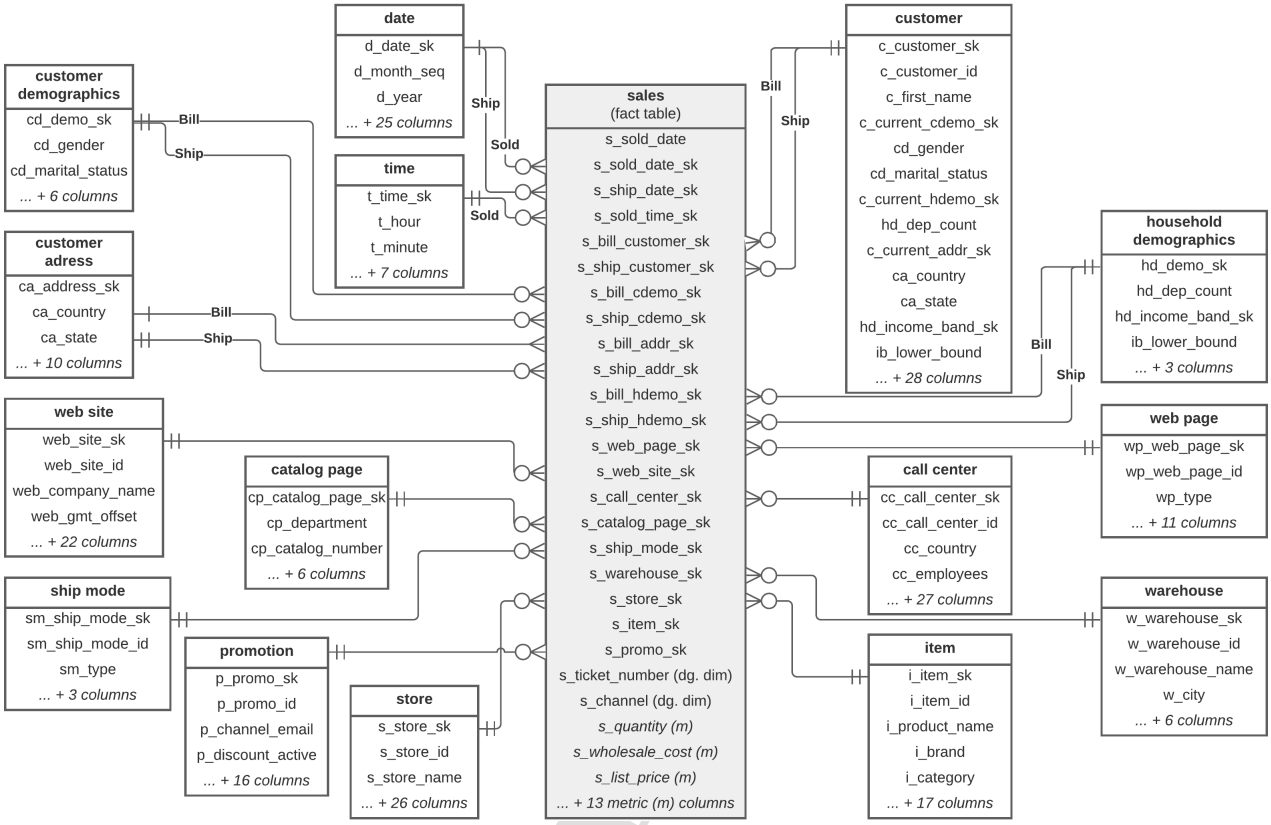



**Fig. 6.** Data model for the implementation of our benchmark in Kylin. Obtained by applying the T1 and T2 transformations.

Once the Kylin architecture was instantiated for the test, we applied the proposed transformation rules to obtain the data model shown in Fig. 6. To obtain it, we first applied T1 to unify the fact tables of the 3 sales channels. We made this decision taking into account that Kylin does not optimize the JOIN operation between fact tables, severely penalizing the performance of this type of queries, even making their execution unfeasible.

Second, we have applied the T2 transformation to denormalize all dimensions composed of more than one table, such as Customer and Customer Demographics, converting the Snowflake model into a pure Star Schema. Although Kylin supports the Snowflake model and therefore it is not mandatory to apply T2, as we discussed in [4] dimensions composed of more than one dimension table will affect the complexity of the resulting OLAP cube and thus penalize the performance of the data loading and updating processes.

Finally, we have not considered it convenient to apply the T3 transformation. As we also pointed out in [4], if we denormalize all the tables of the model into a single fact table with all dimension columns included, Kylin requires to apply pre-aggregation on all dimension columns (attributes), greatly increasing the complexity of the data loading or updating processes or even making their execution unfeasible.

Once we have obtained a data model design suitable for its implementation in Kylin, we have to adapt the 16 proposed OLAP queries to this data model and also to the SQL syntax supported by Kylin. The following is a list of the main types of modifications we carried out in the structure of the proposed queries:

- Rewrite of JOIN operations to the ANSI:92 standard, since Kylin does not support the older ANSI:89 standard used by TPC-DS.
- Removal of JOIN operations on queries that use denormalized dimensions. (e.g. Customer).
- Removal of UNION operations between fact tables of one of the 3 sales channels (store, web or catalog). As these fact tables been unified through a new channel dimension by the application of T1, these UNION operations are removed from the original query or replaced by WHERE filtering operations on the channel.
- Because Kylin does not support row-level data query operations, aggregation operations (e.g. sum(col1-col2)) that operate on non-aggregated columns require the inclusion of these columns in the GROUP BY clause. This involves the addition of a subquery, which uses a higher-level query to apply the final aggregation required. Examples of this are the queries 43 and 62.

Finally, it is necessary to define a segment strategy to load the data into the cube. This strategy will impact the performance of the data maintenance processes. To maintain high query performance, Kylin recommends storing small data segments only for recent data. These small, recent segments should be compacted with the other, large, historical data segments as time progresses. Considering this, we define the following segment strategy: 1-day segments $\in [2002-12-01, 2002-12-31]$ , 2-day segments $\in [2002-11-01, 2002-12-01]$ , 15-day segments $\in [2002-07-01, 2002-11-01]$ and 6-month segments $\in [1998-01-01, 2002-07-01]$.

### *4.2. Apache Druid*

Apache Druid [6] is another of the most consolidated Big Data OLAP systems. As Apache Kylin, it proposes a distributed, and therefore horizontally scalable architecture that supports sub-second queries on tables that store up to tens of billions of rows. To achieve this high performance, Druid's architecture is based on the intensive use of indexing techniques and, optionally, also supports data pre-aggregation by means of roll-up operations that can be applied at data ingest. Unlike Kylin, pre-aggregation is not mandatory in Druid, therefore this feature plays a less important role in its architecture.

Druid also supports the SQL language, with some syntax differences compared to Kylin. But regarding data model, Druid does not support JOIN operations between tables, thus requiring the implementation of fully denormalized data models in a single large table.

Similar to Kylin, Druid does not support row-level data updates. However, it does support seamless integration with Apache Hive, allowing data to be loaded and updated in Druid in a simple and efficient way by defining SQL processes in Hive.

Of the multiple options supported by the Druid architecture, we will use the following for the implementation of our benchmark:

- Apache Hive as source DW and for the implementation of the $ETL_1$ and $ETL_2$ batch processes described in Fig. 5.
- Hive (on Apache Tez) for loading and updating data into Druid, i.e., for the implementation of the $ETL_2$ process.
- Druid columnar storage using HDFS as deep storage of the OLAP cube.
- Execution of SQL queries through the Druid JDBC driver.

Once the Druid architecture was instantiated for benchmark implementation, we applied the three proposed transformation rules (T1,T2 and T3) in order to obtain a suitable data model. To obtain it, we have to apply T1 and T2 in the same way as we did to obtain the Kylin model shown in Fig. 6. However, since Druid does not support the JOIN operation between tables, we must additionally apply the T3 transformation on this model to perform a complete denormalization. As a result, we generated a single large table consisting of 402 columns, including all the columns of the dimensions tables and all the metrics of the sales fact table.

At this point, we have to adapt the 16 proposed OLAP queries to our data model design and the SQL syntax supported by Druid. The following is a list of the main changes made to the shape of the benchmark queries:

- Removal of all JOIN operations between the fact and dimension tables, as these relationships are already implicitly included in the sales fact table after the denormalization performed.
- Addition of a WHERE filter by the (degenerate) channel dimension in the fact table, for all those queries that originally queried independently the sales fact table of one of the 3 channels (store, web or catalog).
- To obtain the best performance, in those queries that require filtering by columns of the dimension (sold) date, we replace the original filters (e.g. sdd_moy=11) by the TIME_EXTRACT(__time, <unit>) function on the __time column (e.g. TIME_EXTRACT(__time, 'MONTH')=11).

Finally, it is also necessary to define a strategy for the size or number of data segments. The Druid architecture recommends maintaining segment sizes between 300MB and 700MB, therefore, unlike Kylin, it does not benefit from the compaction of historical data into very large segments. This is an advantage of Druid over Kylin, as we can maintain smaller segments (e.g. days to months segments) for the entire historical data set. Thanks to this, data maintenance processes in Druid can involve updating less data than in Kylin, thus being much more efficient approach. Considering this, we decided to implement a strategy of daily segments for the whole data set.

### *4.3. General-purpose Big Data SQL tools: Apache Hive and Spark SQL*

Hive is a general-purpose Big Data SQL tool in Hadoop, which allows the execution of OLAP queries, detail queries, data exploration, table creation or the implementation of data transformation processes. In our benchmark implementation we have used 3.1 version using a configuration similar to the one used in the TPC-DS test presented in [17]: applying LLAP (Live Long and Process) to support interactive response, the new transactional tables and the ORC file format with Snappy compression.

Apache Spark is even more flexible than Hive, supporting all kinds of SQL query processing, data transformation, machine learning or graph processing, even in real time. Spark has its own distributed processing engine that differs from Tez (used by Hive) in that it loads all the data to be processed into the RAM memory of the cluster nodes, instead of storing any data on disk. To test Spark with our proposed benchmark, we decided to run it using the Spark Thrift J/ODBC service, which allows Spark to work as an in-memory SQL database server. As for the file format, we also used ORC with Snappy compression.

One advantage of these two general-purpose tools is that they do not require the application of any of the 3 proposed transformations (T1-T3) for the implementation of our benchmark. However, in order to compare them with Kylin and Druid as

objectively as possible, we have decided to use the same data model in both tools. This data model is the one obtained for the implementation of the Kylin version of our benchmark, shown in Fig. 6.

### *4.4. Hardware and configuration of the performance tests*

For the benchmark execution, we have used a Hadoop cluster with the Hortonworks Data Platform (HDP) 3.1 distribution. This cluster comprises 4 nodes, each with 8 vCores and 32 GB of RAM, where we have installed Kylin (2.6), Druid (0.12.1), Hive (3.1, on Apache Tez with LLAP) and Spark (2.3.2). As for the TPC-DS data generator (DSDGEN), we have used an implementation adapted to run on Hadoop [30].

For our test set, we have chosen three different Scale Factors (SF): 10, 30, and 50. In Table 4, we show the volume in GB of the test data generated and stored in HDFS. As we can see, the SF is close to the volume in GB of the generated data. In contrast, the volume of data generated for the data maintenance test is much smaller, but enough suitable to test these systems.

**Table 4**
Size in GB of the data generated by DSDGEN for our experimentation.

| | SF 10 | SF 30 | SF 50 |
|---|---|---|---|
| Historical data | 11.4 | 26.5 | 44.4 |
| Update data | 0.17 | 0.14 | 0.22 |

However, it is not only relevant the size in GB of the data to be loaded, but also the number of rows stored in the fact and dimension tables. The greater the number of rows, the greater the computational needs to execute the join, aggregation or filtering operations involved in our proposed OLAP queries. Big Data OLAP systems promise sub-second performance with tables of up to tens of billions of rows, thus it is necessary to generate a volume of rows high enough to stress their power to the maximum.

In Table 5, we show the number of rows generated by DSDGEN for each of the tables of the data model proposed in Fig. 6. On the other hand, in the case of the fully denormalized data model obtained for Druid, the number of rows in the fact table will be the same as shown in Table 5, but this fact table will include all columns and data from all dimension tables.

For the 3 proposed SF (10, 30, and 50), we performed the two types of proposed tests: (i) Query execution and concurrency and (ii) data load and update. First, for the query latency test, we executed the 16 proposed SQL-OLAP queries in random order and by a single user. Then, we have performed another query execution test simulating 5 and 10 concurrent users in order to test concurrency on query execution performance. As for the second type of proposed performance tests, we have carried out the initial load test, measuring the execution time of the processes that load all the historical data generated for the different SFs. Finally, we have performed the data maintenance test, testing our proposed Equation 4 to measure and compare the performance of the two Big Data OLAP systems tested on data update processes.

**Table 5**
Number of rows of fact and dimension tables for SF 10, 30 and 50.

| Table | SF 10 | SF 30 | SF 50 |
|---|---|---|---|
| sales | 49,029,880 | 147,116,524 | 245,153,371 |
| customer addr. | 250,001 | 216,001 | 383,001 |
| customer dem. | 1,920,801 | 1,920,801 | 1,920,801 |
| date | 73,049 | 73,049 | 73,049 |
| household dem. | 7,201 | 7,201 | 7,201 |
| item | 102,000 | 40,000 | 62,000 |
| promotion | 500 | 411 | 522 |
| ship mode | 21 | 21 | 21 |
| store | 103 | 79 | 145 |
| time | 86,400 | 86,400 | 86,400 |
| warehouse | 11 | 11 | 11 |
| web page | 201 | 469 | 877 |
| web site | 43 | 27 | 23 |

## 5. Results

In the following two subsections we present and analyze the results of the performance tests executed as part of our proposed benchmark.

### *5.1. Query performance and concurrency*

First, we performed a execution of the 16 proposed queries by a single user, but in random order. We have run this process 10 times to obtain more reliable results by calculating the average execution time of each query. The purpose of the random execution of the queries is to reduce the effects of the OS cache of the node itself, which can have effects even if we have disabled the caching functions of the Big Data OLAP tool. The results of this test are shown in Table 6.

The time for each query is the arithmetic mean of the samples in the 10 runs. The total execution time is the sum of all the average execution times of the 16 queries. In addition, we have calculated the query performance increase (aka speedup) as the total execution time of the slowest tool divided by the time of the fastest tool at that SF.

Looking at the total execution times obtained, we observe that Druid is the most efficient tool in all cases, being up to 6.79 times faster than Hive (the slowest tool) for the most demanding scaling factor (SF50). However, when examining the individual query times, we can observe that Kylin is faster than the other tools for most queries, up to 70% of the queries in the SF30 and SF50 tests. Furthermore, Kylin executed about 50% of the queries in less than one second in any of the tested SFs, while Druid has only managed to execute one query in less than one second in the SF50 test.

The results obtained by Hive and Spark are much worse than those obtained by the two Big Data OLAP tools, not executing any query in the benchmark in less than one second. In the case of Hive, no query was executed in less than 5 seconds in any of the SFs tested, and for the SF30 and SF50 factors, most of the execution times were greater than 10 seconds.

The overall worsening of Kylin results compared to Druid and the closeness to the results obtained by Spark in the SF50,

**Table 6**
Results of the single-user query execution performance test.

| | SF 10 | | | | SF 30 | | | | SF 50 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Query | Kylin | Druid | Hive | Spark | Kylin | Druid | Hive | Spark | Kylin | Druid | Hive | Spark |
| 2 | 3.47 | 2.64 | 5.74 | 4.34 | 3.51 | 5.12 | 12.28 | 6.51 | 3.83 | 7.29 | 17.56 | 5.54 |
| 3 | 0.23 | 0.65 | 6.31 | 4.49 | 0.23 | 0.69 | 11.19 | 5.65 | 0.24 | 1.33 | 19.52 | 7.17 |
| 26 | 1.90 | 2.48 | 9.98 | 4.36 | 2.14 | 2.70 | 19.18 | 8.72 | 1.64 | 3.98 | 27.54 | 5.79 |
| 33 | 0.42 | 1.20 | 6.96 | 3.71 | 0.51 | 1.82 | 11.75 | 4.82 | 0.66 | 1.72 | 16.70 | 5.85 |
| 42 | 0.26 | 0.67 | 5.98 | 2.03 | 0.35 | 0.66 | 10.34 | 2.81 | 0.50 | 1.21 | 14.66 | 3.88 |
| 43 | 0.67 | 0.74 | 5.36 | 2.60 | 0.71 | 1.48 | 10.55 | 3.59 | 0.75 | 1.74 | 14.19 | 3.75 |
| 52 | 0.21 | 0.48 | 6.05 | 3.07 | 0.28 | 0.81 | 10.19 | 7.63 | 0.35 | 1.70 | 14.86 | 4.53 |
| 55 | 0.50 | 0.47 | 6.38 | 3.00 | 0.52 | 0.66 | 10.38 | 3.46 | 0.51 | 1.12 | 14.73 | 4.03 |
| 58 | 1.83 | 1.70 | 14.38 | 10.78 | 2.06 | 1.14 | 25.75 | 10.44 | 2.46 | 1.81 | 35.12 | 11.71 |
| 60 | 0.62 | 2.56 | 7.05 | 3.75 | 0.83 | 2.07 | 12.09 | 5.19 | 1.08 | 3.22 | 17.19 | 6.55 |
| 62 | 11.33 | 1.13 | 6.02 | 2.69 | 43.31 | 2.51 | 10.41 | 3.98 | 63.06 | 4.10 | 14.18 | 3.70 |
| 63 | 0.89 | 0.39 | 6.41 | 3.00 | 1.02 | 0.62 | 12.03 | 3.39 | 1.19 | 0.83 | 16.87 | 3.91 |
| 76 | 5.20 | 1.41 | 7.41 | 3.07 | 15.03 | 1.51 | 13.30 | 4.88 | 17.15 | 3.26 | 19.04 | 5.58 |
| 88 | 1.29 | 2.52 | 13.67 | 6.76 | 1.94 | 3.96 | 20.65 | 11.37 | 2.94 | 5.72 | 25.56 | 13.76 |
| 90 | 0.31 | 1.15 | 5.81 | 4.48 | 0.33 | 2.13 | 9.76 | 3.74 | 0.35 | 3.39 | 13.44 | 7.47 |
| 95 | 2.14 | 0.67 | 10.57 | 2.97 | 2.73 | 0.95 | 12.70 | 6.40 | 3.20 | 1.63 | 18.11 | 4.14 |
| Total | 31.27 | 20.85 | 124.06 | 65.10 | 75.51 | 28.84 | 212.54 | 92.59 | 99.91 | 44.05 | 299.27 | 97.37 |
| Speedup | 3.97 | 5.95 | 1.00 | 1.91 | 2.81 | 7.37 | 1.00 | 2.30 | 3.00 | 6.79 | 1.00 | 3.07 |
| Best queries | 9 | 7 | 0 | 0 | 11 | 5 | 0 | 0 | 11 | 4 | 0 | 1 |
| Queries <1s | 9 | 7 | 0 | 0 | 8 | 6 | 0 | 0 | 7 | 1 | 0 | 0 |
| Queries <5s | 14 | 16 | 0 | 14 | 14 | 15 | 0 | 8 | 14 | 14 | 0 | 7 |

is due to the high execution times obtained by Kylin for queries 62 and 76. Query 62 has to be rewritten to run in Kylin, which involves the addition of a subquery that returns many results. This data must be aggregated directly at runtime on the Kylin server and not in HBase, because Kylin cannot make use of the optimization that allows the data pre-aggregation. This is called post-aggregation in Kylin, since this data aggregation has not been pre-computed at OLAP cube building time. Query 76 poses a similar issue as in Kylin, but the post-aggregation required is less than required by query 62. Meanwhile, Druid, Spark and Hive do not suffer from this particular issue, as queries 62 and 76 are executed in a time that does not differ significantly from those obtained by the rest of the queries.

Based on the results of this single-user query performance test, we can conclude that Druid is the fastest tool at full test execution. However, Kylin is faster than the other tools in the individual execution of most queries, except for those involving post aggregation, such as queries 62 and 76. In any case, the times achieved by all tools except Hive are very fast for the data volumes involved in the benchmark. However, Kylin and Druid are the only ones to achieve execution times close to one second for most queries, while Spark times are often close to or over 5 seconds.

However, this first test does not consider user concurrency in a real OLAP scenario. Therefore, in addition to the single-user test, we performed two additional tests simulating 5 and 10 users executing the random sequences of the query set in parallel. The results of these concurrency tests for an SF50 are shown in Fig. 7.

We can observe that as the number of concurrent users increases in Kylin, the average execution time remains mostly constant. Meanwhile Druid and Spark are more affected than Kylin by the higher number of concurrent users, showing an increase to about 15 seconds in the average execution time in the 10-user test compared to the single-user test. Hive is the tool showing the worst results. Moreover, it was only possible to run the tests with 1 and 5 simultaneous users, as in the test with 10 users Hive did not have enough resources to serve all users simultaneously.

### *5.2. Data loading and updating performance*

In addition to the evaluation of the Big Data OLAP system in terms of query latency and user concurrency, we also consider very relevant its evaluation in terms of its performance in data loading and updating processes. Note that we have not included general-purpose Big Data SQL tools in this second test, as they are outside the main scope of this research.

In first place, we have run the initial load performance test of the entire generated data set using the 3 SFs tested. To measure the initial load performance, we must sum the execution time of the $ETL_1$ and $ETL_2$ processes described in Fig. 5a. The execution time of the $ETL_1$ process is the same for both tools, since it is a common process to load data into Apache Hive using the data model design shown in Fig. 6. However, $ETL_2$ requires implementing a specific data loading process for each Big Data OLAP tool, so the performance differences between Kylin and Druid in this test will be due to the execution time of this $ETL_2$ process.

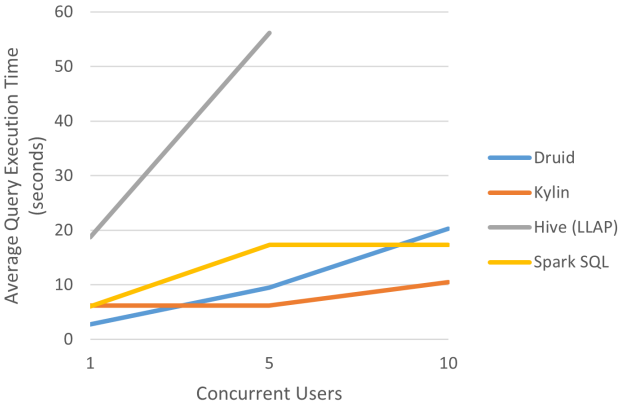



**Fig. 7.** Concurrency test results using SF50.

The results of the Kylin and Druid execution of the initial load performance test are shown in Fig. 8. The common times obtained by the execution of the $ETL_1$ process are 8, 12, and 18 minutes for SFs 10, 30, and 50 respectively, thus representing a very small percentage of the full test execution time. Whereas, when considering the overall execution time ($ETL_1$ + $ETL_2$), the times obtained by Kylin are significantly longer than the ones obtained by Druid. For example, for a SF50 Druid takes about 9 hours to complete the full load process while Kylin takes about 34 hours, thus being up to 3.8x slower than Druid.

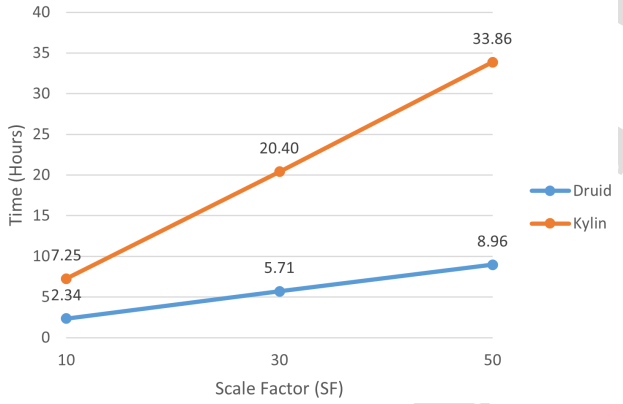



**Fig. 8.** Initial load test results for the 3 SF tested.

Based on these results, we can guess that an initial loading process in Kylin could take up to several days, while in Druid we can probably do it in less than a day. This difference is mainly due to the heavy use that Kylin makes of data pre-aggregation and pre-combination, making the data loading processes more complex and slower since it suffers from the so-called Dimensionality Course problem that we analyzed in our previous research [4].

However, it should be noted that we have used Map Reduce as the cube building engine in Kylin, when Spark is also supported and promises better performance. But experimenting with Spark as an alternative cube building engine in Kylin is beyond the scope of this research.

On the other hand, we must also be aware that, once the initial and full data load process has been executed, future data loads and updates usually do not usually involve such a large amount of data. Instead, we will often run incremental loads for adding new data to the OLAP cube or for refreshing much smaller periods of historical data than in an initial load. Therefore, in addition to the initial load test, in our benchmark we propose to run a data maintenance test with the aim of evaluating the performance of Big Data OLAP tools in the execution of data update processes.

This data maintenance test consists of measuring the time it takes to update 3 historical data intervals spanning 2 contiguous days. The update data is also generated by DSDGEN for the same 3 SFs used in the initial load test. In the data maintenance test originally proposed by TPC-DS, the performance measure consists in adding the execution times of the $ETL_1$ and $ETL_2$ processes described in Fig. 5b.

However, in our benchmark we identified that Big Data OLAP current approaches constraint data updating processes to the use of data segments that often require updating more data than necessary. To address this particularity, we proposed the Equation 4, which weights in $ETL_2$ the individual execution times of each of the 3 data segments according to their age, giving more weight to segments that are more recent in time. By doing this, we consider the fact that in a real Big Data OLAP scenario the most recent data segments are usually updated more frequently than the oldest ones.

Figure 9 shows the result of the Data Maintenance (DM) performance test for the different SFs used. The bars represent the data maintenance times obtained using the original metric approach ($ETL_1$ + $ETL_2$), while the lines show the times obtained using the proposed Equation 4 that weights $ETL_2$ as a function of the age of each segment.

## 6. Discussion

To our knowledge, this is the first study in which Kylin, Druid, Hive and Spark are benchmarked and the differences between general-purpose SQL tools for Big Data and those that optimize OLAP workloads are analysed. Compared to the partial implementations of TPC-DS in Big Data systems [14, 15, 17] reviewed in the state of the art, our proposed benchmark has been fully implemented in all 4 tested systems. This is possible thanks to our 3 proposed transformations (T1-T3), which allow implementing the selected TPC-DS sales data model and the 16 queries in any Big Data OLAP system. On the other hand, the selection of these 16 queries has been made taking into account the specific characteristics of OLAP workloads [4] and the common limitations of current Big Data OLAP tools that we have identified. These are the main novelties that differentiate our approach compared to TPC-DS and the other benchmarking approaches analysed.

Note that our aim is not to define a totally new workload, rather to create a new benchmark derived from TPC-DS that, unlike TPC-DS, could be implemented on current Big Data

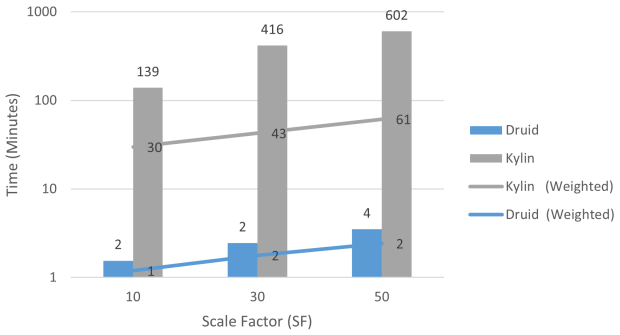



**Fig. 9.** Data maintenance test results for the 3 SF tested.

OLAP systems. The detailed analysis on the 16 selected queries (section 3.3), as well as the results of running the benchmark on Kylin and Druid, demonstrate that the proposed queries represent a suitable workload for benchmarking this type of systems.

However, this workload could be improved in future versions of our benchmark to represent an even more computationally demanding and realistic OLAP scenario. To this end, we consider applying the OLAP session concept as well as the query generator proposed in [28]. When using an OLAP viewer or table, users typically start from a base query and deepen their analysis by generating new queries derived from the initial one, according to their analysis goals. This is mainly achieved through drill-down, roll-up or slice OLAP operations. For example, in TPC-DS query 60 shown in Fig. 3, the data is queried in a drill-down manner by product code and filtered by a specific year and month. For deeper analysis, the user could generate a second query by adding the day number of the month (d_dom) as an additional grouping attribute to the product code (i_item_id).

Through these derived queries we can ensure the use of a wider variety of the attributes and metrics available in the data model, thus obtaining a more realistic and complete workload. In addition, to increase the computational requirements of these generated queries, we propose to increase the number of dimension attributes that are used simultaneously in the same query, which implies more JOIN type operations and combinations between attributes. Finally, we propose to modify the data generator (DSDGEN) to generate more ultra-high cardinality (UHC) dimensions to be used in the queries.

Regarding the results of our benchmark implementation on Kylin and Druid, it should be noted that these systems promise sub-second query latency with data volumes equal to or greater than 1 TB (SF100), being able to maintain this performance up to 100 TB (SF100000) in the most extreme scenarios. However, in our experiments we have used smaller Scale Factors of 10 GB, 30 GB, and 50 GB respectively. The pre-aggregated data structures (the OLAP cubes) that Kylin generates to achieve an ultra-high query performance have required hundreds of GB of disk space in our tests. As such, it has been unfeasible to run tests with higher scaling factors, since the data exceeded our current storage capabilities. Despite this limitation, the results shown that the tests performed were able to stress all four tools benchmarked. While many of the benchmark queries are executed in less than 1 second in Kylin and Druid, others require up to tens of seconds, something that is more evident in the results obtained by Hive and Spark.

The results of our benchmark implementation on Kylin and Druid allow us to conclude that both systems provide very high performance in the execution of OLAP data queries, achieving in most cases query latencies below or close to the second as promised in their specifications. On the other hand, the results obtained by the general-purpose Big Data SQL tools, Hive and Spark, show how these tools perform much worse when executing OLAP workloads. Unlike Kylin and Druid, these tools do not achieve the sub-second query latency required to maintain user interactivity [4] when using OLAP applications, such as dashboard reports or OLAP views.

If we compare our results with those of other studies that partially implement TPC-DS [15] on Hive and Spark with similar scaling factors, we can observe that they obtain similar or even worse times for some of the queries that we also executed in our test. In the case of Druid, we only found implementations [17, 24] of the SSB benchmark, whose data model is less complex than our model derived from TPC-DS. As in our study, these results show how Druid is able to execute queries in sub-second times unlike Hive or Spark, which require additional time to obtain the results.

Another of the advantages that can be highlighted in our benchmark is the proposed performance test for data maintenance processes. Unlike TPC-DS, our benchmark takes into account (i) the constraints imposed by current Big Data OLAP systems on data refresh processes and (ii) the fact that data refreshes usually affect the most recent data in time. Consequently, we propose Equation 4 to measure the performance in data refresh processes, giving more weight to the update times of more recent data blocks (aka segments) in time. The results of executing our data maintenance performance test in Kylin and Druid showed that Druid is a much more efficient tool than Kylin in the execution of data loading and updating processes.

## 7. Conclusions and future work

In this paper we present a new approach specifically aimed at benchmarking Big Data OLAP systems, based on the proven and widely used TPC-DS benchmark. Modern Big Data OLAP systems allow the execution of analytical OLAP queries [4] with interactive response times (from milliseconds to seconds) on data models of up to tens of billions of rows, i.e., up to hundreds of terabytes of data if we talk about data volume.

However, as a result of the state-of-the-art analysis of existing benchmarks [7, 10, 8, 9, 11, 12, 13] we identified a significant lack of suitability of current benchmarking approaches for use in Big Data OLAP systems. However, the TPC-DS approach [7, 8, 13] presents some very interesting features that can be used for benchmarking Big Data OLAP systems. Nonetheless, the inadequacy of its (i) data model, (ii) proposed queries and (iii) performance metrics to the nature and architectures of current Big Data OLAP systems make its implementation in them unfeasible.

To address the issues identified in the existing benchmarking approaches, in this paper we propose a new specific benchmark for Big Data OLAP systems based on TPC-DS. Our proposed benchmark includes in its definition three types of transformations that allow to generate equivalent data models but adapted to the specific nature of the Big Data OLAP system to be tested. In addition, we have identified criteria for discarding those queries proposed in TPC-DS that are not suitable for benchmarking current Big Data OLAP systems, applying those criteria to the selection and modification of the 16 queries that are part of our benchmark workload.

To demonstrate the applicability of our proposed benchmark, we have selected two modern and representative Big Data OLAP systems, Apache Kylin and Druid. Moreover, we have also implemented our benchmark on Hive (LLAP) and Spark (SQL), two general-purpose Big Data SQL tools that unlike Kylin and Druid do not focus on performance optimisation for the execution of OLAP queries. Thanks to our proposed benchmark, it is the first time that the performance of these 4 different systems in the execution of OLAP queries has been compared. Our benchmark also allowed us to compare the differences in data loading and updating performance between the two Big Data OLAP systems.

In our future work, we plan to implement our benchmark using higher scale factors (>= 1 TB) and extend the performance comparison to other Big Data systems with OLAP features, both on-premise (e.g. Pinot or Clickhouse) and cloud approaches (e.g. Azure Analysis Services or Imply). Finally, we also plan to improve the richness and computational demand of the workload by implementing the OLAP session concept [28], as well as evaluate the scalability of these systems.

## Acknowledgements

This research has been funded by the AETHER-UA project (PID2020-112540RB-C43) of the Spanish Ministry of Science and Innovation and by the BALLADEER project (PROMETEO/2021/088), funded by the Conselleria d'Innovació, Universitats, Ciència i Societat Digital.

## References

[1] R. Kimball, M. Ross, The data warehouse toolkit: the complete guide to dimensional modeling, John Wiley & Sons, 2011.

[2] R. Tardío, A. Maté, J. Trujillo, An iterative methodology for defining big data analytics architectures, IEEE Access 8 (2020) 210597–210616.

[3] L. Bellatreche, A. Cuzzocrea, I.-Y. Song, Advances in data warehousing and olap in the big data era, Information Systems 53 (2015) 39–40.

[4] R. Tardío, A. Maté, J. Trujillo, A new big data benchmark for olap cube design using data pre-aggregation techniques, Applied Sciences 10 (23) (2020) 8674.

[5] L. Q. Han, X. Jiang, Y. Song, C. Li, Hadoop olap engine, uS Patent 10,353,923 (Jul. 16 2019).

[6] F. Yang, E. Tschetter, X. Léauté, N. Ray, G. Merlino, D. Ganguli, Druid: A real-time analytical data store, in: Proceedings of the 2014 ACM SIGMOD international conference on Management of data, 2014, pp. 157–168.

[7] M. Poess, R. O. Nambiar, D. Walrath, Why you should run tpc-ds: A workload analysis, in: Proceedings of the 33rd International Conference on Very Large Data Bases, 2007, p. 1138–1149.

[8] R. O. Nambiar, M. Poess, The making of tpc-ds, in: Proceedings of the 33rd International Conference on Very Large Data Bases, 2006, p. 1049–1058.

[9] P. Boncz, T. Neumann, O. Erling, Tpc-h analyzed: Hidden messages and lessons learned from an influential benchmark, in: Technology Conference on Performance Evaluation and Benchmarking, 2013, pp. 61–76.

[10] P. O'Neil, B. O'Neil, X. Chen, The star schema benchmark (ssb), `http://www.cs.umb.edu/~poneil/StarSchemaB.pdf`, (Last accessed 15 April 2021) (2007).

[11] A. Ghazal, T. Ivanov, P. Kostamaa, A. Crolotte, R. Voong, M. Al-Kateb, W. Ghazal, R. Zicari, Bigbench v2: the new and improved bigbench, in: 2017 IEEE 33rd International Conference on Data Engineering (ICDE), IEEE, 2017, pp. 1225–1236.

[12] T. Rabl, A. Ghazal, M. Hu, A. Crolotte, F. Raab, M. Poess, H. Jacobsen, Bigbench specification v0. 1, in: Specifying Big Data Benchmarks, 2012, pp. 164–201.

[13] M. Poess, T. Rabl, H.-A. Jacobsen, Analysis of tpc-ds: the first standard benchmark for sql-based big data systems, in: Proceedings of the 2017 Symposium on Cloud Computing, 2017, pp. 573–585.

[14] V. Aluko, S. Sakr, Big sql systems: an experimental evaluation, Cluster Computing 22 (4) (2019) 1347–1377.

[15] M. Rodrigues, M. Y. Santos, J. Bernardino, Experimental evaluation of big data analytical tools, in: European, Mediterranean, and Middle Eastern Conference on Information Systems, 2019, pp. 121–127.

[16] E. Costa, C. Costa, M. Y. Santos, Evaluating partitioning and bucketing strategies for hive-based big data warehousing systems, Journal of Big Data 6 (1) (2019) 34.

[17] J. Camacho-Rodríguez, A. Chauhan, A. Gates, E. Koifman, O. O'Malley, V. Garg, Z. Haindrich, S. Shelukhin, P. Jayachandran, S. Seth, et al., Apache hive: From mapreduce to enterprise-grade big data warehousing, in: Proceedings of the 2019 International Conference on Management of Data, 2019, pp. 1773–1786.

[18] J. Correia, C. Costa, M. Y. Santos, Challenging sql-on-hadoop performance with apache druid, in: International Conference on Business Information Systems, 2019, pp. 149–161.

[19] J. Correia, M. Y. Santos, C. Costa, C. Andrade, Fast online analytical processing for big data warehousing, in: 2018 International Conference on Intelligent Systems (IS), 2018, pp. 435–442.

[20] W. Chen, H. Wang, X. Zhang, Q. Lin, An optimized distributed olap system for big data, in: 2017 2nd IEEE International Conference on Computational Intelligence and Applications (ICCIA), 2017, pp. 36–40.

[21] F. Ming, S. Guannan, L. Shuaishuai, Research on multidimensional analysis method of drilling information based on hadoop, in: 2017 3rd IEEE International Conference on Computer and Communications (ICCC), 2017, pp. 2319–2322.

[22] M. Song, M. Li, Z. Li, E. Haihong, A distributed self-adaption cube building model based on query log, in: International Conference on Human Centered Computing, 2017, pp. 382–393.

[23] M. S. Wiewiórka, D. P. Wysakowicz, M. J. Okoniewski, T. Gambin, Benchmarking distributed data warehouse solutions for storing genomic variant information, Database 2017 (2017).

[24] M. Rodrigues, M. Y. Santos, J. Bernardino, Big data processing tools: An experimental performance evaluation, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 9 (2) (2019) e1297.

[25] V. C. Storey, I.-Y. Song, Big data technologies and management: What conceptual modeling can do, Data & Knowledge Engineering 108 (2017) 50–67.

[26] J. M. Stephens, M. Poess, Mudd: a multi-dimensional data generator, ACM SIGSOFT Software Engineering Notes 29 (1) (2004) 104–109.

[27] P. Cao, B. Gowda, S. Lakshmi, C. Narasimhadevara, P. Nguyen, J. Poelman, M. Poess, T. Rabl, From bigbench to tpcx-bb: Standardization of a big data benchmark, in: Technology Conference on Performance Evaluation and Benchmarking, 2016, pp. 24–44.

[28] S. Rizzi, E. Gallinucci, Cubeload: A parametric generator of realistic olap workloads, in: International Conference on Advanced Information Systems Engineering, 2014, pp. 610–624.

[29] Public release of tpc-ds (v3.0), `http://tpc.org/tpcds`, (Last accessed 15 April 2021) (2021).

[30] A testbench for experimenting with apache hive at any data scale, `https://github.com/hortonworks/hive-testbench`, (Last accessed 15 April 2021) (2017).

Highlights

- A novel approach for benchmarking Big Data OLAP systems based on TPC-DS.
- Provides a specific set of rules to obtain equivalent designs of the proposed data model.
- Applicable to any of the current wide array of Big Data OLAP systems.
- It has been completely tested with two Big Data OLAP systems: Apache Kylin and Druid.

Roberto Tardío holds a Computer Science Engineering degree from the University of Alicante since 2013, where he also obtained a Msc in Computer Science Technology in 2014 and a PhD in 2021. He joined Lucentia in 2015, a research group associated to the University of Alicante. He currently works as Head of Big Data at the consulting firm Stratebi Business Intelligence, based in Madrid, (Spain). For this company, he manages projects and R&D in Big Data. His research interests are in the areas of Big Data architectures, requirements engineering, data modeling, OLAP tools and database benchmarking.

Alejandro Maté holds a Computer Science Engineering degree from the University of Alicante since 2009, where he also obtained a Msc in Computer Science Technology in 2010 and a PhD in 2013. Since 2019, he has been Associate Professor at the University of Alicante. He has published over 50 papers in international conferences (e.g. ER, CAiSE, RE) and JCR journals (e.g. Information Systems, Future Generations, Information & Software Technology). His research involves conceptual modeling, data warehouses, model driven development, and requirements engineering.

Juan Trujillo is a Full Professor at the University of Alicante, Dept. of Software and Computing Systems. Since he got his PhD in 2001, he has been leading the Business Intelligence and Big Data research in the department and has also been the founder and director of The Lucentia Research Group since 2008. His main research topics include Business Intelligence, Big Data, Data Warehouses, Decision Support Systems and Artificial Intelligence. He is author of more than 200 conference paper, such as ER, UML, DAWAK or CAiSE, and more than 60 JCR papers, such as DKE, DSS, ISOFT, IS, or InfSci.

**Roberto Tardío:** Conceptualization, Methodology, Investigation, Software, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization **Alejandro Maté:** Conceptualization, Validation**,** Writing - Review & Editing, Supervision, Project administration, Funding acquisition **Juan Trujillo:** Conceptualization, Supervision, Project administration, Funding acquisition

Roberto Tardio

Alejandro Mate

Juan Trujillo

**Declaration of interests**

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: