An Innovative Lambda-Architecture-Based Data Warehouse Maintenance Framework for Effective and Efficient Near-Real-Time OLAP over Big Data

Alfredo Cuzzocrea[1(✉)], Rim Moussa[2], and Gianni Vercelli[3]

[1] ICAR-CNR, University of Trieste, Trieste, Italy
alfredo.cuzzocrea@dia.units.it
[2] LaTICE Laboratory, University of Tunis, Tunis, Tunisia
rim.moussa@enicarthage.rnu.tn
[3] University of Genoa, Genoa, Italy
gianni.vercelli@unige.it

Abstract. In order to speed-up query processing in the context of *Data Warehouse Systems*, *auxiliary summaries*, such as *materialized views* and *calculated attributes*, are built on top of the data warehouse relations. As changes are made to the data warehouse through *maintenance transactions*, summary data become stale, unless the refresh of summary data is characterized by an expensive cost. The challenge gets even worst when near *real-time environments* are considered, even with respect to emerging *Big Data features*. In this paper, inspired by the well-known *Lambda architecture*, we introduce *a novel approach for effectively and efficiently supporting data warehouse maintenance processes in the context of near real-time OLAP scenarios*, making use of so-called *big summary data*, and we assess it via an empirical study that stresses the complexity of such OLAP scenarios via using the popular *TPC-H benchmark*.

1 Introduction

Usually, *Data Warehouse Systems* are deployed as part of a decision support system separated from the system of records (a.k.a. production databases). *On-Line Analytical Processing* (OLAP) queries, which execute at the data warehouse level, are long-running and complex query meant to extract *actionable knowledge*, but they are typically resource-consuming (e.g., [1–4]). Hence, in order to speed-up query processing, *auxiliary summaries*, such as *materialized views* and *calculated attributes*, are built on top of the data warehouse relations (e.g., [5]). As changes are made to the data warehouse through *maintenance transactions*, summary data become stale, unless the refresh of summary data is characterized by an expensive cost. Due to performance concerns, fresh data propagate down to the data warehouse system episodically (yearly

F. Y. L. Chin et al. (Eds.): BIGDATA 2018, LNCS 10968, pp. 149–165, 2018.
https://doi.org/10.1007/978-3-319-94301-5_12

or monthly refreshes), periodically (night refreshes) or, at-the-best, after some lag. Performance concerns are mainly resulting from complex data integration workflow executions and ACID (Atomicity, Coherency, Isolation, Durability – [6]) properties enforcement during the transaction maintenance task. Traditional data warehouse systems, with episodic or periodical data refreshing, foster *Retrospective Analytics* (e.g., [7]). The latter provides a look at what has already happened, and allows us to analyze past activities of an organization. On the other hand, in order to provide insights and actionable decisions at the right time, it is important to analyze *real-time data* (e.g., [8]). *Real-Time analytics* use cases occur in multiple instances, e.g.: *(i)* intelligent road-traffic management, *(ii)* remote health-care monitoring, *(iii)* complex event processing systems, and *(iv)* inventory management. Henceforth, novel approaches and paradigms are required in order to deal with OLAP query processing in dynamic environments, as the case of (data) changes at the data source level. In this paper, inspired by the well-known *Lambda architecture* [9], we introduce *a novel approach for effectively and efficiently supporting data warehouse maintenance processes in the context of near real-time OLAP scenarios*, with emerging *Big Data features* (e.g., [10]), and we assess it via an empirical study that stresses the complexity of such OLAP scenarios via using the popular *TPC-H benchmark* [11]. The paper extends a previous introduction short paper [12].

Refresh functions of data warehouse systems represent a common solution to face-of the described problem. They are commonly referred as *batch-incremental update processing* or *maintenance transactions* (e.g., [13]). Indeed, in standard data warehouse systems, *refresh procedures* load data into the data warehouse in a bulk mode. Prior to data loading, data are transformed. Data transformations are modeled in terms of complex *data integration workflows* and are part of the whole *data integration process*. Therefore, *big data processing for data warehouse maintenance* is becoming more and more relevant, as combined with *summary data management*, as also confirmed by recent research initiatives (e.g., [14]). Our research work lies in this specific scientific area, and predicts a new instance of *big data warehouse data*, the so-called *big summary data*, i.e. summary data structures that aid big data warehouse maintenance processes in emerging big data (e.g., *Cloud-based* – [15]) environments.

On a larger extent, data warehouse refresh can be modeled in terms of an eight-step process, as follows:

(Step 1) *Coping fresh data to the staging area*. The *staging area* is an intermediate storage area used for data processing during the data integration process; it sits between the data source(s) and the data target(s) [7,16].

(Step 2) *Running transformations on fresh data*. Transformations include: cleaning, de-duplication, data format conversion, derivation of new calculated values from existing data, filtering, joining, splitting, and so forth.

(Step 3) *Preparing transformed fresh data*. Prepare the insertion of transformed fresh data by usually disabling reference constraints and entity constraints, thus making indexes able to accelerate data warehouse insertion performance.

(Step 4) *Inserting fresh data into the data warehouse.* In some cases, it is necessary to merge fresh and stale data, indicate the time of last data update or maintain multiple data versions in order to handle suitable *Change Data Capture* (CDC).
(Step 5) *Validating inserted data.* Validate inserted data and processing different alerts (e.g., constraint violations). Alerts may need human solutions.
(Step 6) *(Re-)Setting-up constraints.* Re-enable reference constraints, entity constraints and other kinds of constraint over inserted data.
(Step 7) *Refreshing indexes.* Refresh indexes over inserted data.
(Step 8) *Refreshing data summaries.* Refresh auxiliary structures, such as materialized views, over inserted data.

Our research proposal aims at improving the big data warehouse maintenance problem via interacting with the process above. In order to better describe the proposed solution, we first need to focus on state-of-the-art data warehouse benchmarks (e.g., [17]), and how they are exploited to assess the performance of data warehouse systems.

The most prominent benchmarks for evaluating decision support systems are the various benchmarks issued by the *Transaction Processing Council* (TPC). The TPC-H benchmark [11] and its successor *TPC-DS* [18] assess the performance of the system under test according to two different ways, namely: (*i*) *Power Test*, and *Throughput Test*. The *Power Test* measures the query execution power of the system when connected with a single user. It runs the analysis in a *serial manner*, i.e. queries and update functions run one at a time and the elapsed time is measured separate for query run and refresh run. The *Throughput Test* measures the ability of the system to process concurrent queries and update functions in a multi-user environment.

Looking at refreshing operations in greater details, TPC-H benchmark exposes two refresh functions, namely *RF1*, for loading new sales (TPC-H introduces a classical product-order-supplier multidimensional model), and *RF2* for purging obsolete sales. In order to publish a TPC-H compliant performance result, the system needs to support ACID properties, by also specifying one of the four *isolation levels* (namely serializable, repeatable reads, read committed, read uncommitted) as well as the snapshot isolation level [19]. Initially, TPC-DS benchmark follows the TPC-H benchmark. In order to satisfy big data analytics requirements, *TPC-DS 2.0* reverted to a simpler model, in which analytic queries and data maintenance procedures are strictly distinct. Big data solutions are inherently not ACID-compliant, while most systems are indeed BASE-compliant (BASE – Basically Available, Soft state, Eventual consistency) compliant [20].

By inspecting the nature of tests implemented by popular data warehouse benchmarks, we can observe that *the overlapping of ACID-compliant OLAP queries and BASE-compliant data maintenance functions (e.g., refresh) overall turns to be extremely costful.* Hence, ad-hoc solutions must be devised in order to circumvent the target problem.

Specifically, we propose (and experimentally assess) a new big data warehouse maintenance methodology that pursues the idea of *first* performing a shortcut

from (Step 1) to (Step 8), in order to improve both query performance as well as the query accuracy with respect to fresh data, and *then* performing the remaining steps from (Step 2) to (Step 7). The proposed approach is based on so-called *delta computations*, inspired by the well-known Lambda architecture [9], which allow us to (1) serve queries from both stale (big) data summaries in the data warehouse system and fresh (big) data in the staging area. In addition to this, the proposed approach applies *factorization* of streams processing for (2) fast computation of so-called *delta views*, in order to capture dynamic properties of big data systems. Finally, the proposed approach also (3) applies *postponement of the data warehouse maintenance transaction* to an opportune time (still based on a cost-aware analysis).

2 Related Work

Several proposals have been presented to address the management of real-time data in Data Warehouses. Proposals address fast data processing in OLAP systems using different approaches. Hereafter, we overview related work. Authors in [21–23] propose inserting real time data into OLAP multidimensional cube structures, instead of into the Data Warehouse itself. They argue that insertions in the data cubes would occur faster, due to the fact that they are not executed over highly indexed tables that contain a large amount of historical data.

In [24], authors foster data fragmentation of the data warehouse over a shared-nothing architecture, to accelerate the data integration process. The *maintenance transaction* becomes distributed and more complex to manage with well admitted commit distributed protocols (2-PC and 3-PC).

Dehne et al. propose *CR-OLAP* [25–27], a Real-time OLAP system based on a distributed index structure for OLAP, refered to as a *distributed PDCR tree*. *R-Store* [28,29] -A Scalable Distributed System for Supporting Real-time Analytics, which periodically materializes real-time data into a data cube. *R-Store* uses HBase for data storage and MapReduce for query processing, and implements MVCC (Multi-version concurrent control) to support real-time OLAP. In *CR-OLAP*, [28], summary data maintenance is not investigated.

In [30–33], Ferreiran, Cuzzocra et al., demonstrate propose near-real time data warehouses and propose a Rewrite/Merge Approach for Real-Time Data Warehousing. The proposed architecture in [32,33] implements a real-time data warehouse without data duplication. It is composed of three main components: the Dynamic Data Warehouse (D-DW), the Static Data Warehouse (S-DW) and the Merger. The Integration between the D-DW and the S-DW is performed offline.

Real-time new systems were deployed at Google and LinkedIn. The latters have different workloads. In [34], authors propose *Mesa* a highly scalable analytic data warehousing system that stores critical measurement data related to Google's Internet advertising business. *Mesa* satisfies near real-time data ingestion and query-ability requirements. It supports continuous updates which should be available for querying consistently across different views within

minutes. *Pinot* [35] is a real-time distributed OLAP datastore designed to scale horizontally and open sourced by LinkedIn. The database is column oriented and implements bitmaps and inverted indexes. It is suited for analytical use cases on immutable append-only data with exclusively selection, aggregation, filtering, group by, order by, distinct queries on fact data. *Pinot* does not suit TPC DSS benchmarks like workloads, which show complex join operations. *Druid* [36] is an open source data store designed for real-time exploratory analytics on large data sets. The system combines a column-oriented storage layout, a distributed, shared-nothing architecture, and an advanced indexing structure allowing fast data ingestion and analytics of events.

Materialized views (a.k.a. MVs, summary tables, aggregate tables), store pre-computed results, and are widely adopted to facilitate fast queries on large data sets. As update batches arrive at a high rate, it is infeasible to continuously update MVs and a common solution is to group and defer maintenance transactions. Meanwhile, the MVs become stale, which leads to inaccurate query results. Multiple papers investigated Materialized views' refresh mechanisms and optimizations [37–41].

The *Lambda Architecture* [42–45] targets Big Data processing at scale and involves both batch and stream systems. Indeed, batch and stream workloads run in parallel on the same incoming data. The Lambda Architecture is made up of three layers, namely, (*i*) the *Batch Layer* which ingests and stores large quantities of immutable data and calculates batch views, (*ii*) the *Speed Layer* which processes stream data into views and deploys the views on the *Serving Layer*, and (*iii*) the *Service Layer* which queries the batch and real-time views and merges them into serving up views. Challenging problems are handling large quantities of data at the batch layer, unknown and time-varying data streams at the speed layer, as well as fast merge of views at the service layer.

Our paper is inspired by Lambda architecture principles for architecting near-real time OLAP scenarios in big data systems.

3 Big Summary Data: Definition and Management

Data warehousing is based on (1) collecting, cleansing, and integrating data from a variety of operational systems; (2) calculating summary data to address performance leaks, and (3) performing data analysis for decision support.

In order to address performance leaks related to complex OLAP queries' processing over big data, data warehouses build data summaries. Next, we overview *derived attributes* and *materialized views* cycle life from design to refresh. Data Summaries' management has three costs, namely (*i*) building cost, (*ii*) storage cost, (*iii*) refresh cost and have an age which indicates how old are these data snapshots.

Examples in this section, are based on TPC-H benchmark [11]. Figure 1 illustrates the Database Schema of TPC-H Benchmark.

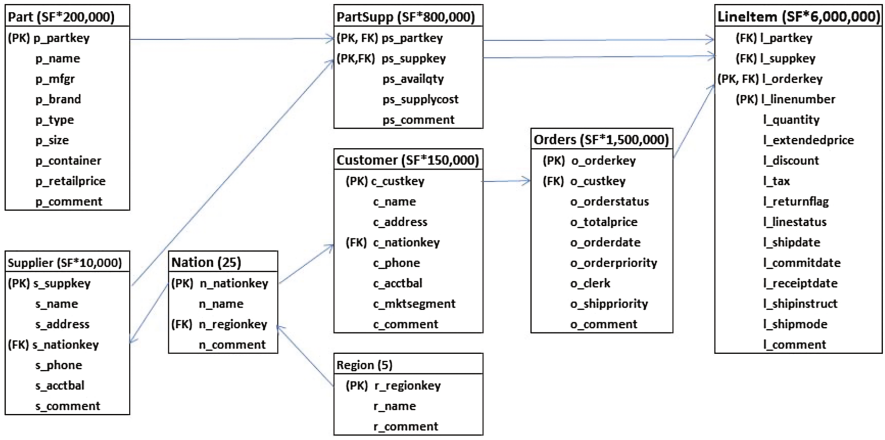

Fig. 1. Database schema of TPC-H benchmark.

3.1 Design of Data Summaries

Given, (*i*) a relational data warehouse schema, (*ii*) a workload composed of OLAP queries, (*iii*) refresh streams, we propose simple recommendations for proposing both of (*i*) derived attributes and (*ii*) materialized views which allow to achieve high performance OLAP over relational database management systems. Hereafter, we briefly recall definitions of *materialized views* and *derived attributes*, and motivate their usage for each type of business query of TPC-H workload.

Materialized Views. A *materialized view* summarizes large number of detail rows into information that has a coarser granularity. As the data is precomputed, an aggregate table allows faster cube processing. Research work propose cost models assessing materialized view's recommendations [46–49]. In this paper, we propose a materialized view for each query, then a grouping along full inclusion of dimensions. That's if the MV -MV_i of a query Q_i is included in the MV -MV_j recommended for query Q_j, then -MV_j is proposed for both Q_i and Q_j. The size of an MV is the number of rows in the MV. The latter is simply derives as the product of all attributes cardinalities, except attributes which are in functional dependency with an other attribute (as customer name is in functional dependency with customer key), and attributes to which refer other attributes (as customer key is referred in Orders table). We investigate Materialized Tables for two types of OLAP queries. First, for OLAP queries which MVs sizes are scale factor independent, and OLAP queries having very sparse cubes. Next, we overview by example, these two types of OLAP queries.

Business query Q12 of TPC-H benchmark is illustrated in Fig. 2(a). Q12, the *Shipping Modes and Order Priority Query* counts, by *ship mode*, for lineitems actually received by customers in a given year, the number of lineitems belong-

ing to orders for which the *l_receiptdate* exceeds the *l_commitdate* for two different specified ship modes. Only lineitems that were actually shipped before the *l_commitdate* are considered. The aggregate table for Q12 is shown in Fig. 2(b) while the MV12 recommended for Q12 is illustrated in Fig. 2(c). Whether is the scale factor of TPC-H benchmark, MV12 has fixed number of rows equal to 49 computed as follows, $\sharp$ *Line Receipt Years* $= 7 \times \sharp$ *line ship modes* $= 7$.

```
SELECT l_shipmode,
SUM(case when o_orderpriority in ('1-URGENT','2-HIGH')
    then 1 else 0 end) as high_line_count,
SUM(case when o_orderpriority not in ('1-URGENT','2-HIGH')
    then 1 else 0 end) as low_line_count
FROM orders, lineitem
WHERE o_orderkey = l_orderkey
AND l_shipmode in ('[SHIPMODE1]', '[SHIPMODE2]')
AND l_commitdate < l_receiptdate
AND l_shipdate < l_commitdate
AND l_receiptdate >= date '[DATE]'
AND l_receiptdate < date '[DATE]' + '1' year
GROUP BY l_shipmode
ORDER BY l_shipmode;
```

(a)

```
CREATE TABLE mv_q12 AS (
SELECT year(l_receiptdate) as years, l_shipmode,
sum(case when o_orderpriority in ('1-URGENT','2-HIGH')
   then 1 else 0 end) as high_line_count,
sum(case when o_orderpriority not in ('1-URGENT','2-HIGH')
   then 1 else 0 end) as low_line_count
FROM orders, lineitem
WHERE  o_orderkey = l_orderkey
AND l_commitdate < l_receiptdate
AND l_shipdate < l_commitdate
GROUP BY year(l_receiptdate), l_shipmode) ;
```

(b)

```
SELECT l_shipmode,sum(high_line_count),sum(low_line_count)
FROM mv_q12
WHERE l_shipmode in ('[SHIPMODE1]', '[SHIPMODE2]')
AND year = '[YEAR]'
GROUP BY l_shipmode ;
```

(c)

```
SELECT c_name, c_custkey, o_orderkey, o_orderdate,
    o_totalprice, sum(l_quantity)
FROM customer, orders, lineitem
WHERE  o_orderkey IN ( SELECT  l_orderkey
    FROM  lineitem
    GROUP BY l_orderkey
    HAVING  sum(l_quantity) > [QUANTITY] )
AND c_custkey = o_custkey
AND o_orderkey = l_orderkey
GROUP BY c_name,c_custkey,o_orderkey,...,o_totalprice
ORDER BY o_totalprice desc, o_orderdate;
```

(d)

```
CREATE TABLE mv_q18 AS (
SELECT c_name, c_custkey, o_orderkey, o_orderdate,
o_totalprice, sum(l_quantity) as sum_qty
FROM customer, orders, lineitem
WHERE c_custkey = o_custkey
AND o_orderkey = l_orderkey
GROUP BY c_name, c_custkey, o_orderkey, ..., o_totalprice
HAVING sum(l_quantity) > 300);
```

(e)

Fig. 2. SQL statement of TPC-H business query $Q12$ (a), aggregate table Agg_Q12 for $Q12$ (b), MV_Q12 for $Q12$ (c), $Q18$ (d), MV_Q18 for $Q18$ (e).

Business query Q18 of TPC-H benchmark is illustrated in Fig. 2(d). Q18 -the *Large Volume Customer Query* finds all customers who have ever placed large quantity orders. The query lists the customer name, customer key, the order key, date and total price and the quantity for the order. Q18 return few rows as 3.8 ppm (parts per million) of orders are big orders. Hence, it's recommended to calculate an MV for Q18 (see Fig. 2(e)).

Derived Attributes. *Derived Attributes* are calculated from other attributes. We recommend derived attributes for OLAP cubes which dimensionality is scale factor dependent, as Q10 (see Fig. 3).

Indeed, for this type of business queries, *derived attributes* are much less space consuming than *aggregate tables*. Q10 identifies customers who might be having problems with the parts that are shipped to them, and have returned

```
SELECT n_name, c_custkey, ..., c_comment,
       sum(l_extendedprice*(1-l_discount))
FROM customer, orders, lineitem, nation
WHERE o_orderdate >= ['DATE']
AND o_orderdate < ['DATE'] + interval '3' month
AND l_returnflag = 'R'
AND o_orderkey = l_orderkey
AND c_custkey = o_custkey
AND c_nationkey = n_nationkey
GROUP BY n_name, c_custkey, ..., c_comment
ORDER BY revenue desc;
```

(a)

```
SELECT n_name, c_custkey, ..., c_comment
    sum(o_sum_lost_revenue) lost_revenue
FROM customer, orders, nation
WHERE o_orderdate >= ['DATE']
AND o_orderdate < ['DATE'] + interval '3' month
AND c_custkey = o_custkey
AND c_nationkey = n_nationkey
GROUP BY n_name, c_custkey, ..., c_comment ;
ORDER BY revenue desc;
```

(b)

```
SELECT  n_name, c_custkey, c_name, ..., c_comment
          sum(lost_revenue) as lost_revenue
FROM (
 SELECT n_name, c_custkey, c_name, ..., c_comment,
          sum(o_sum_lost_revenue) as lost_revenue
 FROM customer, orders, nation
 WHERE c_custkey = o_custkey
 AND o_orderdate >= date ['DATE']
 AND o_orderdate < date ['DATE'] + interval '3' month
 AND c_nationkey = n_nationkey
 GROUP BY n_name, c_custkey, c_name, ..., c_comment
 UNION ALL
 SELECT n_name, c_custkey, c_name, ..., c_comment,
          sum(o_sum_lost_revenue) as lost_revenue
 FROM customer, orders_temp, nation
 WHERE c_custkey = o_custkey
 AND o_orderdate >= date ['DATE']
 AND o_orderdate < date ['DATE'] + interval '3' month
 AND c_nationkey = n_nationkey
 GROUP BY n_name, c_custkey, c_name, ..., c_comment)
GROUP BY n_name, c_custkey, c_name, ..., c_comment;
```

(c)

```
SELECT  n_name, c_custkey, c_name, ..., c_comment,
          sum(lost_revenue) as lost_revenue
FROM (
 SELECT n_name, c_custkey, c_name, ..., c_comment,
 sum(o_sum_lost_revenue) as lost_revenue
 FROM customer, orders, nation
 WHERE c_custkey = o_custkey
 AND o_orderdate >= date ['DATE']
 AND o_orderdate < date ['DATE'] + interval '3' month
 AND c_nationkey = n_nationkey
 GROUP BY n_name, c_custkey, c_name, ..., c_comment
 UNION ALL
 SELECT n_name, c_custkey, c_name, ..., c_comment,
 -sum(l_extendedprice*(1-l_discount)) as lost_revenue
 FROM customer, v_join_rf2_streams, nation, time
 WHERE c_custkey = o_custkey
 AND l_orderkey = o_orderkey
 AND o_orderdate >= date ['DATE']
 AND o_orderdate < date ['DATE'] + interval '3' month
 AND l_returnflag = 'R'
 AND c_nationkey = n_nationkey
 GROUP BY n_name, c_custkey, c_name, ..., c_comment)
GROUP BY n_name, c_custkey, c_name, ..., c_comment;
```

(d)

Fig. 3. SQL statement of TPC-H business query $Q10$ (a), with *o_sum_lost_revenue* immutable derived attributes (b), serving layer processing RF1 refresh function (c), serving layer processing RF2 refresh function (d).

them, for so, it calculates the *lost revenue* for each customer for a given quarter of a year. In order to improve the response time of Q10, we propose the following alternatives, (1) Either add 28 derived attributes *c_sum_lost_rev /year/quarter* to CUSTOMER relation, or (2) add one attribute *o_sum_lost_rev* to ORDERS relation. Notice that, the second alternative is better than the first with respect to both storage overhead and cost of refresh of stale derived attributes. Indeed, following inserts or deletes of orders (respectively TPC-H refresh functions RF1 and RF2), the 28 derived attributes are stale, while refreshes do not render stale the attribute *o_sum_lost_rev*. Attribute *o_sum_lost_rev* will enable a gain in performance results from saving the cost of the join of LINEITEM and ORDERS tables.

Derived attributes alter the data warehouse schema, which is not costful for column-oriented storage systems, and allow a gain in performance through reducing both temporal and spatial complexities. The main point is to choose attributes which are not stale after data refresh or refresh cost is not costful.

TPC-H Benchmark Analysis. Following our directives for recommending on when to recommend Materialized Views and Derived Attributes, We concluded that TPC-H business queries fall into three categories (see Table 1).

Table 1. TPC-H workload taxonomy.

Dimensionality	Sparsity	TPC-H Business Queries	Recommendation
SF dependent	very sparse	Q15, Q18	Materialized Views
SF dependent	dense enough	Q2, Q9, Q10, Q11, Q20, Q21	Derived Attributes
SF independent	very sparse	—	Materialized Views
SF independent	dense enough	Q1, Q3, Q4, Q5, Q6, Q7, Q8, Q12, Q13, Q14, Q16, Q17, Q19, Q22	Materialized Views

Table 2 illustrates a total storage cost of all materialized views equal to 0.88 GB for all scale factors. The cost of derived attributes scales with TPC-H scale factor and is 0.5 GB for SF =10 (see Table 3).

Table 2. Materialized views data for TPC-H benchmark for any scale factor.

MV-Qi	♯Rows	Volume (MB)
mv-q1	129	0.008
mv-q3	2210908	52.712
mv-q4	135	2.241
mv-q5	175	2.563
mv-q6	3850	0.088
mv-q7	4375	0.067
mv-q8	131250	2.128
mv-q12	49 1813	0.002
mv-q13	47 3196	0.003
mv-q14	84 1344	0.001
mv-q15	28 728	0.001
mv-q16	187495	2.861
mv-q17	1000	0.011
mv-q18	624	0.023
mv-q19	39859141	836.278
mv-q22	500018	7.630
	total	*901.817*

Table 3. Tables' volumes respectively before and after adding new derived attributes for SF = 10.

Table	Volume (MB)	New Volume (MB)
Supplier	4.578	4.959
PartSupp	366.211	427.246
Orders	1001.358	1115.799
Lineitem	9381.974	9839.631

3.2 Refresh of Data Summaries

In this section, we first overview different data integration strategies, namely *Lazy integration* or *Eager integration*, which handle differently the processing of fresh data. Then, we detail data summaries management from inception to maintenance.

Data Integration [50]**.** Data Integration is the process of integrating data from multiple sources. Integration is either *Lazy* or *Eager* [51]. *Lazy Integration* keeps data at the sources and requires a mid-tier for query processing. The mid-tier determines the sources which answer the query and devises the execution tree. Once queries' answer sets obtained, the mid-tier performs required post-processing and returns a result set to the user-application. Summarizing, *Lazy integration* leaves data at sources, integrates data on-demand i.e. at query time, and queries' answer sets is accurate. The system is out-of-service when the sources are unavailable. The data sources are queried by the decision support system as well as the transactional system. This might degrade queries' performances.

Eager integration, is based on data warehousing. Thus, information of each source of interest is extracted in-advance and processed as appropriate, then merged with information from other sources and stored. The data warehouse is a database that is designed for query and analysis and is operational even when sources are unavailable. High query performance is achieved by building data summaries and local processing at sources is unaffected by the decision support system workload. Summarizing, with *Eager integration* a business query answer set might be stale. In order to overcome staleness, the data warehouse is refreshed episodically, periodically, or at best after some time. The maintenance transaction is also costful.

The combination of eager and lazy integration approaches is challenging and will enable OLAP over fresh data and load balancing between the transactional and the decision support system.

Refresh Operations. Given, (*i*) a relational data warehouse schema, (*ii*) a workload -a set of queries, (*iii*) refresh streams, (*iv*) calculated attributes and (*v*) materialized views; we have to determine when and how data summaries are refreshed. Two refresh strategies are proposed for the refresh of data summaries, and are detailed hereafter,

- *Eager refresh*: derived attributes and materialized views are refreshed with in the *maintenance transaction*. Hence, the data warehouse is coherent at the expense of costful maintenance.
- *Lazy refresh*: the refresh of calculated attributes and materialized views is delayed and is not part of the *maintenance transaction*. Thus, the data warehouse is incoherent for better performances.

Data Summaries refresh processing is performed is either *incremental*, requires *full reprocessing*, or both,

- *Incremental processing*: an *incremental refresh* executes first a sophisticated merge of the old snapshot and a new snapshot built over fresh data and if needed relations in the warehouse, and second integrates fresh data in the data warehouse.
- *Full reprocessing*: a *full reprocessing* integrates fresh data in the data warehouse, then recomputes data summaries.
- *Hybrid processing*: some parts require full reprocessing, while others can be incrementally refreshed.

In Eqs. (1)–(4), we analyze relational algebra operations involving different types of relations, namely relations which undergo inserts or deletes and immutable relations.

$$\begin{aligned}&\text{For R1 and R2 concerned by new inserts,}\\&(R1 \cup \Delta R1) \bowtie (R2 \cup \Delta R2)\\&= (R1 \bowtie R2) \cup (R1 \bowtie \Delta R2) \cup (\Delta R1 \bowtie R2) \cup (\Delta R1 \bowtie \Delta R2)\\&\text{IF fresh data does not refer to stale data as in TPC-H RF1}\\&\text{THEN } R1 \bowtie \Delta R2 = \text{AND } \Delta R1 \bowtie R2 =\\&= (R1 \bowtie R2) \cup (\Delta R1 \bowtie \Delta R2)\end{aligned} \tag{1}$$

$$\begin{aligned}&\text{For R1 concerned by new inserts and R2 immutable,}\\&(R1 \cup \Delta R1) \bowtie R2\\&= (R1 \bowtie R2) \cup (\Delta R1 \bowtie R2)\end{aligned} \tag{2}$$

$$\begin{aligned}&\text{For R1 and R2 concerned by deletes,}\\&(R1 - \Delta R1) \bowtie (R2 - \Delta R2)\\&= (R1 \bowtie R2) - (R1 \bowtie \Delta R2) - (\Delta R1 \bowtie R2) \cup (\Delta R1 \bowtie \Delta R2)\\&\text{IF } (\Delta R1 = R1 \bowtie \Delta R2) \text{ THEN}\\&\quad \Delta R1 \bowtie R2 = R1 \bowtie \Delta R2 \bowtie R2 = R1 \bowtie \Delta R2\\&\quad AND\ \Delta R1 \bowtie \Delta R2 = R1 \bowtie \Delta R2 \bowtie \Delta R2 = R1 \bowtie \Delta R2\\&= (R1 \bowtie R2) - (\Delta R1 \bowtie R2)\end{aligned} \tag{3}$$

$$\begin{aligned}&\text{For R1 concerned by deletes and R2 immutable,}\\&(R1 - \Delta R1) \bowtie R2\\&= (R1 \bowtie R2) - (\Delta R1 \bowtie R2)\end{aligned} \tag{4}$$

The analysis of TPC-H benchmark workload reveals that all TPC-H benchmark business questions are concerned by refresh functions, except the following business queries Q2, Q11, Q13 and Q16 (i.e. 4 over 22 queries).

4 Near Real-Time OLAP Scenarios

In this section, we detail near real-time OLAP scenarios querying fresh data for an OLAP query improved using derived attribute, and a second for an OLAP query tuned using a materialized view.

4.1 Stream Workflow Management

Each stream has (1) a unique identifier *streamID*, typically a *timestamp*, (2) a *stream type* -for TPC-H benchmark, *stream type* is either $RF1$ or $RF2$, and (3) is composed of a sequence of operations (inserts, updates and deletes). All data entering the system is dispatched to both the *batch layer* and the *speed layer* for processing. The *batch layer* has two functions: (i) to manage the data warehouse, and (ii) to compute the *batch materialized views*. The *speed layer* has also two functions: (i) to manage the incremental updates i.e. streams, and (ii) to compute the *speed materialized views*. The *serving layer* merges *batch materialized views* and the *speed materialized views*, and indexes the resulting views i.e. *serving views*; so that they can be queried in low-latency and ad-hoc way.

In Table 4, we enumerate the different processing related to TPC-H business queries, and regroup them in order to depict which queries perform the same relational operations.

Table 4. TPC-H business queries' processing requirements implied by the batch updates.

Processing	*Business queries: Q_i*
σ LineItem	$l\_shipdate$ [?][?]: Q1,Q3,Q6,Q7,Q14,Q15,Q20 $l\_discount$ [?][?]: Q6 $l\_quantity$ < [?]: Q6,Q19 $l\_commitdate < l\_receiptdate$: Q4,Q12 $l\_returnflag =' R'$: Q10 $l\_shipmode$ [= $\|in$][?]: Q12, Q19 $l\_shipdate < l\_commitdate$: Q12 $l\_shipdate$ [?][?]: Q1 $l\_receiptdate$ [?][?]: Q12 $sum(l\_quantity) >$ [?]: Q18 $l\_shipinstruct =$ [?]: Q19 $l\_quantity$ [?]: Q19 $l\_receiptdate < l\_commitdate$: Q21
σ Orders	$o\_orderdate$ [?][?]: Q3,Q4,Q5,Q8,Q10 $o\_orderpriority$ [?][?]: Q12 $o\_commentlike$[?]: Q13 $o\_orderstatus =' F'$: Q21
LineItem $\bowtie$ R	$R = Orders$: Q3,Q4,Q5,Q7,Q8,Q9,Q10,Q12,Q21 $R = Part$: Q8,Q9,Q14,Q17,Q19 $R = PartSupp$: Q9,Q20 $R = Supplier$: Q5,Q7,Q8,Q9,Q15,Q20,Q21
Orders $\bowtie$ R	$R = Customer$: Q3,Q5,Q7,Q8,Q10,Q13,Q18,Q22

In Fig. 4(a), we show general processing as dictated by TPC-H queries. Then, in Fig. 4(b), we propose the join of *Orders' stream* and *LineItems' stream*, since the join operation is required by multiple queries.

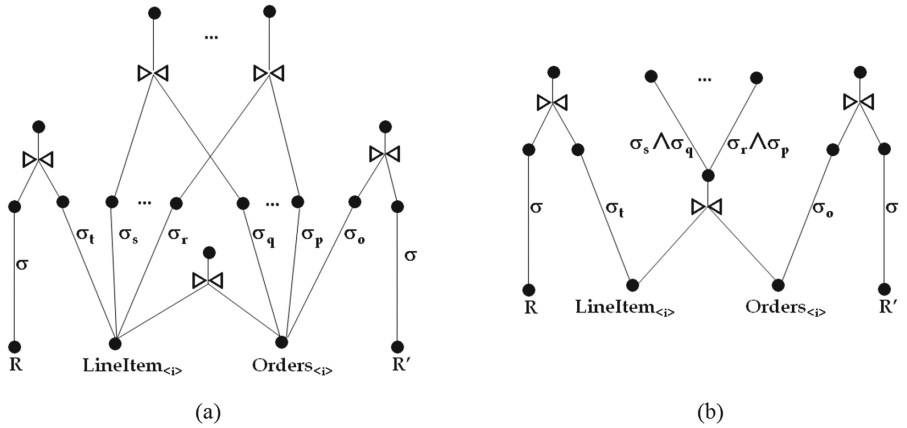

Fig. 4. Non-optimized stream processing (a) - Factorized stream processing (b).

4.2 Materialized View Management

The discounted revenue query Q19 of TPC-H benchmark [11], finds the gross discounted revenue for all orders for three different types of parts that were shipped by air or delivered in person. Parts are selected based on the combination of specific brands, a list of containers, and a range of sizes.

4.3 Derived Attribute Management

Business query Q10 of TPC-H benchmark, illustrated in Fig. 3, identifies customers who might be having problems with the parts that are shipped to them. The query calculates for each customer the lost revenue of returned parts, i.e. lineitems fulfilling ($l\_returnflag$ = 'R'). In order, to accelerate the processing of Q10, we propose the derived attribute *o_sum_lost_revenue* to be calculated for each order. This derived attribute allows to save the join of the two tables *lineitem* and *orders*. The computation of *o_sum_lost_revenue* is described in Fig. 3(b). Next, we discuss Q10 processing in case there are refreshes, namely process new orders (RF1) and process deletes of orders (RF2).

- Query Q10 is affected by new inserts of orders (RF1). In order to enable real-time Q10, the derived attribute *o_sum_lost_revenue* is calculated for each new order in each new stream, as illustrated in Fig. 3(c). This processing performed at the speed layer requires different relational operations such as restrictions, join, projection and scalar functions. The service layer is responsible for the merge of Q10 resultsets at the batch layer and at the speed layer.

– Query Q10 is also affected by delete of orders (RF2). The corresponding processing is illustrated in Fig. 3(d).

5 Conclusions and Future Work

Inspired by the well-known Lambda architecture, in this paper we have introduced and experimentally assessed a novel approach for effectively and efficiently supporting data warehouse maintenance processes in the context of near real-time OLAP scenarios, which makes use of innovative big summary data. Experiments have been conducted against the popular TPC-H benchmark.

In future work, we will first conduct further experiments for bigger scales of the TPC-H benchmark. Secondly, we will generate *sketch synopsis* rather than derived attributes for several query classes of the TPC-H benchmark, namely: Q2, Q9, Q10, Q11, Q20 and Q21. Those, in fact, are suitable to sketch-based computations.

References

1. Cuzzocrea, A., Song, I., Davis, K.C.: Analytics over large-scale multidimensional data: the big data revolution! In: Proceedings of DOLAP 2011, pp. 101–104. ACM (2011)
2. Cuzzocrea, A.: Aggregation and multidimensional analysis of big data for large-scale scientific applications: models, issues, analytics, and beyond. In: Proceedings of the 27th International Conference on Scientific and Statistical Database Management, SSDBM 2015, La Jolla, 29 June–1 July 2015, pp. 23:1–23:6 (2015)
3. Cuzzocrea, A., Bellatreche, L., Song, I.: Data warehousing and OLAP over big data: current challenges and future research directions. In: Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP, DOLAP 2013, San Francisco, 28 October 2013, pp. 67–70 (2013)
4. Cuzzocrea, A.: Analytics over big data: Exploring the convergence of data warehousing, OLAP and data-intensive cloud infrastructures. In: 37th Annual IEEE Computer Software and Applications Conference, COMPSAC 2013, Kyoto, 22–26 July 2013, pp. 481–483 (2013)
5. Gupta, H., Mumick, I.S.: Selection of views to materialize in a data warehouse. IEEE Trans. Knowl. Data Eng. **17**(1), 24–43 (2005)
6. Härder, T., Reuter, A.: Principles of transaction-oriented database recovery. ACM Comput. Surv. **15**(4), 287–317 (1983)
7. Kimball, R., Ross, M.: The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. John Wiley, New York (2013)
8. Cuzzocrea, A.: CAMS: OLAPing multidimensional data streams efficiently. In: Pedersen, T.B., Mohania, M.K., Tjoa, A.M. (eds.) DaWaK 2009. LNCS, vol. 5691, pp. 48–62. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03730-6_5
9. Marz, N.: Big Data: Principles and Best Practices of Scalable Realtime Data Systems. O'Reilly Media, [S.l.] (2013)
10. Cuzzocrea, A., Saccà, D., Ullman, J.D.: Big data: a research agenda. In: 17th International Database Engineering & Applications Symposium, IDEAS 2013, Barcelona, 09–11 October 2013, pp. 198–203 (2013)

11. Transaction Processing Council: TPC-H Benchmark (2013). http://www.tpc.org/tpch
12. Cuzzocrea, A., Moussa, R.: Towards lambda-based near real-time OLAP over big data. In: 42nd IEEE International Conference on Computers, Software and Applications, Tokyo, 23–27 July 2018
13. Gupta, A., Mumick, I.S.: Maintenance of materialized views: problems, techniques, and applications. IEEE Data Eng. Bull. **18**(2), 3–18 (1995)
14. Krishnan, K.: Data Warehousing in the Age of Big Data. Morgan Kaufmann, Waltham (2013)
15. Agrawal, D., Das, S., El Abbadi, A.: Big data and cloud computing: current state and future opportunities. In: Proceedings of the 14th International Conference on Extending Database Technology, EDBT 2011, Uppsala, Sweden, 21–24 March 2011, pp. 530–533 (2011)
16. Inmon, W.H.: Building the Data Warehouse. Wiley, New York (2005)
17. Transaction Processing Council: TPC-DS Benchmark (2013). http://www.tpc.org/tpcds
18. Nambiar, R.O., Poess, M.: The making of TPC-DS. In: Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, 12–15 September 2006, pp. 1049–1058 (2006)
19. Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E., O'Neil, P.: A critique of ANSI SQL isolation levels. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD, pp. 1–10 (1995)
20. Pritchett, D.: Base: an acid alternative. Queue **6**(3), 48–55 (2008)
21. Nguyen, T.M., Tjoa, A.M., Schiefer, J.: Towards the stream analysis model in grid-based zero-latency data stream warehouse. In: Professional Knowledge Management - Experiences and Visions, Contributions to the 3rd Conference Professional Knowledge Management - Experiences and Visions, WM, pp. 630–635 (2005)
22. Nguyen, T.M., Brezany, P., Tjoa, A.M., Weippl, E.R.: Toward a grid-based zero-latency data warehousing implementation for continuous data streams processing. IJDWM **1**(4), 22–55 (2005)
23. Doka, K., Tsoumakos, D., Koziris, N.: Efficient updates for a shared nothing analytics platform. In: Proceedings of the Workshop on Massive Data Analytics on the Cloud, MDAC, pp. 7:1–7:6 (2010)
24. Pereira, D., Azevedo, L.G., Tanaka, A.K., Baião, F.A.: Real time data loading and OLAP queries: living together in next generation BI environments. JIDM **3**(2), 110–119 (2012)
25. Dehne, F., Zaboli, H.: Parallel real-time OLAP on multi-core processors. In: Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2012, pp. 588–594. IEEE Computer Society (2012)
26. Dehne, F.K.H.A., Kong, Q., Rau-Chaplin, A., Zaboli, H., Zhou, R.: A distributed tree data structure for real-time OLAP on cloud architectures. In: Proceedings of the IEEE International Conference on Big Data, pp. 499–505 (2013)
27. Dehne, F., Zaboli, H.: Parallel real-time OLAP on multi-core processors. IJDWM **11**(1), 23–44 (2015)
28. Li, F., Özsu, M.T., Chen, G., Ooi, B.C.: R-store: a scalable distributed system for supporting real-time analytics. In: IEEE 30th International Conference on Data Engineering, ICDE, pp. 40–51 (2014)
29. Li, F., Özsu, M.T., Chen, G., Ooi, B.C.: R-Store - Source Code (2015). https://github.com/lifeng5042/RStore

30. Ferreira, N., Martins, P., Furtado, P.: Near real-time with traditional data warehouse architectures: factors and how-to. In: 17th International Database Engineering & Applications Symposium, IDEAS, pp. 68–75 (2013)
31. Ferreira, N., Furtado, P.: Real-time data warehouse: a solution and evaluation. IJBIDM **8**(3), 244–263 (2013)
32. Cuzzocrea, A., Ferreira, N., Furtado, P.: Enhancing traditional data warehousing architectures with real-time capabilities. In: Foundations of Intelligent Systems - 21st International Symposium, ISMIS Proceedings, pp. 456–465 (2014)
33. Cuzzocrea, A., Ferreira, N., Furtado, P.: Real-time data warehousing: a rewrite/merge approach. In: 16th International Conference on Data Warehousing and Knowledge Discovery, DaWaK, pp. 78–88 (2014)
34. Gupta, A., Yang, F., Govig, J., Kirsch, A., Chan, K., Lai, K., Wu, S., Dhoot, S.G., Kumar, A.R., Agiwal, A., Bhansali, S., Hong, M., Cameron, J., Siddiqi, M., Jones, D., Shute, J., Gubarev, A., Venkataraman, S., Agrawal, D.: Mesa: Geo-replicated, near real-time, scalable data warehousing. PVLDB **7**(12), 1259–1270 (2014)
35. LinkedIn: Pinot - A Realtime Distributed OLAP Datastore (2015). https://github.com/linkedin/pinot/
36. Yang, F., Tschetter, E., Léauté, X., Ray, N., Merlino, G., Ganguli, D.: Druid: a real-time analytical data store. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2014, pp. 157–168. ACM (2014)
37. Salem, K., Beyer, K., Lindsay, B., Cochrane, R.: How to roll a join: asynchronous incremental view maintenance. SIGMOD Rec. **29**(2), 129–140 (2000)
38. Quass, D., Widom, J.: On-line warehouse view maintenance. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD, pp. 393–404 (1997)
39. Agrawal, D., El Abbadi, A., Singh, A., Yurek, T.: Efficient view maintenance at data warehouses. SIGMOD Rec. **26**(2), 417–427 (1997)
40. Huyn, N.: Multiple-view self-maintenance in data warehousing environments. In: Proceedings of 23rd International Conference on Very Large Data Bases, VLDB 1997, pp. 26–35 (1997)
41. Krishnan, S., Wang, J., Franklin, M.J., Goldberg, K., Kraska, T.: Stale view cleaning: getting fresh answers from stale materialized views. Proc. VLDB Endow. **8**(12), 1370–1381 (2015)
42. Marz, N., Warren, J.: Principles and Best Practices of Scalable Realtime Data Systems. Manning, New York (2015)
43. Marz, N., Warren, J.: Big Data: Principles and Best Practices of Scalable Realtime Data Systems, 1st edn. Manning Publications Co., Greenwich (2015)
44. Kiran, M., Murphy, P., Monga, I., Dugan, J., Baveja, S.S.: Lambda architecture for cost-effective batch and speed big data processing. In: IEEE International Conference on Big Data, pp. 2785–2792 (2015)
45. Piekos, J.: Simplifying the (Complex) Lambda Architecture (2014). https://voltdb.com/blog/simplifying-complex-lambda-architecture
46. Roussopoulos, N.: Materialized views and data warehouses. SIGMOD Rec. **27**(1), 21–26 (1998)
47. Agrawal, S., Chaudhuri, S., Narasayya, V.R.: Automated selection of materialized views and indexes in SQL databases. In: Proceedings of 26th International Conference on Very Large Data Bases, pp. 496–505 (2000)
48. Aouiche, K., Jouve, P.E., Darmont, J.: Clustering-based materialized view selection in data warehouses. In: Proceedings of the 10th East European Conference on Advances in Databases and Information Systems, ADBIS, pp. 81–95 (2006)

49. Hose, K., Klan, D., Marx, M., Sattler, K.: When is it time to rethink the aggregate configuration of your OLAP server? PVLDB **1**(2), 1492–1495 (2008)
50. Cuzzocrea, A., Moussa, R.: Multidimensional database modeling: literature survey and research agenda in the big data era. In: IEEE ISNCC 2017, pp. 1–6 (2017)
51. Widom, J.: Integrating heterogeneous databases: lazy or eager? ACM Comput. Surv. **28**(4es), 91 (1996)