Real time data loading and OLAP queries: Living together in next generation BI environments

Diego Pereira, Leonardo Guerreiro Azevedo, Asterio Tanaka, Fernanda Baião

Department of Applied Informatics, Federal University of the State of Rio de Janeiro (UNIRIO)
NP2Tec (Research and Practice Group in Information Technology)
Rio de Janeiro, RJ, Brazil
{diego.pereira, azevedo, tanaka, fernanda.baiao}@uniriotec.br

Abstract. Real time ETL (Extraction, Transformation and Loading) of enterprise data is one of the foremost features of next generation Business Intelligence (BI 2.0). This article presents a proposal for loading operational data in real time using a Data Warehouse (DW) architecture with faster processing time than current approaches. Distributed processing techniques, such as data fragmentation on top of a shared-nothing architecture, are used to create fragments that are specialized in most current data and optimized to achieve real time insertions. Using this approach, the DW is updated near-line from operational data sources. As a result, DW queries are executed over real time data or very close to that. Moreover, real time loadings do not impact queries response time. In addition, we extended the Star Schema Benchmark to address loading operational data in real time. The extended benchmark was used to validate and demonstrate the efficiency of our approach, when compared to other in the literature. The experiments were performed in the CG−OLAP research project environment.

Categories and Subject Descriptors: H.2.4 [**Systems**]: Distributed Databases

Keywords: Real Time Data Warehouse, Business Intelligence 2.0, Database Distribution, Database Fragmentation

1. INTRODUCTION

Business Intelligence (BI) encompasses a collection of technologies, tools and practices for data warehousing, data mining, analytical processing, reporting and queries [Watson and Wxom 2007]. The goal of BI is to collect, integrate, cleanse and mine business information to assist experts making strategic decisions based on historical data stored in the data warehouse (DW) [Chaudhuri et al. 2001], [Dayal et al. 2009].

The emergence of new business models demanded significant changes to the original BI architecture, what has been called BI 2.0. One of the most relevant features of BI 2.0 is the need to provide real time access to operational data [Stodder 2007]. This requires that operational information is made available for analytical processing, typically supported by On Line Analytical Processing (OLAP) tools, as soon as possible, and differs from the traditional BI scenario where DW updates are performed periodically (e.g., at the end of the day) and out of business hours.

Real time access to operational data is fundamental to improve decision-making processes in organizations, especially in domains such as e-commerce, stock exchange and telecommunications. Moreover, with the globalization and the intensive use of Internet, critical businesses need their data sources available 24 hours a day, 7 days a week for decision making process. Thus, the time window for loading a DW is shortening. In this way, a major requirement for BI 2.0 success is to evolve the DW and the ETL process to support a continuous flow of data, decreasing any downtime.

This article proposes a physical DW architecture that applies horizontal data fragmentation techniques on the fact table to provide real time data loading. The goal is to allow data insertions in an acceptable time, reducing the impact on the performance of OLAP query executions. We extended the Star Schema Benchmark (SSB) [O'Neil et al. 2007], in order to evaluate the performance of real time queries. This proposal is part of the CG−OLAP project (Implementation and Performance Analysis of PC clusters using OLAP applications over Spatial Databases) at UNIRIO, which defined a low-cost, yet very efficient, architecture for processing OLAP queries. The architecture relies on the use of distributed and parallel techniques over a PC cluster. In [Castro et al. 2010], we proposed a tool to monitor network traffic in such environments.

This article is organized as follows. Sections 2 and 3 present concepts and existing approaches on real time DW. In Section 4, the proposed architecture is described. The performance evaluation results are presented in Section 5. Finally, Section 6 presents the conclusions and final remarks.

2. REAL TIME DATA WAREHOUSING

In traditional BI scenarios, ETL processes are executed against operational data sources in a daily-basis periodicity, and at off-hours, to load the Data Warehouse. Adapting this scenario to allow the use of real time data poses several problems. Jorg and Dessloch [Jorg and Dessloch 2009] formalized several anomalies that may arise from the use of traditional ETL jobs for near-real time data warehousing, since in this scenario changes to the operational sources and data warehouse refreshment may happen concurrently. Generically, considering real time data within an ETL process poses a number of challenges that may be grouped into: Gathering Data, Processing Data and Storing Data [Inmon et al. 2008], [Kimball and Caserta 2004].

- Gathering data usually occurs on top of transactional databases (the ones accessed by the main organizational operational information systems), which are optimized to perform large amounts of small writes in a very short time. The need to query those databases to obtain data during business hours, when their activities are running, may generate impacts on transactional applications response time, thus impacting business performance.
- Processing data comprise the execution of a number of data transformations techniques. However, current data transformation tools are built considering the traditional ETL architecture, where there is a small number of update executions (once in a 24 hour period, considering daily data refresh) against a large amount of data. In the BI 2.0 scenario, those tools must be adapted to execute constantly against small amounts of data.
- Traditional solutions for storing large amounts of data use a lot of techniques, such as indexes and materialized views, to decrease query response time against data warehouses. Those techniques directly affect insertions performance, which in the traditional scenario run in batches on off-hours. However, in BI 2.0 scenario, those insertions must be executed all time, making their performance a new point of concern.

Among the areas identified as potential problems in the scenario of real time data loading [Inmon et al. 2008], [Kimball and Caserta 2004], this article focuses on solving problems related to the storage of real time data in Data Warehouses. Gathering and processing problems are abstracted by using the Star Schema Benchmark extension, proposed in this article, to simulate the data insertion step.

3. RELATED WORK

Several proposals have been presented to address the storage of real time data in Data Warehouses. The proposals of Thiele et al. [thiele et al. 2007], Shi et al. [shi et al. 2009] and Martins [Martins 2009] focus on developing mechanisms to organize which operations should execute first on DWs. They let users choose which queries / inserts they want to be executed with higher priority, thus allowing

real time insertions to execute while degrading the performance of other operations. These proposals depend on regular user feedback about what is more important to be executed, which is a hard and continuous work.

The works of Tho and Tjoa [Tho and Tjoa 2004] and Doka et al. [Doka et al. 2010] propose inserting real time data into OLAP multidimensional cube structures, instead of into the DW 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 (which is the case of DW tables). The authors also argue that OLAP tools would access this data in the same way as if it were stored into the DW. However, other BI tools - such as dashboards or data mining applications - may not be able to access those structures, making real time data unavailable for them.

Inmon et al. [Inmon et al. 2008] defines the ability to perform online updates as one of the main innovations that a DW needs to address in order to fit into the BI 2.0 scenario. They propose that online updates should happen on a real time environment called interactive sector, in which data is stored following the application layer structure. This prevents transformations, thus reducing insertion execution time. Moreover, materialized views and indexes updates become less striking, as they occur in smaller amounts of data. Later, when it is desirable to execute more complex queries over real time data, it is integrated and moved to the integrated sector. This approach does not provide query and update transparency, since it is not possible that queries access both real time and historical data simultaneously and in a continuous basis.

Thomsen et al. [Thomsen et al. 2008] present a middleware proposal where data coming from operational sources is stored in main memory. It allows queries executing on historical data and real time data. When the amount of data reaches a threshold, an ETL process is executed to load main memory data into the DW. The proposal results in faster insertions than loading data directly on DW; however, querying main memory structures take longer than querying data on a common database. Also, this proposal uses a specific JDBC driver to provide user transparency, which was not developed to consider a distributed scenario.

Santos and Bernardino [Santos and Bernardino 2008] propose an adaptation to the DW model. New tables must be created with the same structure as those from the original DW, but without query optimizations mechanisms, such as indexes. These tables are added to the traditional model and become the destination of real time operational data. When real time data access performance becomes unacceptable, because of the lack of optimization mechanisms, they are extracted and loaded on the conventional DW tables. The drawback in this approach is that queries used by analytical processing tools need to be changed to consider both DW and new tables. This results in reconfiguration of dashboards, scorecards and cubes of business environments. These reconfigurations are very difficult to handle and error prone.

4. APPLYING DATA FRAGMENTATION TECHNIQUES IN REAL TIME DATA LOADING

This article proposes the use of data fragmentation and distribution techniques to allow real time loading without impacting queries response time. Also, query and update transparency are provided through the use of a Distributed Database Management System (DDBMS), a database management system in which data management functions is distributed over several nodes that are interconnected in a network [Risch 2009]. The use of a DDBMS that implements distributed concurrency control and transaction processing protocols also prevents the anomalies mentioned in [Jorg and Dessloch 2009].

The use of fragmentation and distribution techniques in data warehouses is historically related to database tuning. However, the distribution of the fragments usually takes query performance into account exclusively [Furtado 2004], generating scenarios where loading time optimization is ignored. With the need for real time load execution, the behavior of these insertions becomes a key issue in the whole performance of the DW and their applications.

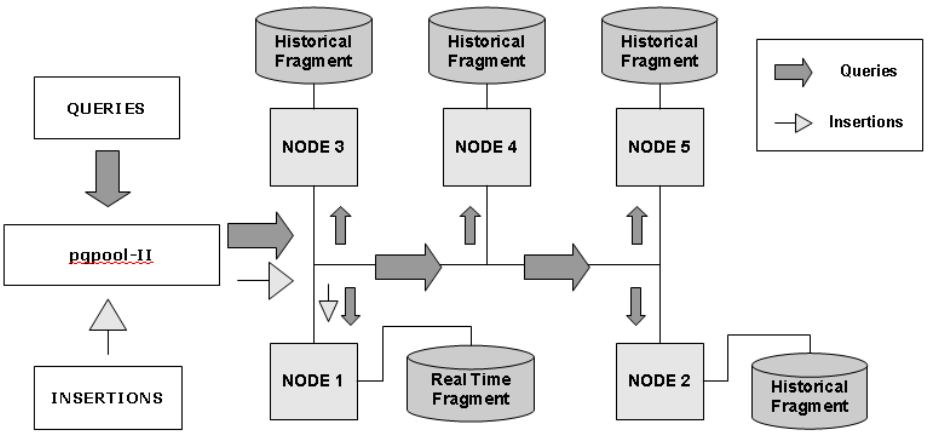

Fig. 1. Proposed Distributed Architecture

DW usually applies database optimization mechanisms to decrease queries response time, such as indexes and materialized views. These mechanisms need to be refreshed whenever an insertion happens on the DW, impacting the performance of insertion operations. We propose to execute these insertions in a table containing few tuples, where those mechanisms are unnecessary, while still working with previously inserted historical data in traditional fact tables. Hence queries can be executed with low response times, and insertions can be executed without the cost of refreshing the whole fact tables.

To achieve this goal, our proposal assumes a shared-nothing architecture where each processor has its own database, disk and memory areas [Ozsu and Valduriez 2011], and applies horizontal fragmentation techniques to create a fragment (called Real Time Fragment) where every insertion into the DW is executed. Horizontal fragmentation was preferred since it preserves distribution transparency, which is not the case with vertical fragmentation. The time dimension is used as a fragmentation key, performing a derived horizontal fragmentation of the fact table. Other dimension tables are replicated on all nodes. The proposal of this article was first presented by Pereira et al. [Pereira et al. 2011]; in the present article, we extend the description of our approach, present related work in the literature and compare to it, contextualize the work in the CG−OLAP institutional project, and detail the extensions proposed to the SSB Benchmark.

Each tuple inserted into the fact table represents one business transaction, and contains a date column that indicates the transaction execution time. When these operations are performed in real time, this column indicates the current date. As in the Star Schema specification, the time dimension stores date values, while the fact table has a foreign key pointing to it. Fact table fragments are created using values from the time dimension. Specifically, the Real Time Fragment is created using the current date; other fragments are created using different criteria, such as hash, round-robin or another date range partition. This proposal implements a balanced distribution of historical data among historical fragments. If a data range partition is applied, this may be considered an implementation of the data life cycle, as proposed in [Inmon et al. 2008]. After being created, these fragments are distributed over computer cluster nodes, in the same way as traditional fragment allocation is conducted.

Figure 1 illustrates the proposed architecture. Node 1 is responsible for real time insertions, while other nodes contain all historical data, distributed among them. A middleware works as a DDBMs and manages where data insertions and queries should be executed. All insertions are routed by this middleware to node 1 (the Real Time Fragment), while queries are partitioned and forwarded to all nodes. Thus, this middleware features query and update transparency to user data access, allowing this proposal to bypass the user data location awareness problem that happens in related approaches. Among available options, we chose pgpool-II as the middleware implementation in our experiments.

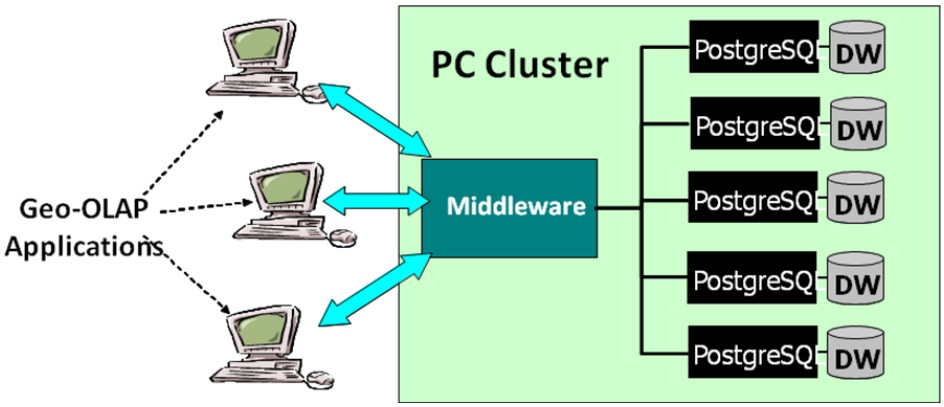

Fig. 2. CG−OLAP Cluster Architecture

As more insertions occur into the Real Time Fragment, its size continuously grows and its performance decreases over time. When a threshold is reached, data in this fragment is redistributed to the historical nodes. This operation is faster than an ETL process gathering data from operational data sources, because all data transformations have already been done in the Real Time Fragment. Thus, DW downtime for data redistribution is quite lower. An important issue for this approach is this threshold tuning, that is, finding the right moment to redistribute real time data among the historical fragments. Santos and Bernardino [Santos and Bernardino 2008] list several criteria that could be adopted for this adjustment: a maximum number of real-time rows, a maximum database size, every t time units, a maximum execution time for some query or an ad-hoc, when the database administrator (DBA) decides to do it. In our proposal, we adopt the query response time criteria. A pre-defined query Q chosen by the DBA (usually the most executed one) is periodically executed on historical data and on the Real Time Fragment. Data redistribution is fired whenever the second response time is greater than the first one (which means that the Real Time Fragment is delaying query execution).

5. EXPERIMENTAL EVALUATION

Experimental tests were executed to compare our proposal against a traditional scenario. Those tests include Star Schema Benchmark queries response times, load times, Real Time Fragment redistribution frequency and comparisons about data location awareness problems.

The experiments were performed in the context of the CG−OLAP project (whose architecture is illustrated in Figure 2), which investigates and proposes techniques and tools to process, in an efficient way, queries that involve massive data manipulation through relational database systems over PC clusters. Its cluster has five computers with Intel Core 2 Duo E6750 (2.66 GHz) and 2GB of RAM. All computers run OpenSUSE 10.3 operating system and are interconnected through a Gigabit network. The databases are stored in PostgreSQL 8.4 clustered through pgpool-II middleware set in parallel mode.

5.1 Star Schema Benchmark

The Star Schema Benchmark (SSB) [O'Neil et al. 2007] is an adaptation of the TPC−H benchmark for decision support systems, where the DW schema is converted to the dimensional model and the queries are adapted accordingly. Its data model has a fact table, called LineOrder, and four dimension tables: customer, supplier, part, and date. This benchmark is composed of 30 queries grouped into four categories, named Q1, Q2, Q3 and Q4. The SSB also adapted the TPC−H data generator, called DBGEN, to work with the dimensional model.

```
Q5.1
select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, date
where lo_orderdate = d_datekey and d_datekey = 19990101
and lo_discount between 1 and 3 and lo_quantity < 25;
```

```
Q5.2
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey
and lo_partkey = p_partkey
and lo_suppkey = s_suppkey and p_category = 'MFGR#12'
and s_region = 'AMERICA' and d_datekey = 19990101
group by d_year, p_brand1 order by d_year, p_brand1;
```

```
Q5.3
select c_nation, s_nation, d_year,
sum(lo_revenue) as revenue
from customer, lineorder, supplier, date
where lo_custkey = c_custkey and lo_suppkey = s_suppkey
and lo_orderdate = d_datekey and c_region = 'ASIA'
and s_region = 'ASIA' and d_datekey = 19990101
group by c_nation, s_nation, d_year
order by d_year asc, revenue desc;
```

```
Q5.4
select d_year, c_nation,
sum(lo_revenue - lo_supplycost) as profit
from date, customer, supplier, part, lineorder
where lo_custkey = c_custkey and lo_suppkey = s_suppkey
and lo_partkey = p_partkey and lo_orderdate = d_datekey
and c_region = 'AMERICA' and s_region = 'AMERICA'
and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
and d_datekey = 19990101
group by d_year, c_nation order by d_year, c_nation;
```

Fig. 3. Real Time SSB Queries

We extended the SSB towards supporting and evaluating Real Time DW behavior. Specifically, we inserted fact rows dating as of 01/01/1999 (considered the current date), in addition to the original period from 01/01/1992 to 12/31/1998. DBGEN was also configured to generate loading files using its original date range and new files with tuples on 01/01/1999.

Regarding data size, DBGEN has a parameter called scale factor which allows users to choose the amount of data it will generate. With a scale factor of 1, DBGEN generates about 6,000,000 fact table rows; with a scale factor of 10, it generates 60,000,000 fact table rows, and so on. In the experimental scenarios, the new fact table files for real time loading have about 0.1 % of fact table rows. This percentage was derived from the TPC−H refresh functions size.

This newly generated data can be loaded according to several strategies of Real Time Loading, such as Micro Batch, ETL Capture, Transform and Flow (CTF), Enterprise Information Integration (EII) and Enterprise Application Integration (EAI). Which strategy to use depends entirely on the load characteristics of the simulated scenario. Kimball and Caserta [Kimball and Caserta 2004] present a comparison between these strategies, classifying them by its reporting capabilities, such as historical data support, integration complexity and data freshness. They argue that Micro Batch ETL and EAI solutions are suitable for organizations facing data integration challenges, while CTF, EII and EAI are suitable for organizations expecting low latency reporting. This article chooses to use Micro Batch ETL strategy during the experiments, mainly because it allows the same level of integration and transformation as a regular ETL tool. This means that real time data reports have the same level of complexity as the historical ones.

Moreover, a new set of queries, named Q5 (Figure 2) was added to assess query response time on real time data. Such queries were derived from each of the four SSB query groups, generating queries Q5.1 to Q5.4, in order to represent the same level of analysis being performed on the most current data. So, every one of them has a where clause $d\_datekey$=19990101 as this date correspond to the current date in the benchmark. Since the real-time data volume is about 0.1% of the total data volume, the selectivity of the new queries is very high.

5.2 Comparison Procedures

First, DBGEN was executed using a scale factor of 100, resulting on 600,000,000 fact table rows and about 100 GB of data. This size was large enough to allow fragmentation impact analysis. Micro

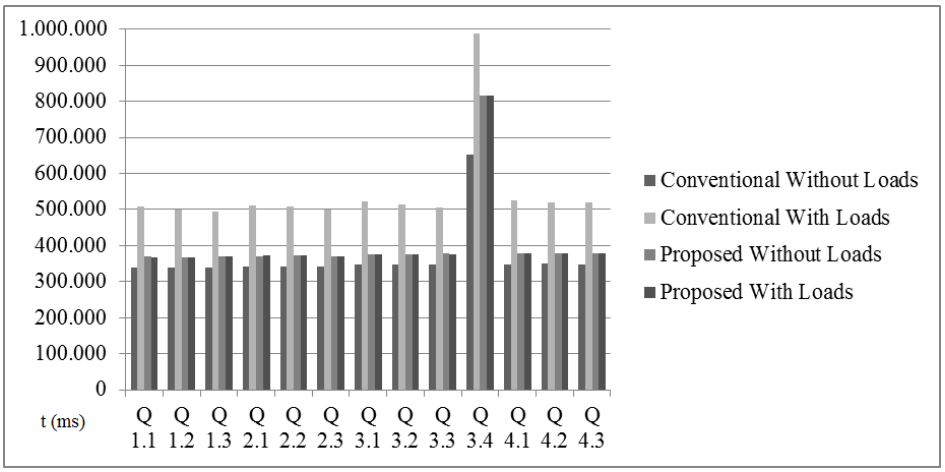

Fig. 4. Time Comparison over Fragmentation Scenarios

Batch ETL strategy was used to execute real time insertions. The performance results using our proposal of Real Time Data Fragmentation (section 3 - Fig. 1) were compared against the traditional approach using Furtado proposal [Furtado 2004], which is the only approach that already considered distribution and parallel techniques. The author proposed horizontal data fragmentation using a hashing algorithm applied to the primary keys of facts table. Data is randomly distributed and balanced among fragments. The insertion of new data occurs in the same way.

The comparison procedure was performed as follows: (i) Databases are loaded with data generated by DBGEN; (ii) Integrity constraints are activated; (iii) Database memory flush; (iv) SSB queries are executed without loads; (v) Database memory flush; (vi) Real time loads are simulated using files generated by DBGEN; (vii) SSB queries are executed simultaneously with load simulations.

An analysis about Real Time Fragment degradation over time was also made. This assess whether our proposal needs an excessive number of redistributions. At last, a comparison was made regarding the data location awareness problem of related approaches in the literature, presenting benefits that query and update transparency provides to our proposal. This is achieved comparing how Santos and Bernardino [Santos and Bernardino 2008] proposal accesses data over its Real Time DW.

5.3 Results

The first test considers the execution of SSB queries 1.1 to 4.3 [O'Neil et al. 2007]. Tests "without loads" illustrate cases where data loads and analytical queries executions are not concurrent. Tests "with loads" consider query executions concurrently with real time loading. Fig. 4 shows that the performance of traditional fragmentation ("Conventional without loads") was better (9.8% in average) than the proposed fragmentation ("Proposed without loads") when loads are not concurrently executed with queries. However, if data loading is executed concurrently (which is the case in the BI 2.0 scenario that we are addressing), query response time in the traditional fragmentation ("Conventional with loads") is in average 35% higher than in the proposed scenario ("Proposed with loads"). Moreover, the "Proposed with loads" execution times remained almost unchanged compared to the "Proposed without loads", thus evidencing that our proposal did not imply an overhead on query execution times, when data loading updates are executed concurrently. Query 3.4 presented a higher execution time when compared to others, following the original SSB specification in which it queries the total revenue considering all clients and suppliers in a specific month and geographic region.

The time required to execute loadings was also assessed. The longer loading time takes, more users will have to wait to analyze fresh data. The loading time of our proposal was 22m 27s, while the tradi-

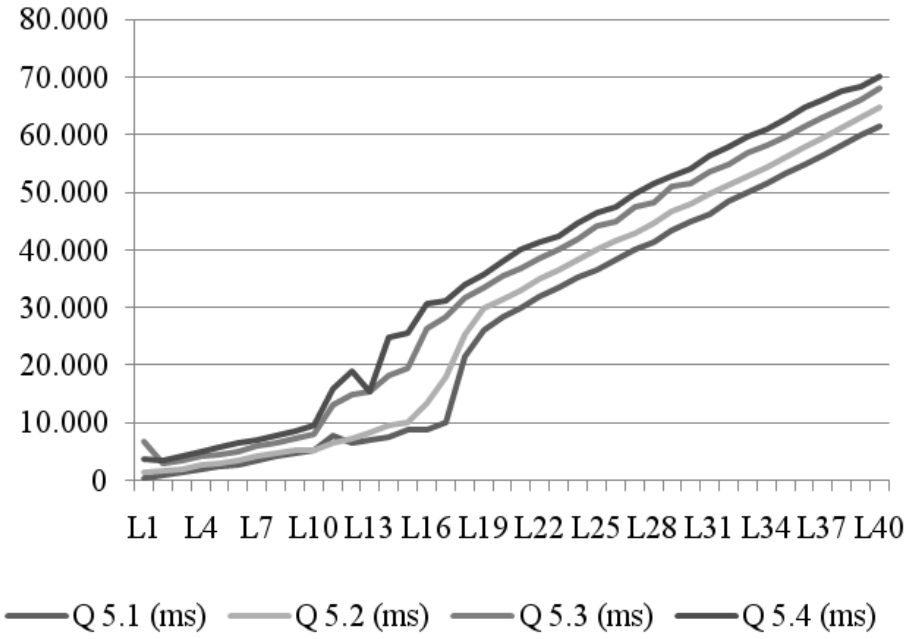

Fig. 5. Queries Degradation on Real Time Fragment

tional approach required 04h 55m 58s, which means that our approach consumes only approximately 8% of the traditional approach loading time.

Another important analysis is real time fragment degradation over time. This analysis was executed running benchmark loads and evaluating queries responses. The chosen queries were the new set of SSB queries proposed in Section 4.1 - Fig. 3, which access only real time data. Fig. 5 presents response times after every new load. As each benchmark load represents an increment of 0.1% in THE fact table, after forty loads the real time fragment has a number of rows equal to 4% of the original inserted fact table rows. Considering this volume of data, queries response time on real time fragment was about 70 seconds. This is still lower than the average response time presented on Fig. 4, which is more than 300 seconds. This demonstrates that the redistribution process does not need to occur frequently, which avoids impacts that this process causes on the DW.

Each load time was also measured in order to analyze whether the performance decreases when real time fragment size grows. Fig. 6 shows that the load time was not impacted by the size of the real time fragment, as it varied between 1290s (21m:30s) and 1440s (24m:00s) independently of the size.

Due to the use of a DDBMS, a consequence of this proposal is transparency in writing SQL statements. In other words, the user writes statements considering only one database, and the DDBMS is responsible for re-writing them to execute in the database cluster. To demonstrate this characteristic, we compared query Q1.1 from SSB against the same query when using Santos and Bernardino [Santos and Bernardino 2008] proposal. Fig. 7 presents the original query and its corresponding execution plan generated from PostgreSQL, while Fig. 8 presents the same query and its execution plan considering Santos and Bernardino approach. The query plan and its estimated cost show that complexity and cost increases significantly when employing Santos and Bernardino approach.

6. CONCLUSIONS

The broader use of BI tools has brought new challenges for their architecture. A relevant one is the need to consider real time operational data in OLAP analysis, thus requiring real time data loading

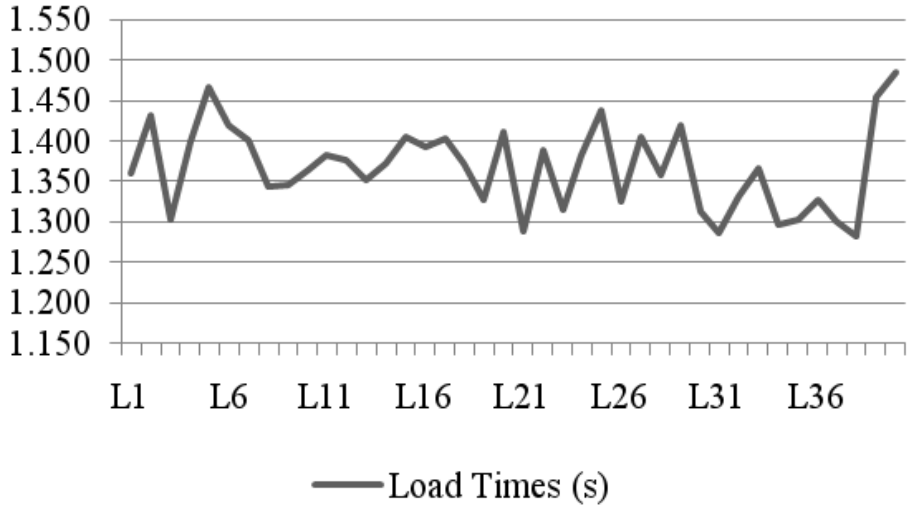

Fig. 6. Insertion Degradation on Real Time Fragment

ORIGINAL QUERY

```
select sum(lo_extendedprice*lo_discount) as revenue
    from lineorder, date
    where lo_orderdate = d_datekey
        and d_year = 1993
        and lo_discount between 1 and 3
        and lo_quantity < 25
```

ORIGINAL QUERY PLAN

```
Aggregate  (cost=4856615.10..4856615.11 rows=1 width=14)
 ->  Hash Join  (cost=86.51..4850599.51 rows=2406233 width=14)
      Hash Cond: (lineorder.lo_orderdate = date.d_datekey)
      ->  Seq Scan on lineorder  (cost=0.00..4766970.56 rows=15861362 width=23)
            Filter: ((lo_discount >= 1::numeric) AND (lo_discount <= 3::numeric)
            AND (lo_quantity < 25::numeric))
      ->  Hash  (cost=81.95..81.95 rows=365 width=9)
            ->  Seq Scan on date  (cost=0.00..81.95 rows=365 width=9)
                  Filter: (d_year = 1993::numeric)
```

Fig. 7. SSB Query Q1.1

SANTOS AND BERNADINO QUERY

```
select sum(lo_extendedprice*lo_discount) as revenue
    from ( select lo_extendedprice, lo_discount
            from lineorder, date
            where lo_orderdate = d_datekey
                and d_year = 1993
                and lo_discount between 1 and 3
                and lo_quantity < 25
          UNION ALL
          select lo_extendedprice, lo_discount
            from lineorderTmp, dateTmp
            where lo_orderdate = d_datekey
                and d_year = 1993
                and lo_discount between 1 and 3
                and lo_quantity < 25 ) as result
```

SANTOS AND BERNADINO QUERY PLAN

```
Aggregate  (cost=5139137.31..5139137.33 rows=1 width=64)
 ->  Append  (cost=86.51..5131120.21 rows=3206840 width=64)
      ->  Hash Join  (cost=86.51..4850599.51 rows=2406233 width=14)
            Hash Cond: (lineorder.lo_orderdate = date.d_datekey)
            ->  Seq Scan on lineorder  (cost=0.00..4766970.56 rows=15861362 width=23)
                  Filter: ((lo_discount >= 1::numeric) AND (lo_discount <= 3::numeric)
                  AND (lo_quantity < 25::numeric))
            ->  Hash  (cost=81.95..81.95 rows=365 width=9)
                  ->  Seq Scan on date  (cost=0.00..81.95 rows=365 width=9)
                        Filter: (d_year = 1993::numeric)
      ->  Nested Loop  (cost=0.00..248452.30 rows=800607 width=14)
            Join Filter: (lineordertmp.lo_orderdate = datetmp.d_datekey)
            ->  Seq Scan on datetmp  (cost=0.00..11.88 rows=1 width=32)
                  Filter: (d_year = 1993::numeric)
            ->  Seq Scan on lineordertmp  (cost=0.00..238432.84 rows=800607 width=23)
                  Filter: ((lineordertmp.lo_discount >= 1::numeric)
                  AND (lineordertmp.lo_discount <= 3::numeric)
                  AND (lineordertmp.lo_quantity < 25::numeric))
```

Fig. 8. SSB Query 1.1 (adapted according to Santos and Bernardino [2008] approach)

in DWs. Most approaches that handle real-time data storage in DW focus on separating insertions from queries on historical data.

In this article, we propose an approach that applies horizontal fragmentation to solve the problem of loading data in real time, as well as to minimize the impact of real time loading in the BI environment as a whole. An extension to the Star Schema Benchmark was proposed to evaluate the impact of our proposal. Results using the extended benchmark confirmed that using those well known techniques in a simple distributed DW architecture reduced the response time of data loading operations for about 13 times when compared to a traditional DW architecture. Also, queries executed 35.6% faster on

this proposal during loads executions.

Future work will perform further experiments comparing with related work and applying our proposal on real DW scenarios. Furthermore, we plan to evaluate different configurations of our fragment distribution strategy such as: allocation of more than one fragment at each node; consider more than one specialized node for insertions; distribution scenarios with fewer or more nodes. Also, evolving this proposal to avoid redistribution tasks is one of our focuses, which we plan to achieve by creating new Real Time Fragments every time the current one performance starts do degrade, instead of moving its data as current proposal does. This will also address a limitation of our current approach, which does not handle faults in the node hosting the real-time fragment.

REFERENCES

Castro, E., Diniz, M., and Baiao, F. Performance Evaluation of Distributed Databases in Clusters through Markov Models. In *Proceedings of the Brazilian Symposium on Databases - Demos Section*. Florianopolis, Brazil, pp. 1–1, 2010.

Chaudhuri, S., Dayal, U., and Ganti, V. Database technology for decision support systems. *Computer* 34 (12): 48–55, 2001.

Dayal, U., Castellanos, M., Simitsis, A., and Wilkinson, K. Data integration flows for business intelligence. In *Proceedings of the Intl. Conference on Extending Database Technology: Advances in Database Technology*. Saint-Petersburg, Russian Federation, pp. 1–11, 2009.

Doka, K., Tsoumakos, D., and Koziris, N. Efficient updates for a shared nothing analytics platform. In *Proceedings of the Workshop on Massive Data Analytics on the Cloud*. North Carolina, USA, pp. 1–6, 2010.

Furtado, P. Experimental Evidence on Partitioning in Parallel Data Warehouses. In *Proceedings of ACM Seventh International Workshop on Data Warehousing and OLAP*. Washington, D.C., USA, pp. 23–30, 2004.

Inmon, W., Strauss, D., and Neushloss, G. *DW 2.0 - Architecture for the Next Generation of Data Warehousing*. Morgan Kauffman, 2008.

Jorg, T. and Dessloch, S. Near Real-Time Data Warehousing Using State-of-the-Art ETL Tools. In *Third International Workshop on Enabling Real-Time for Business Intelligence*. Lyon, France, pp. 100–117, 2009.

Kimball, R. and Caserta, J. *The Data Warehouse ETL Toolkit*. John Wiley & Sons Inc, 2004.

Martins, D. B. *Extracao Personalizada e Incremental de Dados em Ambientes de BI Tempo Real*. Ph.D. thesis, Federal University of the State of Rio de Janeiro (UNIRIO), Rio de Janeiro, Brazil, 2009.

O'Neil, P., O'Neil, B., and Chen, X. The Star Schema Benchmark (SSB). http://www.cs.umb.edu/ poneil/StarSchemaB.pdf, 2007.

Ozsu, M. T. and Valduriez, P. *Principles of Distributed Database Systems*. Springer, 2011.

Pereira, D., Azevedo, L. G., and Tanaka, A. Real Time Loading of Enterprise Data Using Fragmentation of Data Warehouses. In *Proceedings of the Brazilian Symposium on Databases*. Florianopolis, Brazil, pp. 65–72, 2011.

Santos, R. J. and Bernardino, J. Real-time data warehouse loading methodology. In *Proceedings of the 2008 International Symposium on Database Engineering & Applications*. Coimbra, Portugal, pp. 49–58, 2008.

shi, J., Y, B., Leng, F., and Yu, G. Priority-Based Balance Scheduling in Real-Time Data Warehouse. In *Proceedings of the 2009 Ninth International Conference on Hybrid Intelligent Systems*. Shenyang, China, pp. 301–306, 2009.

Stodder, D. Good BI, Cruel World? *Network Computing* 18 (6): 56–66, 2007.

thiele, M., Fischer, U., and Lehner, W. Partition-based workload scheduling in living data warehouse environments. In *Proceedings of the ACM tenth international workshop on Data warehousing and OLAP*. Lisboa, Portugal, pp. 57–64, 2007.

Tho, M. N. and Tjoa, A. M. Grid-Based Zero-Latency Data Warehousing for Continouos Data Streams Processing. In *Proceedings of the 6th International Conference on Information Integration and Web-based Applications & Services*. Jakarta, Indonesia, pp. 777–786, 2004.

Thomsen, C., Pedersen, T., and Lehner, W. RiTE: Providing On-Demand Data for Right-Time Data Warehousing. In *Proceedings of the IEEE International Conference on Data Engineering*. Cancun, Mexico, pp. 456–465, 2008.

Watson, H. and Wxom, B. The Current State of Business Intelligence. *Computer* 40 (9): 96–99, 2007.