Near Real-Time with traditional Data Warehouse Architectures: Factors and How-to

Nickerson Ferreira
DEI - FCT
University of Coimbra
Coimbra, Portugal
nickerson@dei.uc.pt

Pedro Martins
DEI - FCT
University of Coimbra
Coimbra, Portugal
pmom@dei.uc.pt

Pedro Furtado
DEI - FCT
University of Coimbra
Coimbra, Portugal
pnf@dei.uc.pt

ABSTRACT
Traditional data warehouses integrate new data during lengthy offline periods, with indexes being dropped and rebuilt for efficiency reasons. There is the idea that these and other factors make them unfit for realtime warehousing. We analyze how a set of factors influence near-realtime and frequent loading capabilities, and what can be done to improve near-realtime capacity using a traditional architecture. We analyze how the query workload affects and is affected by the ETL process and the influence of factors such as the type of load strategy, the size of the load data, indexing, integrity constraints, refresh activity over summary data, and fact table partitioning. We evaluate the factors experimentally and show that partitioning is an important factor to deliver near-realtime capacity.

Keywords
ETLR, Data Warehouse, near real-time.

1. INTRODUCTION
Currently, many industries require the most current information possible in their data warehouses. There is an increasing need to have always up-to-date information in systems used not only for strategic long-term planning but also for quick operational decisions, sometimes in near-realtime.

The traditional DW assumes periodic offline loads with heavy operations during dead periods, and those loads do not fit well with simultaneous analysis and mining. The offline loading is frequently defined as daily, weekly or monthly, during overnight periods. Given the market need for up-to-date information in some domains, if possible in real time, questions arise such as: "Does a traditional DW architecture have the capability to achieve a context of near real-time? What are the factors involved? How can we achieve the near-realtime capability?".

In order to answer these questions, this article studies the main factors that affect the capacity. Extraction and transformation is done in a staging area in its own dedicated machine, while loading and refreshing of aggregated data directly impact queries performance and vice-versa. We analyze the influence of the extraction and processing of the source data, loading and refreshing of information into the DW, as well as the influence of indexing and queries being performed simultaneously. Through this work we will analyze the main reasons why a traditional DW has difficulties in supporting updates in real-time. Data loading throughput is drastically reduced if indexes are not dropped and/or when queries run simultaneously (online loading), which significantly affects near-realtime capabilities. Likewise, query performance is affected significantly if loads are done online. But we will also show that, although online loading is quite complicated in terms of performance for both loading and queries, it is possible to have near real-time if we configure the database and loading process in a way that reduces index rebuilding overheads. With the right solution, we can for instance have offline periods of 10 to 20 seconds every 10 minutes, loading small batches and only impacting performance for those small offline periods.

We will identify the factors that influence the near real-time capacity and present the results of experiments that simulate a traditional DW with ETL, in particular with the process of updating stars' data and aggregated data views.

The rest of the paper is divided as follows: Section 2 presents related work, in Section 3 we review the processes of a traditional DW, and Section 4 discusses the issues and factors that the traditional DW faces in near real-time contexts. Section 4 is responsible for addressing the main factors that exert significant influence on the process of maintaining the DW. In section 5 we analyze the results obtained in the tests with graphs showing the cost of operations or throughput measured in lines per second. Finally, in Section 6 we have the final considerations, where we conclude the article.

2. RELATED WORK
Related work in this subject includes studies concerning data warehouses realtime capabilities, benchmarking and refresh performance for data warehouses.

RTDW-bench [2] is a benchmark proposed to evaluate the performance of an RTDW system. The authors use TPC-H to assess two main factors: data arrival frequency that the system is able to meet to update its tables without delays, with and without refreshing materialized views. Comparing to our work, the main difference of [2] is that the authors focus on maximum loading rate solely, do not consider query workload execution (online loading), or a set of other factors that we do consider, including batch sizes and loading strategy, index rebuild, fact table partitioning with local indexes. Additionally, we consider the whole Transformation process in (ETL), not only the loading. Our work evaluates the influence of several factors on near real-time processes that are not studied in [2].

IDEAS'13, October 09-11 2013, Barcelona, Spain

http://dx.doi.org/10.1145/2513591.2513650

In [3] the authors propose a benchmark to compare performance of ETL systems. They recommend a set of factors to be evaluated, for instance, most common transformations, error processing, security of the ETL process and scalability. They also propose two performance assessment alternatives: response time and throughput. The authors themselves inform that the benchmark is not appropriate for near-realtime, since it is batch-oriented.

The paper [11] focuses RTDW and ETL optimizations for near-real-time. The authors study the best way to optimize loading to make the data as fresh as possible. Using TPC-H as a basis, the authors study performance of data loading, propose a change in architecture and evaluate the impact of queries on loading efficiency. In this case the authors propose a modification of the traditional architecture, by means of an addition, and the focus is on optimizing data loading only, while we study several other factors.

3. ARCHITECTURE AND BASE PROCESS OF A DATA WAREHOUSE

In this section we review the architecture of a traditional data warehouse system, as well as the processes that are used for data maintenance and updates. This will allow us to identify the major factors influencing the processes. The base architecture includes a set of stars, containing data from different business perspectives. The data warehouse can also contain a large set of aggregated data views, with the objective of offering varied data perspectives and to reduce the time needed to provide query answers.

Figure 1 shows data extraction, the process of retrieving new data from data sources (E), and sending it to the staging area. We can assume that the sources are either databases or file data in different possible formats (e.g. XML, CSV). Transformation (T) cleans and transforms the data to make it consistent with the star schemas. With the data already adequately formatted, it is loaded (L) into the star schemas. For convenience, we separate loading (L) from refreshing of aggregate data (R), and where appropriate also from index rebuilding. The complete process is therefore called ETLR (extract, transform, load, refresh). Besides these processes, a data warehouse should be ready to accept a set of sessions submitting queries, also known as the query workload (QW).

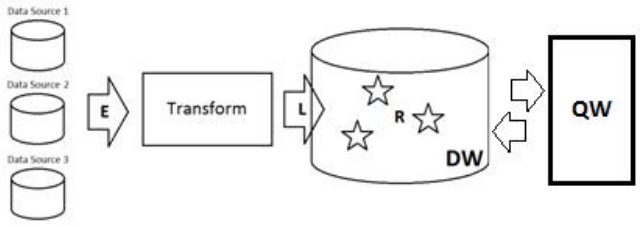

Figure. 1. Traditional DW architecture.

Data loading implies insertion of the data into the star schemas. The existence of indexes delays the insertion processes significantly, due to the need to insert also into the indexes and the need to restructure the indexes. For this reason, it is common to remove the indexes at the start of the loading process and to recreate them only after this task and the refreshing of the data warehouse data are done. If we consider online loading (loading while at the same time processing queries), then it is infeasible to drop indexes before the loading and to recreate them after the loading. One alternative that might be acceptable depending on the application is to have reasonably small offline periods during which we drop, load and rebuild indexes.

Foreign keys typically link the fact table to dimension tables. They serve mainly for referential integrity checking, i.e., to make sure that a row inserted into the table corresponds to a row in the table referenced by the foreign key (there is a matching key value there). But since referential checking using foreign keys can be quite costly for each newly inserted row, many data warehouse implementations do not define the foreign keys, and instead assume that referential integrity is checked only at the staging area, during transformation. For this reason we consider the DW with and without foreign keys.

4. NEAR REAL-TIME FACTORS

In this section we present the main factors that may influence near realtime capabilities in the traditional data warehouse architecture, given the fact that the data integration process has a high cost in those systems.

In a traditional DW data integration occurs with the system offline, at dead hours (overnight), and a supposedly very large set of data coming from different data sources is extracted, transformed and loaded. After transformation, that data is stored in batches for loading. When one wants to get to near-realtime capabilities, this is done by reducing the interval between execution of two consecutive ETL processes, for instance, they can be run with a 5 minutes interval between them. The size of the batches is therefore smaller, but the cost of the ETL process is high even when runs offline. In a near-realtime context this process runs with a much higher frequency, over much smaller batches and most probably while the system remain online and serving queries. Another factor is the dropping and recreation of indexes, which is not possible in a near-realtime system if it stays online, and may incur in unacceptable delays if the system is taken offline to allow it. As a consequence, an online near-realtime system is going to be quite slow inserting new data.

The various factors are discussed next in more detail.

4.1 Data Loading Method and Batch Size

When considering near-realtime, the data loading method is a relevant factor in terms of performance and consequences. The incoming data can be inserted tuple-by-tuple as soon as transformation ends, or a set of tuples can be collected into a batch of memory to insert them in a batch insert (e.g. jdbc batch insert), or a CSV batch file can be created and then loaded using the bulkload mechanisms that the database servers typically provide. Bulk loads using such tools are typically optimized for loading speed of large quantities of data, and committed only after a large number of rows have been inserted. Batch inserts from a memory batch spares writing to disk and also commits only after the insert, but a jdbc batch insert is not as efficient as bulkload when there are large amounts of data to be loaded. Tuple-by-tuple insertion is of course the least efficient, since the command is submitted and parsed for each row that needs to be inserted.

In general, and up to some size, larger batches are loaded much more efficiently than the same size of data loading in small batches or in tuple-by-tuple fashion. On the other hand, smaller batches can refresh the DW with current data faster. Optimization of the loading process may result in a better near-realtime capability for the DW.

In terms of evaluation, we can use different batch sizes to test data loading throughput and the influence of batch sizes in the total

time and throughput of the ETLR processes. In the experimental section we tested sizes from 1 row at a time to 100 million rows, in order to evaluate the influence of this factor.

4.2 Query Sessions

A DW is created to allow users to do their analysis over historical data, sometimes also over current data, in order to help them to take decisions. Query sessions are an important factor in the evaluation of the near-realtime capabilities of a data warehouse architecture. A near-realtime DW may be expected to be always online, which means to allow queries to be submitted while data is loading. Query sessions will interfere with loading and vice-versa, causing significant performance degradation in both. We will experiment with those to reach conclusions concerning how queries affect the ETLR process and how load and refresh operations affect queries performance.

4.3 Indexing

Indexes can provide significant speedups for some query patterns. However, they also degrade insert, update and delete performance. Two typical types of indexes are b-trees and bitmap indexes. The B-tree has a tree of nodes structure, where each node contains Keys and pointers to lower-level nodes, until the leaf nodes that include Keys and row identifiers for the corresponding rows in the tables. Operations such as insertions into tables also require modifications (insertions) in the indexes that are associated with that table, and sometimes those modifications involve significant restructuring of parts of the tree [5]. Bitmap indexes [6], which achieve a high level of compression, are preferably used in DWs, rather than in Online Transactions Processing Systems (OLTP), since they usually incur in high insert, update and delete overheads.

Due to the high costs associated with operations such as recreation of indexes, they are an important factor that should be assessed when evaluating near-realtime capabilities of a data warehouse. If indexes are dropped prior to loading and rebuilt afterwards, the cost associated with rebuilding of the indexes is quite high. But if the data is loaded into the data warehouse without dropping indexes, loading time will suffer significantly. In our experimental setup structures we included b-tree and bitmap indexes, and we considered scenarios in which we recreate or load online, in order to test the factors associated with indexes.

4.4 Referential Integrity using Foreign Keys

Foreign-key constraints are enforced to provide referential integrity checks. For instance, the database engine is supposed to test, for every insert, whether the value of the foreign-key attribute exists in the referenced table. This mechanism is also responsible for performance degradation during data loading, since the database engine is supposed to check the constraint for each row that is inserted, typically by searching the key in a primary key index [7]. To speedup this process, referential integrity can be checked in the transformation phase in the staging area. We evaluate this factor, such as how much it influences the loading performance.

4.5 Refresh of Aggregated Summaries

Refreshing aggregated data is one of the last tasks in the ETLR process. Aggregated data may exist in the form of materialized views, e.g. in Oracle [12], or other forms of summarized, pre-computed data that needs to be refreshed after (or during) the loading process. As with the rest of the loading process, refreshing may affect query performance and query sessions may also affect the refresh performance, due to concurrent execution of the tasks.

4.6 Table Partitioning

When we consider big tables, we must always take into account the cost of querying and index creation over such big data. Partitioning [8] offers a simple approach to restrict the amount of processing that is necessary in some queries or index operations to some partitions. Figure 2 shows an example of partitioning. In a data warehouse the data can frequently be partitioned by time intervals, so that queries accessing only recent data or a specific interval of time may be answered much faster. Even more importantly from the point of view of the ETLR process and its interaction with query processing, instead of dropping and rebuilding indexes for the whole data, it is possible to keep local indexes (indexes for each individual partition) and to drop and recreate indexes only in the last time interval partitions, which are the ones being loaded. The potential of this option is very important, since it removes an extremely large overhead of index re-creation that would be incurred over the whole table.

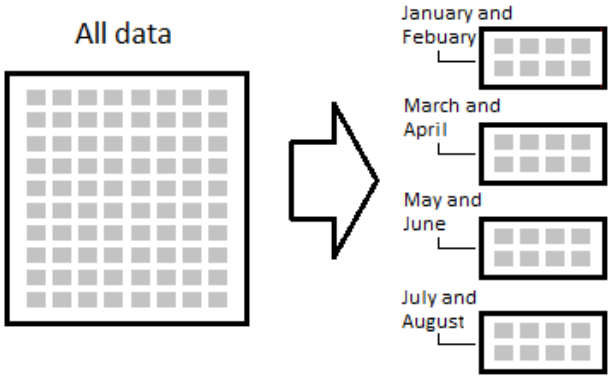

Figure 2. Data partitioning.

Using partitioning, we are able to optimize the time needed for some operations, in particular the index re-creation step of ETLR. Indexes are re-organized only in the partition(s) that was inserted (the last partition(s)). For instance, in a 100 GB table with 10GB partitions, the indexes are only dropped and re-created in one 10GB partition, which saves significant amounts of time to the ETLR process. By saving so much time, it becomes possible in some deployments to shorten loading times to small fractions of the online time, therefore to consider near-realtime with short off-line periods for loading.

5. EVALUATION OF FACTORS

This section details the experimental setup and analysis the factors that influence performance when we try near-realtime loading over a traditional data warehouse architecture.

We have chosen the structure and query workload of SSB (Star Schema Benchmark) [11], as our data model for the tests, but we changed the SSB so that we would be able to test the near-realtime characteristics. We added star schema and typical data warehouse data integration and refreshing elements. We named this version the SSB-RT, as proposed in [13]. Figure 3 shows the database schema of SSB-RT. The SSB is derived from the TPC-H, a data warehousing benchmark of the transaction processing council (TPC). The SSB was created to represent a more typical star schema than that of the TPC-H data model. In SSB-RT the data model is composed of two main stars and a number of aggregated views: A) the Orders star, representing sales, and the

corresponding dimension tables (Time_dim, Date_dim and Customer); B) a star representing LineOrder, which contains order items, with Five dimension tables (Time_dim, Date_dim, Customer, Suppllier and Part).

The aggregated views are 12, and they measure sales by Supplier, by Part, by Customer and according to various aggregations (hour, day, month and year). SSB-RT includes auxiliary temporary tables, one per fact, which are used to load the incremental data for computing refresh summarized data from the current aggregated views and the incremental data.

The initial data loading for this data schema was made using the data generation utility DBGEN of TPC-H, creating the necessary load files. These simulate data extracted from data sources. The data is loaded and then transformed to the format required by the star schemas. This transformation step is done through a Java application. The resulting data set is then loaded into the database using alternative approaches that were described in the previous section. The initial size of the database is 30 GB. After the load, aggregated views which contain the aggregated data per periods of time (hour, day, month, year), are refreshed with the new data. There are 12 materialized views which are refreshed. Indexes can be dropped and re-created in the ETLR process, to speed up the loading. In this experimental setup we created 12 B-tree indexes, 6 bitmap indexes and 8 foreign keys.

With SSB-RT, we are able to analyze the factors that influence system performance, including data integration, possibly in near-real-time, indexing, view refreshing, effects of different sizes of logs to be loaded, loading periodicity, while at the same time having query workloads being submitted through client sessions. This allows us to observe how the factors correlate and influence each other.

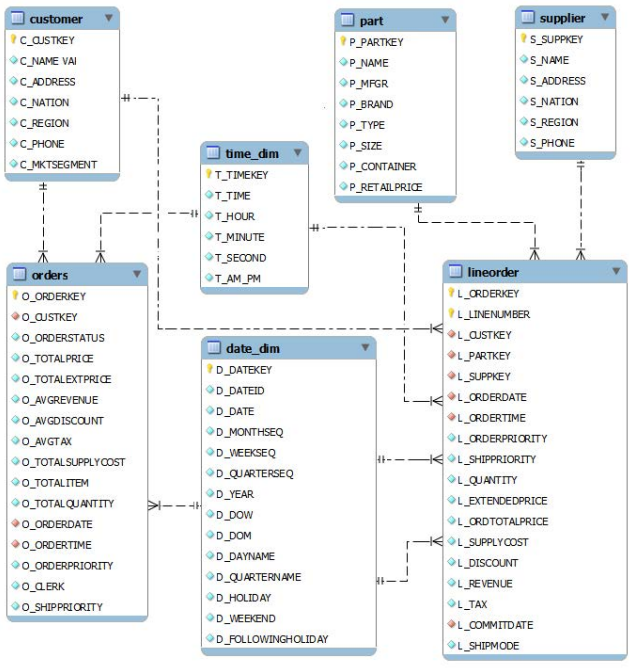

Figure. 3. Data Model of SSB-RT.

We used two computers for these experiments. One contained the staging area, executing a Java application that did all transformations from the log with data extracted from the data sources, while the other one had the database engine of the data warehouse. The computer used by the Java transform application was an Intel(R) Core(TM) 2 Quad 2.5GHz with 4GB of RAM memory. The database Server was an Intel(R) Core(TM) i5 3.40GHz with 16GB of RAM memory, Oracle version 11g.

5.1 Impact of Online Load and Refresh in the Query Workload Execution Time

In this section we analyze the impact of Loading (L) and Refreshing (R) while the system is "online", i.e., while it is answering queries. A query workload is submitted continuously, and we evaluate the impact in query response time. One of the tasks (L) or (R) is done constantly in cycle, and each query from the query workload (13 queries) is ran 15 times to obtain a statistic measure of the response time from the series (the 5 values with largest offset from the average are discarded, then the average is taken). The standard deviation was always less than 5%. Figure 4 shows the results. We can see that there is a significant impact in query response time.

By analysis of the results obtained in the experiment, it is possible to conclude that the queries suffer a significant performance loss, of 44% in average, when they run concurrently with the loading process (minimum loss of performance is 7% with query Q3, maximum loss is 111% with query Q5). The loss of performance due to refresh was 35% in average (minimum 8% for query Q10, maximum 51% for query Q8).

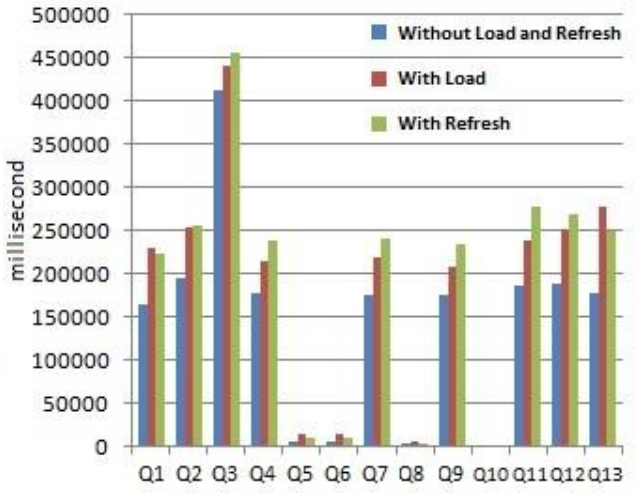

Figure. 4. Impact of Load and Refresh in Query response Times.

5.2 Analysis of the Throughput for the ETLR Process (Online versus Offline)

Some of the factors that influence ETLR performance include the size of the batch to be processed through ETLR, and the queries that are executed in concurrent sessions against the database during that ETLR process. The analysis that we do in this section presents the ETLR throughput (rows/sec) in different circumstances. We compare doing it offline dropping and re-

creating indexes, offline without dropping and recreating indexes and doing it online. The online case concerns running ETLR while at the same time having ten threads submitting queries randomly selected from the query workload, one query at a time. The objective of running this "online" case is to evaluate the influence of running queries simultaneously with the loading and refreshing of the data. The experiment compares these cases against the size of the batch file with the source data. This way we are also analyzing how the ETLR throughput varies as we increase the size of the batch that is to be loaded, from a few rows (2) to 10 million rows. The three scenarios are therefore: I) Offline with indexes and keys; II) Offline recreating (redo) the indexes and keys; III) Online with 10 simultaneous query sessions.

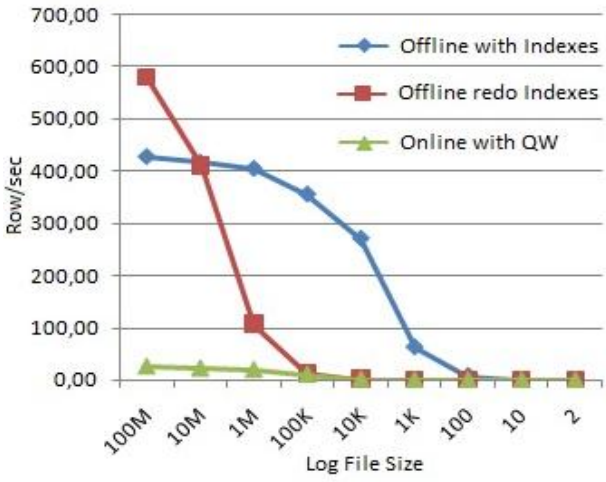

Figure. 5. ETLR throughput with log size, online and offline.

From the results, it is possible to see that the throughput of the Offline scenario with indexes exhibits a performance that is better than the Offline scenario with indexes recreation when the size of the batch is lower than 10 M. This is actually due to the extremely high overhead of recreating the indexes for the 30 GB data set in the "Offline redo Indexes" approach, which only paid-off for very large loads (more than 10M rows). This means that, even though the loading itself is much faster in the "Offline redo indexes" scenario, for relatively small-sized batches the overhead of re-creating indexes offsets the increase in loading speed. This and the ETLR versus batch size results in the chart also confirm that the traditional approach followed by data warehouses is efficient provided that the amount of data to be loaded is large.

The worst results were those obtained for the scenario "Online with QW" (queries running concurrently). The queries have caused a large overhead, especially in the loading and refreshing of the data. It is clear that the queries have priority in resource consumption, affecting the loading performance drastically. This is a very relevant problem with near-real-time in traditional data warehouses. Besides, this overhead can increase even more as more query sessions are added (results in next session), and causes instability in performance of the ETLR process if it is run online.

5.3 Impact of Query Sessions in the Performance of the ETLR process

The configuration for this experiment was quite straightforward: the benchmark was setup to load a log with only 10 rows of data (we had to load only 10 rows for comparison purposes, since the amount of time to load data with 50 or 100 query sessions was extremely high), and query sessions submitting randomly chosen queries continuously. The first result is with zero sessions, indicating the performance of ETLR running alone (offline). The same setup was ran with increasing numbers of query sessions. The number of sessions tested was: 5, 10, 50 and 100. Figure 6 shows the time taken by the ETLR process for each case.

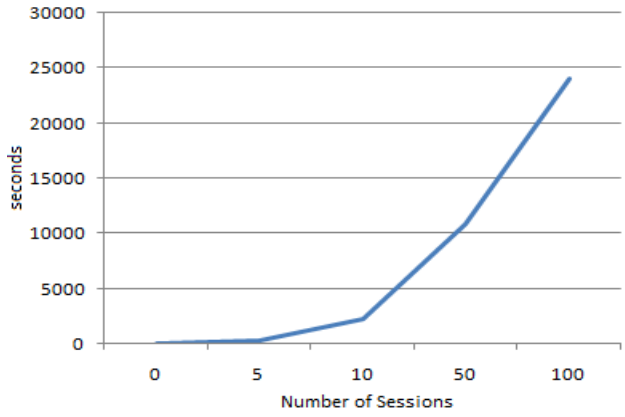

Figure. 6. Evaluation of the impact of simultaneous number of sessions over the ETLR process.

From these results it is clear that the impact of multiple query sessions on the ETLR of even only 10 rows is very large. While without queries the system was able to end the ETLR process in only 12 seconds, this value was raised to 291 seconds with only 5 sessions, an increase of approximately 2425%.

5.4 Impact of Data Loading Strategy in the ETLR process (Offline)

As the loading frequency gets nearer to near-realtime, the number of rows to load per load decreases, up to the point of a row-by-row loading. The relationship between the size of the batches to be loaded (mini-batches), the mechanism to load (bulk load, batch jdbc inserts or row-by-row insert commands), and performance is important in near-realtime systems. For instance, if we are to load each arriving row immediately, then using a simple insert command is faster than creating a micro-batch file with a single row and loading it using a database loading tool, due to the unnecessary overhead of the latest alternative. However, if there are many rows to be loaded, this second alternative is more attractive than row-by-row insert commands. The objective of this experiment is to identify the best loading strategy for optimizing the ETLR process, depending on the mini-batch sizes. For this experiment we have ran the whole ETLR process without dropping indexes from fact tables and without simultaneous query sessions. We tested the three strategies – bulkload, batch jdbc inserts and single row inserts, on the Oracle database used in the experimental setup. In Figure 7a we show the throughput of loading only (L), measured in number of rows loaded per second, when loading 10M rows was very small when row-by-row inserts were used (39 rows per second), compared with 982 with batch inserts of 1000 rows at a time and 20000 with bulkloads. Figure

7b compares bulk loads with batch inserts as the size of the batch (log file size) increases. This result shows that bulk loads become more efficient than batch loads for batches larger than 1k rows.

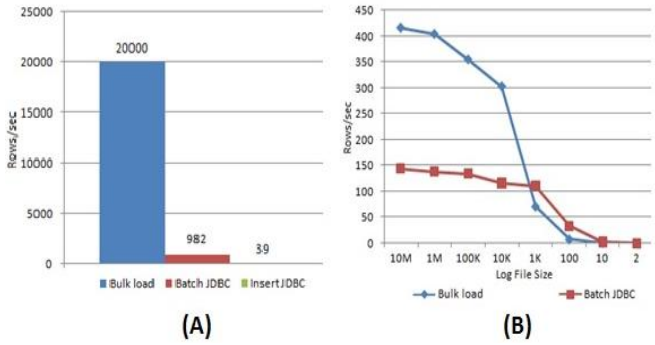

Figure. 7. (A) shows the throughput of each loading strategy. (B) shows the total throughput of the ETLR process using bulk load and batch jdbc methods with various log sizes.

The conclusion from these experiments is that loading based on bulkloads is the Best option when batches are large (larger than 1k rows). Below 1k rows, batch jdbc is slightly faster.

5.5 Analysis of the Cost of each Activity of ETLR (Offline)

In this section we analyze the cost of each step of the ETLR process. The parts are: Extraction (which we do not evaluate here), transformation and creation of the loading file (E+T), loading (which includes loading of auxiliar tables and loading of stars), refresh of aggregated data and recreation of indexes, in case the indexes had been dropped prior to loading for efficiency reasons. For efficiency reasons of online execution of the processes, the staging area and transformation reside in a different machine from the data warehouse, with the loading process also running in the data warehouse machine. We consider the following cases concerning the loading of stars: without indexes, with indexes and no foreign Keys, and with both indexes and foreign keys.

For these tests we had to create an initial configuration of the database. The dataset size is 30GB and is assumed to reside in a single partition (in the following section we will analyze the case where we partition the data to render recreation of "local" indexes less time consuming). The main objective of these experiments is to evaluate the impact of the following factors in ETLR performance:

- Index recreation: we analyze the impact of index recreation in the test scenarios that use this procedure;
- Foreign keys: it is tested how much the recreation of foreign keys affects performance;
- Loading (L-Star) with/no indexes, with/no foreign keys;

These experiments were ran with the batch size of 10M rows, the benchmark being ran to obtain the execution time for each scenario. Figure 8a shows only the time taken to load the stars from the batch of already transformed data (loading times). Figure 8b shows the execution time of the whole ETLR process, also detailing the fraction of time taken by the following parts: E+T, Load, Refresh and redo indexes.

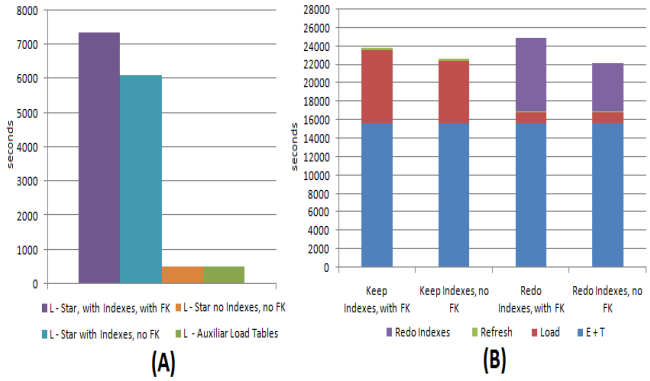

Figure. 8. (A) shows loading times for different configurations concerning whether indexes and foreign keys are there or not. (B) details the weight of each factor in ETLR processing time, for each scenario of (A).

These results show that the main factors influencing the performance of the ETLR process in this benchmark were (E + T) on one hand, and either loading, if indexes are not dropped and recreated, or indexes recreation, if indexes are dropped before loading and recreated afterwards. For this setup and size of the log file, it was clear that the whole ETLR process with index dropping and recreation was as slow as the same process while keeping the indexes. This agrees with the results in Figure 5, where it can be seen that the performance of the two alternatives for 10 M rows was similar, and that index dropping and recreation becomes more favorable only for larger data loads.

Not using foreign keys in the star schemas has some influence in loading performance and index recreation times as well. Without foreign Keys, loading was 16 minutes faster (for a total of about 2 hours), and index recreation was 45 minutes faster (for a total of 2.2 hours).

5.6 Partitioning for Near-Realtime with Offline Periods

The analysis of the previous experiments has shown that there is a very significant degradation of both loading and query performance when we try to load incremental data while the system is kept online. The near-realtime option should therefore be to try to have small offline periods for loading, instead of keeping it online always. If those offline periods happen to be sufficiently small and also a small fraction of the total time, near-realtime becomes feasible in scenarios where the lack of response for a small amount of time may be acceptable.

Since extraction, staging and transformation can be done in different machine(s) form that of the data warehouse and we have to assume that these processes are sufficiently fast to supply the ingress data at the desired rate, we concentrate in loading, refreshing and index re-creation. Indexes are dropped prior to loading, and re-created after loading, to allow very efficient loading. The key to efficiency becomes to minimize index re-creation times, and this is achieved by organizing data in time interval partitions and limiting drop and recreation to only the most recent data partition, where the new data was loaded (or to a small number of partitions in case the loaded data spans more than one partition). For the experiment, we configured the fact table as time-interval partitioned, with a partition size of 1 GB, comparing with the previous scenario of 30 GB partition. The benchmark was ran, loading a batch file of 10 M rows, and two scenarios

were tested: I) recreating indexes and foreign keys; II) recreating only indexes, no foreign keys. The results can be seen in Figure 9.

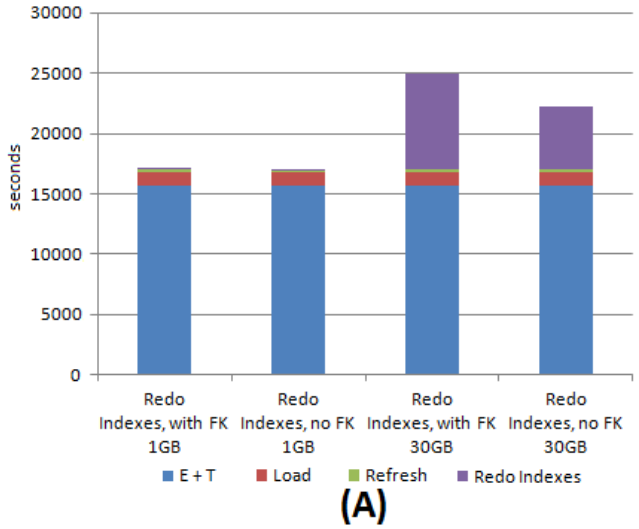

Figure. 9(A). (A) shows ETLR times for a database with 30GB and with 1GB.

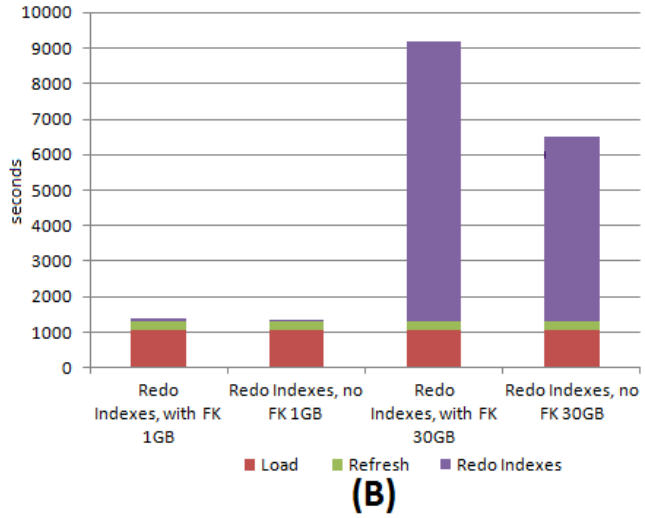

Figure 9(B). (B) shows the time taken to process LR (Load and Refresh) in the same scenarios as (A).

From Figure 9a it is possible to see that the time taken to recreate indexes and foreign keys for the 1GB partitions was quite small comparing with the index recreation time over 30GB. If we do not consider the E+T time (since those are done in parallel and in different nodes), it is possible ot see that there is a drastic decrease in loading and refreshing time. The most relevant time in Figure 9b with 1 GB partitions is then the loading time, which depends only on the size of the log file. If the log (or batch) file is much smaller than the 10M rows in the results, the total loading and refreshing time (the time that the system needs to be offline) drops considerably. For instance, if the batch file has 10k rows, the loading and refreshing time is about 15 seconds in our experimental setup. If it is necessary to load those 10k rows every 5 minutes, then the system will be offline for 15 seconds every 5 minutes, which may be acceptable for many application scenarios.

6. CONCLUSIONS

In this paper we have analyzed the influence of a set of factors on near-realtime capabilities of a traditional data warehouse architecture. We concluded regarding the difficulty of a traditional data warehouse architecture to enforce updates in near-realtime. We have proved that when such as system is loaded online, i.e., with queries running in parallel to the ETLR process, the loading performance is very poor and the query sessions are affected significantly. This is also quite visible when compared with the same process running offline. This means that in most cases a traditional data warehouse architecture should not assume a realtime context keeping the system online while updating its data.

However, we also concluded that it is possible, under certain circumstances, to have near-realtime in traditional data warehouse architectures, if we are able to partition the data and index by partition (to minimize index re-creation times), and if the application scenario accepts short offline periods. This is an important option that renders traditional data warehouse architectures useful in many scenarios that may have some degree of requirements regarding information freshness.

Finally, if there is a need for fresh information at any time and permanent online availability of the data warehouse, then it is preferable to use live data warehouse technologies, which we are currently working with in our research efforts in this field.

7. REFERENCES

[1] O'Neil, P., O'Neil, E., Chen, X., Revilak, S.: Star Schema Benchmark. In: R. Nambiar and M. Poess (eds.): TPCTC 2009. LNCS 5895. Springer-Verlag, Berlin Heidelberg (2009) 237-252.

[2] Wyatt, L., Caufield, B., Pol, D.: Principles for an ETL Benchmark. In: R. Nambiar and M. Poess (eds.): TPCTC 2009. LNCS 5895. Springer-Verlag, Berlin Heidelberg (2009) 183-198.

[3] Simitsis, S., Vassiliadis, P., Dayal, U., Karagiannis, A., Tziovara, V.:Benchmarking ETL Workflow. In: R. Nambiar and M. Poess (eds.): TPCTC 2009. LNCS 5895. Springer-Verlag, Berlin Heidelberg (2009) 199-220.

[4] Jedrzejczak, J., Koszlajda, T., Wrembel, R.: RTDW-bench: Benchmark for Testing Refreshing Performance of Real-Time Data Warehouse. In: S.W. Liddle et al. (eds.): DAXA 2012, Part II, LNCS 7447. Springer-Verlag, Berlin Heidelberg (2012) 199-206.

[5] Oracle, SQL em Oracle, http://aserlorenzo.com/manSQL/Oracle/ddl/indices.htm.

[6] Wikipedia, Bitmap index, http://en.wikipedia.org/wiki/Bitmap_index.

[7] Wikipedia, Foreign Key, http://en.wikipedia.org/wiki/Foreign_key.

[8] Oracle, Partitioned Tables and Indexes, http://docs.oracle.com/cd/B10501_01/server.920/a96524/c12parti.htm.

[9] Waas, F., Wrembel, R., Freudenreich, T., Thiele, M., Koncilia, C., Furtado, P.: On-Demand ETL Architecture for Right-Time BI. In: Proceedings of the 6th International Workshop on Business Intelligence for the Real-Time Enterprise (BIRTE), Istanbul (2012).

[10] Waas, F., Wrembel, R., Freudenreich, T., Thiele, M., Koncilia, C., Furtado, P.: On-Demand ELT Architecture for Right-Time BI: Extending the Vision. In: International Journal of Data Warehousing and Mining (IJDWM), volume 9 number 2 (2013).

[11] Santos, R., Bernardino, J.: Optimizing Data Warehouse Loading Procedures for Enabling Useful-Time Data Warehousing. In: International Database Engineering & Applications Symposium (IDEAS), pp. 292- 299, 2009.

[12] Wikipedia, Materialized View, http://en.wikipedia.org/wiki/Materialized_view.

[13] Ferreira, N.: "Realtime Warehouses: Architecture and Evaluation", MSc Thesis, U. Coimbra, June 2013.