Handling Big Dimensions in Distributed Data Warehouses using the DWS Technique

Marco Costa
Critical Software S.A.
Coimbra, Portugal
mcosta@criticalsoftware.com

Henrique Madeira
DEI-CISUC
University of Coimbra, Portugal
henrique@dei.uc.pt

ABSTRACT
The DWS (Data Warehouse Striping) technique allows the distribution of large data warehouses through a cluster of computers. The data partitioning approach partition the facts tables through all nodes and replicates the dimension tables. The replication of the dimension tables creates a limitation to the applicability of the DWS technique to data warehouses with big dimensions. This paper proposes a strategy to handle large dimensions in a distributed DWS system and evaluates the proposed strategy experimentally. With the proposed strategy the performance speed up and scale up obtained in the DWS technique are not affected by the presence of big dimensions. Furthermore, it extends the scope of the technique to queries that browse big dimensions that can also benefit of the performance increase of the DWS technique.

Categories and Subject Descriptors
H.2.7 [Database Management]: Database Administration – Data Warehousing and Repository.

General Terms: Performance.

Keywords: Data warehousing, distributed query execution.

1. INTRODUCTION
Data warehousing applications typically involve massive amounts of data that push database management technology to the limit. A scalable architecture is crucial, not only to handle very large amount of data but also to assure interactive response time to the OLAP (On-Line Analytical Processing) users. In fact, the decision making process using OLAP is often based on a sequence of interactive queries. That is, the answer of one query immediately sets the need for a second query, and the answering of this second query raises another query, and so on and so forth in an ad hoc manner.

In order to assure acceptable response time to allow the interactive OLAP querying style, even when the data warehouse becomes extremely large in size, data warehouses implementation normally use very expensive platforms, typically based on high-end servers or high-performance clusters. The use of classical parallel processing techniques [10, 11] proposed to relational database systems is also common in big data warehouses [7]. Two types of parallelism can be explored at the query level: inter-query parallelism [9], wherein multiple transactions are executed in parallel in a multiprocessor environment, and intra-query parallelism [13], where several processors cooperate to concurrently execute a single SQL statement. The latter is particular interesting to the complex queries executed in a data warehousing as the parallelism is used to improve performance through parallel implementation of the various operations of the query execution plan. However, the use of parallelism in the complex data warehouse queries is clearly more difficult and less effective than the parallel execution of multiple small transactions that characterize typical database applications in an on-line transaction processing (OLTP) environments.

Another possibility for high volumes of data is to distribute the data across multiple data warehouses in such a way that each individual data warehouse cooperates to provide the user with a single and global view of the data. In spite of the potential advantages of the distributed data warehouses, especially when the organization has a clear distributed nature, these systems are always very complex and difficult to manage [2]. Furthermore, the performance of many distributed queries is normally poor, mainly due to load balance problems and the volume of data exchanged between servers.

The data warehouse striping (DWS) approach [3, 6] is aimed to provide a cost effective alternative for the very expensive servers typically used in large data warehouses by implementing a data warehouse over an arbitrary number of inexpensive computers (typically cheap workstations, server blades, or standard PCs) and, at the same time, integrating this approach in the data warehousing technology available in the market. That is, DWS can be used with the database management systems (DBMS) available today (without changes), including small and cheap ones such as open source DBMS.

The DWS approach is based on the clever combination of two simple ideas: 1) uniform data striping to partition the data warehouse facts over an arbitrary number of computers, in such a way that queries can be executed in a true parallel fashion (a query is actually split into many partial queries), and 2) an approximate query answering strategy (AQA) to deal with the momentary unavailability of one or more computers in the cluster. The experimental evaluation of the DWS technique has shown that this approach assures nearly optimal speed up and scale up [4] and that a momentarily unavailability of one of the computers (which is plausible, as a DWS system may consist of a large

DOLAP'04, November 12–13, 2004, Washington, DC, USA.

number of small computers) does not force the system to stop, as the answers can be approximated with a small error [5] using the data in the remaining computers of the DWS system.

Recently, the company Critical Software, SA (www.criticalsofware.com) developed a middle-layer implementation of the DWS technique targeted for several commercial DBMS systems and OLAP tools, allowing the transparent use of the DWS technique with the data warehousing technology available in the market (i.e., no changes are required for both the DBMS and OLAP tools) [8].

However, the DWS technique has an important limitation: it is specifically thought to typical data warehouses organized in an ideal star schema consisting of a large fact table surrounded by a set of small dimension tables, as proposed by Kimball [12]. In a DWS system the fact rows are uniformly distributed by all the available machines while the dimensions (supposedly small in size) are replicated in all the computers in the DWS system. This means that DWS is not effective (or cannot be used at all) in data warehouses with big dimensions, which is an important limitation as there are a significant number of businesses that have big dimensions as part of their business model.

This paper proposes a new approach called selective loading to deal with data warehouses with big dimensions in DWS systems and evaluates the proposal using the TPC-H performance benchmark [14], whose data schema is quite far from a typical star schema and includes big dimensions.

The paper is organized as follows. Next section summarizes the key aspects of the DWS technique. Section 3 presents the selective loading proposal for handling big dimensions. Section 4 presents the experimental evaluation using the TPC-H schema and finally section 5 concludes the paper.

2. THE DWS TECHNIQUE

The DWS technique relies on specific features of the star schema and the typical data warehouse queries to optimize the way data is partitioned among the computers in the DWS system and the way queries are distributed and executed.

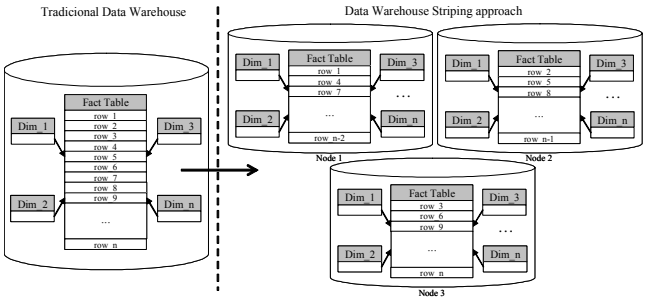

Figure 1 - Data partitioning in the DWS technique

The infrastructure required to implement a DWS is just a set of inexpensive computers (called system nodes, typically PCs or server blades) connected by fast Ethernet (share nothing configuration) and having a DBMS installed in each node. All the nodes have the same star schema, which is the star schema of the equivalent data warehouse in a centralized version. The dimension tables are replicated in each node (i.e., each dimension has exactly the same rows in all the nodes) and the fact data is distributed over all the fact tables of each node using a strict row-by-row round-robin partitioning approach (see Figure 1), which assures an uniform partitioning. The uniform partitioning is necessary to assure optimal load balance and to facilitate the computation of confidence intervals in the cases (rare, in principle) when one or more nodes in the system are not available [1].

As is easy to see, the data partitioning is tightly related to the star schema and the speed up and scale up is very dependent on how close the data warehouse model is to the ideal star schema [12]. The replication of dimension tables does not represent a serious overhead in many cases because usually the dimensions only represent a small percentage of the space occupied by the data warehouse. The row-by-row partitioning of the fact table assures that the fact table in each node has an equivalent number of rows.

The queries are generated by an OLAP tool that assumes the data warehouse is centralized (i.e., all the details of the DWS are hidden by a middle layer [8]) and are executed in a distributed way by all the nodes that constitute the DWS system. A given query is first analyzed to check if it is necessary to transform it and after then it is split into N sub-queries, being N the number of nodes in the DWS system [4, 8]. As the data schema is exactly the same in all nodes (and equal to the equivalent centralized version of the data warehouse), in most of the cases the sub-queries are similar to the same as the original query generated by the OLAP tool.

The nearly optimal speed and scale up achieved by DWS is due to the fact that typical queries lead to sub-queries that are executed in a completely independent way in each node and each query return a partial result that is easy to merge to compute the final result. That is, each node processes approximately the same amount of data as all the others, and mainly within its local data, reducing the exchange of data between nodes to a minimum and almost ignorable. Of course, some queries need more than one step (when some intermediate result is needed to complete the query execution) and in those cases the speed up is not optimal, but these queries represent a small minority of all possible queries [12]. The experimental evaluation using APB-1 benchmark confirmed a practically linear speed up and scale up [4]. In a more detailed way, the key aspects that contribute to the nearly optimal speedup and scale up are:

- *Speed up* (when adding more nodes the response time decrease proportionally):
  1. *Optimal load balance among the nodes:* all the sub-queries take about the same time (assuming all the nodes are well tuned) because the fact rows are distributed by the nodes using a uniform row-by-row round-robin partitioning. The number of fact rows stored in each node about the same (1/N ±1 fact rows in each node, with N the number of nodes), and the low level round-robin partitioning assures the data in each node is not biased in any sense (as would be in the case of using attribute range partitioning) and for each query each node processes the same amount of data as the all others. Furthermore, each query is executed by all nodes, taking advantage of the entire power computation available in the cluster, using a new degree of parallelism not available in other systems.

2. *Independence of sub-queries:* most of the queries over a star schema can be transformed into N independent partial sub-queries (i.e., no need of communication among nodes during the query execution, only the exchange of the partial results) because of the mixture of disjoint partitioning in the fact table with the replication of the dimensions in all nodes. As mentioned before, some queries need to access data stored in the other computers in the DWS system and in these cases the query is executed in more than one step. However, these queries are rare and, if needed, it is possible to estimate the partial result that need the access to data stored in other computers by an estimation calculated from the local data, avoiding the need of truly distributed query [6] (note that accurate approximate result are useful for decision support in many cases).
3. *Negligible overhead to merge partial results:* typical OLAP queries return a small result calculated from the aggregation of a normally large number of fact rows. This is the ideal situation, as in this case the computation of the final result from the partial results is very fast and has a negligible impact in the overall query execution time.

– *Scale up* (when adding more data it is possible to maintain the response time by adding more nodes):

  A very large number of nodes can be used which means DWS represents an effective way to handle very large data warehouses. The only limiting factor for the number of nodes that can be used in a DWS system is the overhead due to the computation of the final query result from the partial results. However, for typical OLAP queries this overhead is small, which means it does not represent a problem even when a large number of nodes is used.

It is worth saying that the DWS technique can be easily adapted to the typical data warehouse life-cycle [12]. The only requirement is to change the final bulk load from the staging area to the data warehouse (either for the initial load or for the subsequent incremental loads) in order to handle replicated dimensions and partition the facts in the required way. A set of administrative tools to facilitate the administration of DWS systems has already been built [8].

The administration of a DWS system is more time consuming then the administration of a centralized server, as the administration of a high number of small machines could be more time consuming than the administration of a big server. However, this is also means that each node can use all the panoply of traditional techniques to reduce query response time, such as B-tree and bit-map indexing, materialized views, table partitioning in each none (note that each node may have more than one disk), in addition to the physical optimization of the storage parameters of the tables in each node and specific tuning techniques available in most DBMS. Furthermore, all nodes have the same star schema model, and will execute the same queries, so in fact, the optimization techniques applied to one node can, and should, be applied to all.

As mention before, the big problem of DWS technique is that it is specifically meant for typical star scheme and the performance gains are greatly affected by big dimensions. In classical business with big dimensions [12], as the client dimension in business such as air transportation, telecommunications, or big e-commerce sites, the DWS technique would be completely inappropriate because of the replication as the big dimensions. Similar situation happens with data warehouses that do not follow a pure star approach, which leads to large tables that play the role of dimensions (the TPC-H schema is an example of this).

In this paper we propose the selective loading of the big dimensions to extend the use of the DWS technique to cases where the business model include big dimensions or non-typical star scheme and evaluate experimentally the viability of this approach.

3. THE SELECTIVE LOADING TECHNIQUE

As explained before, the basic idea behind the DWS approach is to use low cost computers to handle large and very large data warehouses with a reduced infrastructure cost. Each node in a DWS system must process a small portion of the data warehouse. However, if the business model includes big dimensions (i.e., dimensions that may grow to several hundreds or even thousands of MB), the replication of these tables in all DWS nodes will cause a very high storage overhead, but more important than that, will require each node to process a huge amount of data, which turn the DWS technique impracticable or, at the best, significantly reduces the speed up gain obtained by having a large number of computers in the DWS cluster.

A typical data warehouse query aggregates rows of the fact table selected according to some restriction over one or more dimension attributes. In fact, what these queries do is join the fact table with one or more dimension tables in order to filter the fact table rows that should be included in the aggregation. All rows of the Dimension table that have no correspondent row in the fact table will be discarded in the join, and so, each query only needs the dimension rows that can be joined with the fact table rows. In the original DWS approach, the dimensions are fully replicated in all nodes independently of their size, and for each query that uses the big dimension it is necessary to process all its rows in order to select the dimension rows that should be joined with the fact rows.

The selective load technique proposed in this paper explores the fact that the subset of the fact table rows stored in each node are only related to a small part of the rows of the big dimension and not related to all of them. This is a consequence of the partitioning of the facts by a large number of DWS nodes and also a consequence of the nature of the big dimensions (as discussed next). Thus, the idea is to store in each node only the dimension rows that are related to fact rows stored in that node, an not to replicate the entire dimension. That is, the number of rows of the big dimension stored in each node does not depend uniquely on the large dimension itself, but also depends of the rows in the facts tables related to the dimension.

Figure 2 presents a dimension table *Customers* and a fact table *Miles*. The dimension table contains information about airline customers and the fact table contains information about the miles traveled. Let's assume that the data in the fact tables is the data of one DWS node. The customers A and C have at least one fact in the fact table. Without the Selective Load technique the Dimension table would be completely loaded to the DWS Node

including the customers that have no rows in the fact table for the current node. With the Selective Load technique only customer A and C will be loaded to the current DWS node.

Customers

ID	Customer
...	...
1000	A
1001	B
1002	C
...	...

Miles

CustomerID	DayID	Miles
...	...	...
1000	400	3000
1000	401	3500
1002	600	2100
...	...	...

Figure 2 - The selective load technique

At the first sight it may appear that the selective loading technique proposed for the big dimensions does not reduce much the numbers of rows of the big dimension stored in each DWS node. However, a more careful analysis shows that is exactly the opposite, and this proposal leads in general to a big reduction of the volume of rows stored in each node. In fact, a dimension is considered big not only because it has a very high number of rows (absolute size) but also because the number of rows in the dimension represents a significant percentage of the number of rows in the fact table. This means that each row in the big dimension is related to a very small number of facts, which means that a star schema with a big dimension is always very sparse (if it was dense it would be impossible to cope with such data warehouse, no mater the technology used, as it would grow up to pentabytes or more).

Some examples may show the sparsity effect in a more clear way. For example, most of the rows in the client dimension of an air company (i.e., each passenger) are only related to a few number of fact rows (the flights of each passenger). In order to realize how sparse a star like this can be we have to consider the other dimensions of the model (such as time, flights, promotions, etc). In fact, most of the people don't fly very often, but, above all, do not fly everyday, in all the flights of the company, with all the promotions programs, etc, etc. The result is that each row in a big dimension is related with just very few rows in the fact tables (that potentially has zillions of rows). The same happen with other business models that normally have big dimensions such as large e-commerce business with millions of clients, telecommunication companies, big hospitals, etc.

With the selective loading approach we solve the problem of replicating the big dimensions in DWS systems but, unfortunately, we create a new problem: How to answer queries to big dimension table only, the so called dimension browsing queries? With the selective load, there is no complete vision of the big dimension table as each node has only the rows necessary to perform joins with the fact tables, and nothing guarantees that the union of all the dimension rows in all nodes provides a complete vision of the big dimension (not to mention that a union like that would be time consuming). For instance, dimension rows with no facts will not be loaded at all. The solution to this second problem can be found on another foundation of the DWS technique: the data partitioning algorithm.

The selective load technique creates a partitioned version of the big dimension table, using the same row-by-row algorithm used for the distribution of the fact tables. That is, in addition to the partial copy of the big dimension stored in each node, the big dimension is also partitioned among all the nodes for browsing. With this extra redundancy it will be possible to query the big dimension, (i.e., dimension browsing), but even more interesting, it will be possible to do it with the speed up and scale up achieved in the DWS technique for typical data warehouse queries. The distribution of the big dimension does not represent a significant storage overhead to each node because each node will receive a small portion of the data.

The processing required to load the selective version and the partitioned versions of the big dimensions is not negligible, although, these partitioning algorithm is already implemented and optimized in the DWS technique (for the facts), and the preparation of the selective version of the big dimensions for the incremental loads (see [12] for details on the data warehouse life cycle) is performed in the data staging area while the data warehouse is up and running.

4. EXPERIMENTAL RESULTS

In order to evaluate experimentally the gains in storage and performance obtained with the proposed selective load we have decided to implement the TPC-H workload in a DWS system enhanced with the selective load approach and using several DWS configurations concerning number of nodes.

The Figure 3 presents the TPC-H data model.

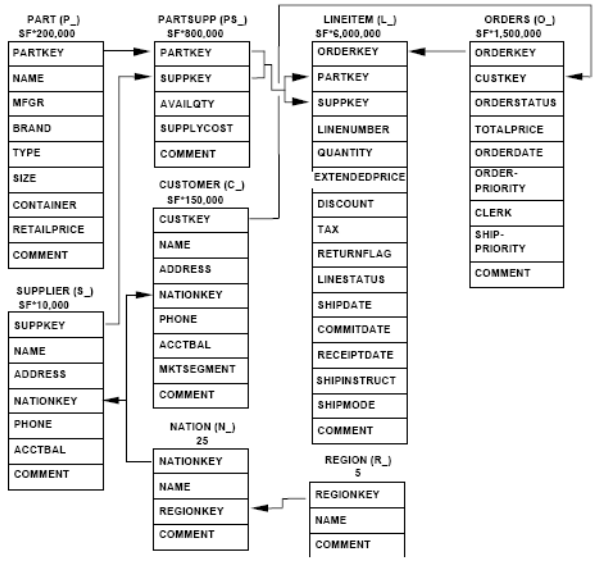

Figure 3 – TPC-H data model [14]

The TPC-H is a good workload for this study as this benchmark is the standard performance benchmark for decision support systems and the TPC-H schema includes big dimensions (although it is not a pure star schema). The TPC-H queries chosen for the experiments are the most demanding ones (because they address particularly the big dimension) and can be found in Annex A. The workload has been generating with 10 GB (i.e., a small size) in order to allow experiments with a wide range of configuration, especially configurations with a single node to be used as a reference (representing a centralized data warehouse).

The tests were performed with the following configurations:

a) Single Node – no DWS technique at all to be used as a reference

b) DWS_5 – DWS with 5 nodes and full replication of dimension tables

c) DWS_SL_5 – DWS with 5 nodes and Selective Load

d) DWS_10 – DWS with 10 nodes and full replication of dimension tables

e) DWS_SL_10 – DWS with 10 nodes and Selective Load

f) DWS_20 – DWS with 20 nodes and full replication of dimension tables

g) DWS_SL_20 – DWS with 20 nodes and Selective Load

Figure 4 shows the storage values of each configuration for the more relevant tables: LinetItem (the fact table), Orders (the big dimension) and Orders_dist (the partitioned version of the big dimension).

In the Figure 2 it is possible to observe that the selective load configuration is always better, in terms of space occupied (considering the point of view of a DWS node), compared with other solutions. It should be noted that the Selective Load configuration includes the LineItem data, the selective loaded version and the distributed version of the Orders Large Dimension table.

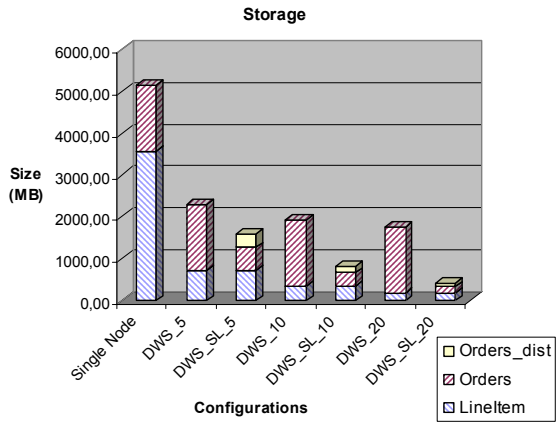

Figure 4 – Storage space per node

Table 1 presents the detailed storage values of the different storage configurations.

	LineItem	*Orders*	*Orders_dist*	*Total*
Single Node	3576,25	1573,56		5149,82
DWS_5	715,25	1573,56		2288,81
DWS_SL_5	715,25	557,97	314,71	1587,93
DWS_10	357,63	1573,56		1931,19
DWS_SL_10	357,63	312,10	157,36	827,09
DWS_20	178,81	1573,56		1752,38
DWS_SL_20	178,81	157,01	78,68	414,51

Table 1 - Storage space in each node (detailed values)

Table 2 presents the experimental results of the execution of the selected TPC-H benchmark queries, in the different configurations. As it can be seen, the selective load technique provides an enormous increase of the query execution performance compared to the corresponding DWS configurations that do not use the selective load. Combined with the DWS technology the execution times can lower from 43 minutes (all the queries) in the single node execution to less than 3 minutes in the 20 node configuration using both DWS and selective load technique. This represent a speed up of 16 times: that is, going from 1 computer to 20 computers the execution time of all the queries tested is reduced 16 times. This is a little less than the optimal speed up obtained for a pure star scheme (the APB-1 benchmark: see in [4]) but quite close anyway. Furthermore, Figure 3 and table 2 shows quite clearly that the DWS technique alone is completely useless (1 computer takes 43:12 while 20 computers take 36:08) in a schema such as the TPC-H, while the DWS together with the selective load introduces a dramatic reduction in the queries execution time. It is worth noting also that due to the synthetic nature of the data used by the TPC-H the sparse factor is not as high as it would be in a real example of big dimensions. Thus the results obtained with the TPC-H are conservative measure of the performance gains obtained with the selective load in a more realistic scenario with big dimensions.

Figure 5 presents the sum up of the execution results.

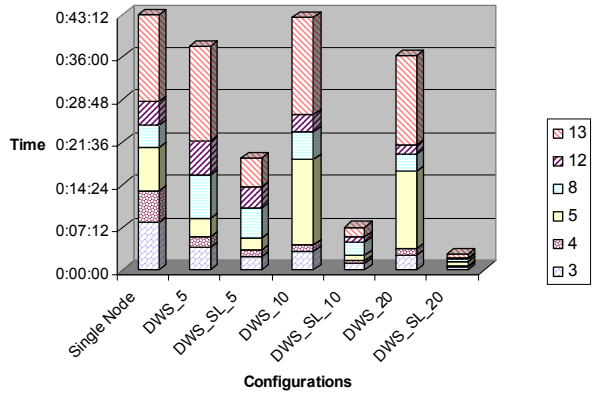

Figure 5 - Queries execution time

Query	*Single Node*	*DWS_5*	*DWS_SL_5*	*DWS_10*	*DWS_SL_10*	*DWS_20*	*DWS_SL_20*
3	0:08:04	0:03:50	0:02:22	0:03:09	0:01:13	0:02:34	0:00:37
4	0:05:16	0:01:52	0:01:03	0:01:13	0:00:23	0:01:01	0:00:07
5	0:07:22	0:03:01	0:02:02	0:14:27	0:00:58	0:13:03	0:00:36
8	0:03:49	0:07:18	0:04:59	0:04:29	0:02:08	0:03:02	0:00:35
12	0:04:02	0:05:47	0:03:34	0:02:58	0:00:59	0:01:26	0:00:05
13	0:14:39	0:15:58	0:04:58	0:16:21	0:01:37	0:15:02	0:00:42
Total	0:43:12	0:37:46	0:18:58	0:42:37	0:07:18	0:36:08	0:02:42

Table 2 – Queries execution time (detail)

5. CONCLUSIONS

This paper presented a new technique called selective load developed to overcome the limitation of the DWS technique in handling big dimensions. The proposed technique enables the DWS technique to handle data warehouses with large dimensions maintaining nearly linear speed up in query execution time. The

experiments performed with the TPC-H schema and queries have revealed that the selective load can improve dramatically the performance of a DWS system when processing queries in data warehouse scheme with big dimensions.

6. REFERENCES

[1] S. Acharya, P.B. Gibbons, V. Poosala, and S. Ramaswamy. "Join synopses for approximate query answering", ACM SIGMOD Int. Conference on Management of Data, pp.275-286, June 1999.

[2] Albrecht, J., Gunzel, H., Lehner, W.: An Architecture for Distributed OLAP. Int. Conference on Parallel and Distributed Processing Techniques and Applications PDPTA (1998).

[3] J. Bernardino and H.e Madeira, "A New Technique to Speedup Queries in Data Warehousing", ABDIS-DASFA, Symposium on Advances in Databases and Information Systems - Enlarged Fourth East-European Conference on Advances in Databases and Information Systems, Prague, Check Republic, September 5-8, 2000.

[4] J. Bernardino and H. Madeira, "Experimental Evaluation of a New Distributed Partitioning Technique for Data Warehouses", IDEAS'01, Int. Symposium on Database Engineering and Applications, Grenoble, France, June 16-20, 2001.

[5] J. Bernardino, P. Furtado, and H. Madeira, "Approximate Query Answering Using Data Warehouse Striping", Dawak'01, 3rd International Conference on Data Warehousing and Knowledge Discovery, Dawak'01, Munich, Germany, September 5-7, 2001.

[6] J. Bernardino, P. Furtado and H. Madeira, "Approximate Query Answering Using Data Warehouse Striping", "Journal of Intelligent Information Systems- Integrating Artificial Intelligence and Database Technologies", Volume 19, Issue 2, Elsevier Science Publication, September 2002.

[7] S. Chauduri and U. Dayal, "An overview of data warehousing and OLAP technology", SIGMOD Record, 26(1):65-74, March 1997.

[8] M. Costa, J.. Vieira, J. Bernardino, P. Furtado, and H. Madeira, "A middle layer for distributed data warehouses using the DWS-AQA technique", 8th Conference on Software Engineering and Databases, Alicante, Spain, November 12-14, 2003.

[9] DeWitt ,D. J. et al.: The Gamma Database Machine Project. IEEE Trans. Knowledge and Data Engineering, Vol. 2, Nº1 (1990) 44-62.

[10] D. J. DeWitt and Jim Gray, "Parallel Database Systems: The future of high performance database systems", Communications of the ACM, 35(6), June 1992, pp.85-98.

[11] Hongjun Lu, Beng Chin. Ooi, and Kian Lee Tan. Query Processing in Parallel Relational Database Systems. IEEE Computer Society, May 1994.

[12] R. Kimball, L. Reeves, M. Ross, W. Thornthwalte, "The Data Warehouse Lifecycle Toolkit", , Ed. J. Wiley & Sons, Inc, 1998.

[13] Stonebraker, M., Katz, R., Patterson, D., Oustershout, J.: The Design of XPRS. Proceedings of VLDB'98, Los Angeles, USA (1988)

[14] Transaction Processing Performance Consortium, "TPC Benchmark H, Standard Specification", 2003 version, available at: http://www.tpc.org/

ANNEX A – TPC-H queries used

Query 3:

```
select l_orderkey,
     sum(l_extendedprice * (1 - l_discount)) as revenue,
     o_orderdate,
     o_shippriority
 from customer, orders, lineitem
 where c_mktsegment = 'BUILDING' and c_custkey = o_custkey
and
     l_orderkey = o_orderkey and
     o_orderdate < to_date('1995-03-15', 'YYYY-MM-DD') and
     l_shipdate > to_date('1995-03-15', 'YYYY-MM-DD')
 group by l_orderkey, o_orderdate, o_shippriority
 order by revenue desc, o_orderdate;
```

Query 4:

```
select o_orderpriority, count(*) as order_count
 from orders
 where o_orderdate >= to_date('1993-07-01', 'YYYY-MM-DD')
     and o_orderdate < add_months(to_date('1993-07-01',
'YYYY-MM-DD'), 3)
     and exists
      (select *
        from lineitem
       where l_orderkey = o_orderkey and l_commitdate <
l_receiptdate)
 group by o_orderpriority
 order by o_orderpriority
```

Query 5:

```
select n_name, sum(l_extendedprice * (1 - l_discount)) as
revenue
 from customer, orders, lineitem, supplier, nation, region
 where c_custkey = o_custkey and l_orderkey = o_orderkey and
     l_suppkey = s_suppkey and c_nationkey = s_nationkey and
     s_nationkey = n_nationkey and n_regionkey = r_regionkey
and
     r_name = 'ASIA' and
     o_orderdate >= to_date('1994-01-01', 'YYYY-MM-DD') and
     o_orderdate < add_months(to_date('1994-01-01', 'YYYY-
MM-DD'), 12)
 group by n_name
 order by revenue desc
```

Query 8:

```
select o_year,
     sum(case
          when nation = 'BRAZIL' then
           volume
```

```
          else
           0
        end) / sum(volume) as mkt_share
  from (select to_number(to_char(o_orderdate, 'yyyy')) as o_year,
           l_extendedprice * (1 - l_discount) as volume,
           n2.n_name as nation
        from part,
           supplier,
           lineitem,
           orders,
           customer,
           nation n1,
           nation n2,
           region
       where p_partkey = l_partkey and s_suppkey = l_suppkey
and
           l_orderkey = o_orderkey and o_custkey = c_custkey
and
           c_nationkey = n1.n_nationkey and n1.n_regionkey =
r_regionkey and
           r_name = 'AMERICA' and s_nationkey =
n2.n_nationkey and
           o_orderdate between to_date('1995-01-01', 'YYYY-
MM-DD') and
           to_date('1996-12-31', 'YYYY-MM-DD') and
           p_type = 'ECONOMY ANODIZED STEEL') all_nations
 group by o_year
 order by o_year
```

Query 12:

```
select l_shipmode,
      sum(case
           when o_orderpriority = '1-URGENT' or o_orderpriority =
'2-HIGH' then
            1
           else
            0
          end) as high_line_count,
      sum(case
           when o_orderpriority <> '1-URGENT' and
              o_orderpriority <> '2-HIGH' then
            1
           else
            0
          end) as low_line_count
  from orders, lineitem
 where o_orderkey = l_orderkey and l_shipmode in ('MAIL',
'SHIP') and
      l_commitdate < l_receiptdate and l_shipdate < l_commitdate
and
      l_receiptdate >= to_date('1994-01-01', 'YYYY-MM-DD')
and
      l_receiptdate < add_months(to_date('1994-01-01', 'YYYY-
MM-DD'), 12)
 group by l_shipmode
 order by l_shipmode
```

Query 13:

```
select c_count, count(*) as custdist
  from (select c_custkey, count(o_orderkey) as c_count
        from customer, orders
        where c_custkey = o_custkey(+) and
            o_comment(+) not like '%special%requests%'
        group by c_custkey) c_orders
 group by c_count
 order by custdist desc, c_count
```