# DATA WAREHOUSE STRIPING: IMPROVED QUERY RESPONSE TIME

Jorge Bernardino
*Instituto Superior de Engenharia de Coimbra, Dept. Informática e de Sistemas*
*Quinta da Nora, Apartado 10057, 3030-601 Coimbra, Portugal*
*Email: jorge@isec.pt*

Henrique Madeira
*Universidade de Coimbra, Departmento de Engenharia Informática*
*Pólo II, Pinhal de Marrocos, 3030-290 Coimbra, Portugal*
*Email: henrique@dei.uc.pt*

Keywords: Data Warehousing, Online Analytical Processing, Query Performance

Abstract: The increasing use of decision support systems led to an explosion in the amount of business information that must be managed by the data warehouses. Therefore, data warehouses must have efficient Online Analytical Processing (OLAP) that provides tools to satisfy the information needs of business managers, helping them to make faster and more effective decisions. Improving query response time in such an environment is very difficult and can only be achieved by a combination of different approaches, in particular the use of materialized views, advanced indexes and parallel query processing. However, achieving quick response times with complex OLAP queries is still an open issue. This paper is an extension of our previous work [Bernardino00], where we proposed a novel approach to this problem. In this paper, we analyse the scalability performance of data warehouse striping system (DWS) system using different environments. DWS is experimentally evaluated with Oracle 8 as back-end DBMS, for the most typical OLAP operations using different types of queries and it is shown that an optimal speedup and scale-up can be obtained. A new technique to process subqueries is also proposed and experimentally evaluated.

## 1. INTRODUCTION

Data warehouses integrate massive amount of data from multiple sources, which is primarily used for decision support purposes. This information is historical in nature and can include detailed transaction information from operational databases, legacy systems, or even external data sources. These large data volumes bring significant challenges to database engines, requiring high level of query performance and scalability. Data warehouses must have efficient Online Analytical Processing (OLAP) tools to process complex analytical queries satisfying the information needs of business managers and helping them to make faster and more effective decisions.

Typical warehouse queries are very complex and ad hoc in nature and generally access huge volumes of data and perform many joins and aggregations. Therefore, to handle this demanding in data and processing a highly scalable architecture is vital in a warehouse environment.

In this paper, we build upon our previous work [Bernardino00] where a horizontal data partitioning of the fact table specially designed for relational data warehouses based on a star schema was proposed. This approach, called data warehouse striping (DWS), takes advantage of the specific characteristics of star schemas and typical data warehouse queries profile to guarantee optimal load balance of query execution and assures high scalability. In DWS the fact tables are distributed by an arbitrary number of computers and the queries are executed in parallel by all the computers, guarantying a nearly linear speedup and significantly improve query response time.

This paper presents an experimental evaluation of the speedup and scaleup of the proposed

technique based on an implementation with three, five and ten computers. We also investigate a technique to deal with subqueries. In this type of queries, the independent calculation of partial results required for nearly optimal speedup is obtained through estimation of some steps of the query. However, it will be shown that the error introduced this way in the final result is negligible for most of the cases.

The remainder of the paper is organized as follows. In the next section, we review related work and discuss the problems associated to the manipulation of large amounts of data in data warehouses. Section 3 describes the proposed approach and section 4 discusses experimental results. The final section contains concluding remarks and future work.

## 2. RELATED WORK

The most common strategies to improve query response times in warehousing environments are the precomputation of data in the form of materialized views [Chauduri97, Agrawal00] and the use of special indexes structures [Wu98, Srinivasan00]. Although necessary to tune a DW for speed, these techniques have clear limitations and in the end cannot avoid some very time consuming queries. The precomputation strategy needs to anticipate queries so that it can materialize them in the data warehouse. However, OLAP queries are most frequently of an ad hoc nature, which restricts the precomputation to the imagination of the DW administrator. Users might query on dimensions that are not materialized in views. The indexing structures provide faster access to the data stored in the warehouse, but increases the size of the tables stored in that data warehouse. One can say that the strategies presented provide faster query processing, but they may not be fast enough as perceived by the user. Additionally, they require constant care from the DW administrator.

Parallel processing techniques have been applied to relational database systems [Lu94, DeWitt92] to speedup queries. The basic idea behind parallel databases is to carry out evaluation steps in parallel whenever possible, in order to improve performance. Although some vendors support parallel data warehousing to various degrees, e.g. Oracle8 [Oracle97] and Red Brick Warehouse [RedBrick98], the fact is that parallel query processing in data warehouses has received little attention in research community.

Most of today's OLAP tools require data warehouses with a centralized structure where a single database contains all the data. However, a large centralized data warehouse is very expensive because of the great setup costs and it is not very flexible due to its centralized nature. The fact that data warehouses tend to be extremely large in size [Chauduri97] and grow quite fast in many cases means that a scalable architecture is crucial to the success of these systems. A truly distributed data warehouse can be achieved by distributing 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 distributed data warehouses, these systems are always very complex and have a difficult global management [Albrecht98]. On the other hand, the performance of many distributed queries is normally poor, mainly due to load balance problems.

In this context the data warehousing striping approach is more flexible because is a distributed implementation of a centralized data warehouse. The data warehouse striping is inspired in both the distributed data warehouse architecture and in classical round-robin partitioning techniques. The data is partitioned in such a way that the load of query execution is uniformly distributed by the available computers. Normally, a partial result is computed in each computer in a pure independent way. For the queries that need communication among computers during query execution we propose a solution that maintain these type of queries execution independent of the others.

## 3. QUERY PROCESSING USING DWS

In this section, we review the data warehouse striping approach [Bernardino00]. In particular, is explained how the queries are processed using the DWS system. First, we give some background on data schemas used in data warehouse environments. Then, we present strategies for rewriting queries that cannot be executed independently in each computer.

### 3.1 Background and DWS

A data warehouse is organized, in logical terms, according to the multidimensional model. This model consists of a set of dimensions and a set of measures (facts), where each dimension is typically arranged in a hierarchy. All dimensions have some attributes, describing a different perspective for the analysis of business. For instance, in the classical example of a chain of stores business, some of the

dimensions are Products, Stores, and Time. In this simple three-dimension example, we have a cube (see Figure 1) where each cell within the multidimensional structure contains measures (typically numerical facts) along each of the dimensions. For example, a single cell may contain the total sales for a given product in a given store in a single day.

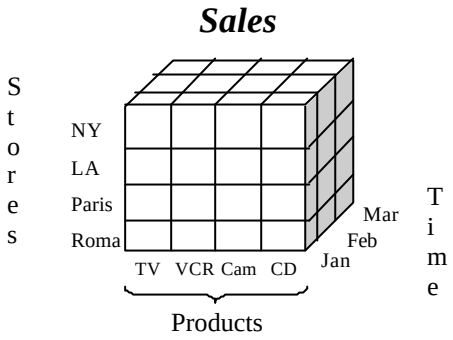



Figure 1: The multidimensional schema.

The conceptual multidimensional data model can be physically realized in two ways, (1) by using trusted relational databases (star schema) [Chauduri97] or (2) by making use of specialized multidimensional databases.

In this paper, we assume that a multidimensional database is a relational data warehouse in which the information is organized as a star schema [Kimball96]. It offers flexibility, but often at the cost of performance because more joins for each query are required. A star schema models a consistent set of facts (aggregated) in a central fact table and the descriptive attributes about the facts are stored in multiple dimension tables. The equivalent star schema of the example of a chain of stores presented before is shown in Figure 2. The fact table is Sales and the dimensions are Products, Stores and Time.

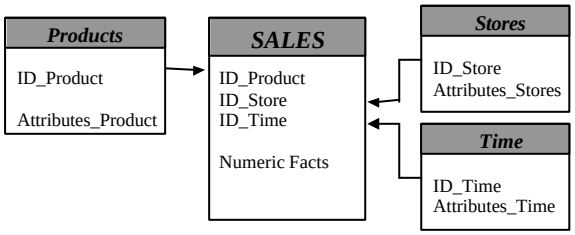



Figure 2: Star Schema.

The size of fact table is much bigger than the size of dimension tables. The size of dimension tables could be hundreds or thousands of rows while the fact table could be millions of rows [Kimball96]. Queries tend to use aggregations and joins on two or more tables, have often a large computation time, and are mainly ad hoc in nature.

Taking in account these characteristics of data warehouses our approach replicates the dimensions of star schema and distributes evenly the rows of fact table by the computers that form the DWS system.

All the computers that comprise the DWS system have the same star schema structure, equivalent to the centralized version. The dimension tables are replicated in each computer and the fact rows are distributed by the fact tables of each computer using a row-by-row round robin partitioning approach as illustrated in Figure 3.

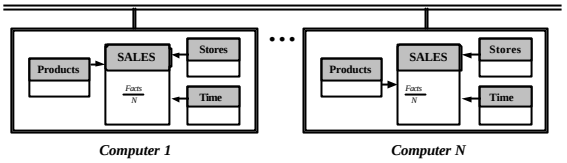



Figure 3: Data Partitioning in DWS.

Using DWS, the number of fact rows stored in each computer is about the same, meaning that each computer has a fraction of the fact rows (Facts/*N*), with *N* being the number of computers. In the next section, we explain how the queries are executed in the DWS system.

## 3.2 Processing Queries with DWS

In DWS, typical OLAP queries are transformed into *N* partial queries that are executed in parallel in each of *N* computers that represent the DWS system (see Figure 4), avoiding the need of communication between computers during query execution. The parallel execution of a query by the available computers maintains the best load balance because the number of fact rows stored in each computer is about the same and the row distribution is random.

The Query Distribution and Processing (QDP) is a DWS layer responsible to receive the original query from the client application, rewriting it if necessary, and distribute this "modified" query by all computers. After the execution of these partial queries, the QDP layer merges these partial results and sends the final result to the user.

Most of the queries over a star schema can be transformed into *N* independent partial queries just because of the nature of the queries (and because the fact rows are partitioned in a disjoint way: i.e., the fact rows stored in a computer are not replicated in the other computers). For some queries, the computation of the partial result in one computer needs to access data stored in the other computers of

the DWS system (i.e., needs a truly distributed query). However, in most of these cases it is possible to accurately estimate the partial result that need to access the data stored in other computers by an estimation calculated from the local data, avoiding the need of a truly distributed query. Even though this is only possible because we are working with aggregation functions.

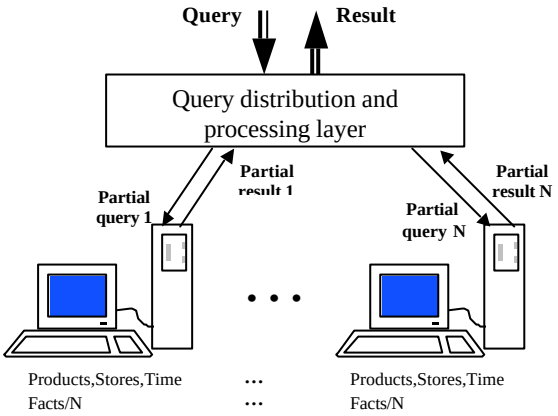



Figure 4: Query Processing in DWS.

In reality, processing some queries in a DWS system could need to access intermediate results computed from all the computers. Thus, in a DWS system concerning the need of communication among computers during query execution, typical queries can be divided in two groups:

- *Independent queries*: queries that can be independently executed in each computer, with no need of communication among computers during the execution of the query. In this case the global result can be computed very quickly from the partial results.
- *Dependent queries*: queries that need communication among computers to compute the final result. In these queries the computation of the partial results in each computer need to access data stored in others computers. In this case, the execution of queries is dependent of partial results calculated over the data in all the computers of the DWS system. We designate the number of times that is necessary to access the data in all the computers as number of executions.

In a DWS system the ideal speedup is equal to the number of computers (*N*) used in a DWS system. Independent queries are distributed by the available computers in a DWS system in such a way that each computer needs to execute only one partial query (number of executions=1), which corresponds to the nearly optimal linear speedup.

On the other hand, dependent queries raise several problems concerning the speedup of DWS, as it is not possible to attain the optimal speedup, because the nodes must compute intermediate results that are required in additional steps to compute the partial and final results.

An example of dependent query is a query containing subqueries, where we use the results of one query as part of another query. In the DWS system, this means that we must access the data stored in the others computers to compute the results of subqueries. For example to calculate the number of units sold for all products whose sales price is greater than the average sales price. Using the star schema of Figure 5 (that will be explained in next section), this query corresponds to the following SQL statement:

```
Q0:  select sum(unitssold)
     from actvars
     where dollarsales >
          (select avg(dollarsales)
          from actvars)
```

During the execution the inner query that computes the average dollarsales is executed first. The result of this subquery is used in the main query. The problem with this type of query when DWS is used is that the original query must be partitioned and reformulated, resulting into new partial queries. This means that the distribution of subqueries requires two phases:

*Phase one*: The innermost subqueries are send to all the computers and the results are computed. The intermediate results are made available for all computers.

*Phase two*: The main query is rewritten taking into account the intermediate results and it is sent to all computers. If the deep level of the query is greater than two levels these subqueries must be executed sequentially from the innermost query until the main query.

In our example, the original query Q0 gives origin to the following two queries that must be executed in sequence (but each one is executed in parallel in all the available machines):

```
Q0.1:     select sum(dollarsales),
               count(dollarsales)
          from actvars

Q0.2:     select sum(unitssold)
          from actvars
          where dollarsales>
               average_dollarsales
```

The first query, Q0.1, computes the product average sales price and the result can be stored in the auxiliary variable *average_dollarsales*. The second query, Q0.2, computes the units sold for revenues greater than the value computed by Q0.1 (*average_dollarsales*). DWS does not achieve optimal speedup with this type of query because it requires the value of the subquery to be computed first.

A solution to the problem of dependent queries could be the use of an approach similar to materialized views where we store aggregation values like sum, count or average. But this solution has the typical problems of materialized views: we do not know a-priory all the values needed by the user because we are working essentially with ad hoc queries.

Instead, we propose another approach that uses the estimation of aggregation values. When a business manager is "digging the data", s/he usually wants to find data trends, extracting important information to future decisions. In this way, the most important action is to explore the data quickly, even if the results are only approximate. This form of working with DW is well suited to DWS technique because this way we can transform dependent queries into approximate independent queries, producing a final result that is an estimation of the exact value but that is obtained much more quickly.

In short, in dependent queries we propose the elimination of the first step by estimation of the aggregation value locally. This means that we transform all dependent queries in independent ones. Using this approach with our example, we use the local average of each computer as an estimate of the global average. In the next section, we analyze more deeply the proposed approach and the errors introduced in the final result with queries that aggregate groups with different number of elements. As we will see later on, the error introduced in the final result is negligible in most cases.

# 4. EXPERIMENTAL STUDY

The main goals of this study are to show typical speedup and scaleup characteristics of the DWS approach, to illustrate the independent execution of subqueries, and to demonstrate the small error obtained in this case.

All the experiments are conducted in a Windows NT environment, where all the computers have the same hardware characteristics, using Intel Pentium Celeron 466 MHz CPU, with 6.4 GB IDE hard disk and 64 MB of main memory. The computers use Oracle 8, release 8.0.5 as back-end DBMS and are linked together in an Ethernet network at 100 Mbps.

To evaluate the performance of the proposed technique we used the data set of the Analytical Processing Benchmark-1, Release II [APB98], that simulates a realistic OLAP business situation.

## 4.1 Experimental testbed

Figure 5 shows our star schema based on the APB-1 benchmark. The schema provides a typical sales analysis environment with one fact table, ACTVARS (24,786,000 rows) and four dimension tables PRODLEVEL (9,000 rows), CUSTLEVEL (900 rows), TIMELEVEL (24 rows) and CHANLEVEL (9 rows). As usual for star schemas, dimension tables are denormalized to reduce join overhead. The fact table ACTVARS holds the measuring attributes *UnitsSold*, *DollarSales* and *DollarCost* for calculating aggregations.

In order to have a baseline reference for the experiments we apply our technique to one computer, simulating a centralized data warehouse and denote it as CDW (Centralized Data Warehouse). DWS-3 corresponds to the DWS technique applied to three computers. DWS-5 and DWS-10 correspond to data warehouse striping applied to five and ten computers, respectively.

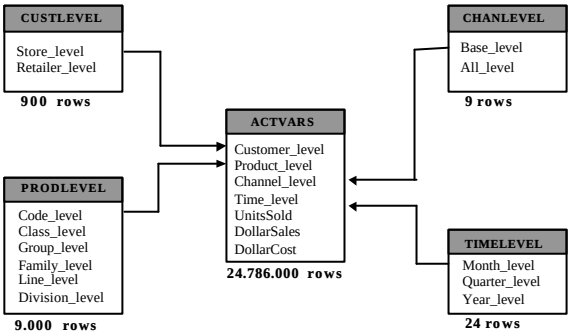



Figure 5: Simplified star schema of APB-1 benchmark.

Typical OLAP queries use aggregation functions (SUM, COUNT, MAX, MIN, AVG, STDDEV and VARIANCE) as operators that aggregate a large number of rows in the fact table and return few rows as result. Accordingly, our experiences are simple queries that use these aggregation functions and return only one row. Another type of usual operation of OLAP queries is the join of the fact table with one or more dimensions, which is one of operations computationally most expensive. The objectives of these experiences are the evaluation of the speedup

and scaleup of DWS using queries that use aggregation functions and joins.

The vast majority of queries, using the DWS approach, can be converted into queries that compute partial results, independently obtained in the different computers, and the global result can be computed very fast from these partial results. However, for some queries, the independent calculation of partial results is obtained through estimation of some steps. This is another important objective of these experiences, quantifying the error introduced this way in the final result.

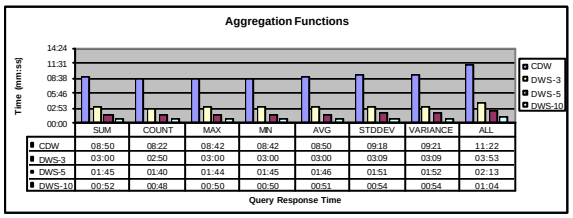
## 4.2 Performance study

### 4.2.1 Aggregation performance

In order to evaluate the speedup of DWS using aggregation functions, the experiments are made with a simple type of queries, expressed in SQL:

```
select aggregate_function(unitssold)
from actvars
```

Different aggregation functions are used (SUM, COUNT, MAX, MIN, AVG, STDDEV, VARIANCE and ALL functions simultaneously) in order to evaluate the impact of the aggregation function in the query response time. The results query response time using the different aggregate functions are illustrated in Figure 6, for the centralized data warehouse (CDW), using three computers DWS-3, five computers DWS-5, and ten computers DWS-10.

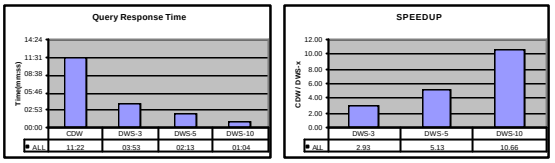



| | SUM | COUNT | MAX | MIN | AVG | STDDEV | VARIANCE | ALL |
|---|---|---|---|---|---|---|---|---|
| CDW | 08:50 | 08:22 | 08:42 | 08:42 | 08:50 | 09:18 | 09:21 | 11:22 |
| DWS-3 | 03:00 | 02:50 | 03:00 | 03:00 | 03:00 | 03:09 | 03:09 | 03:53 |
| DWS-5 | 01:45 | 01:40 | 01:44 | 01:45 | 01:46 | 01:51 | 01:52 | 02:13 |
| DWS-10 | 00:52 | 00:48 | 00:50 | 00:50 | 00:51 | 00:54 | 00:54 | 01:04 |

Figure 6: Query response time using aggregates.

For each DWS environment the query response time does not depend significantly on which aggregation function is used because the same amount of rows have to be fetched and I/O dominates the response time. The execution times of STDDEV or VARIANCE functions are slightly larger, due to the increased complexity of the operations, and that of ALL aggregation functions simultaneously is about 30% larger in CDW.

The speedup is measured by adding more computers to the DWS system, but keeping the database size constant. Dividing the query response time of CDW by the corresponding time of DWS-X (where X denotes the number of computers) achieves a theoretical speedup of X. For instance, we could determine the speedup of the results from Figure 6 by taking the query response time for the centralized version (CDW) and dividing it by the query response time obtained with three, five and ten computers (DWS-3, DWS-5, DWS-10). The Figure 7b shows the speedup correspondent to query response time of Figure 7a when we are using ALL aggregation functions simultaneously.




(a) (b)

Figure 7: Speedup using all aggregation functions

The most surprising result is that we obtained a speedup of 5.13 and 10.66 for DWS-5 and DWS-10 respectively, which is greater than the theoretical speedup, suggesting a super linear speedup of DWS when we distribute the data by more computers. As we see, this phenomenon is not present when we use DWS-3 because the workload in individual computers was approaching the memory limits. This result also emphasizes the scaleup advantages of DWS, as distributing the fact table over a larger number of smaller machines allows DWS system to cope with very large DW data.

### 4.2.2 Join performance

We have also tested DWS with queries that join the fact table with all the dimensions of the star schema of Figure 5. Four queries have been used in the experiments that join the fact table (ACTVARS) with the dimensions PRODLEVEL, TIMELEVEL, CUSTLEVEL and CHANLEVEL, which corresponds to 1-Join, 2-Join, 3-Join and 4-Join, respectively. For example, the 4-Join query corresponds to the following SQL statement:

```
select sum(unitssold)
from actvars, prodlevel, timelevel,
     custlevel, chanlevel
where < join conditions >
```

The other queries used in the experiments are similar to this one. Figure 8 shows the query response times for these four join queries. These experiments show that queries that take about 11 minutes in the centralized version (CDW) are executed in less than 4 minutes in DWS-3, about 2 minutes using DWS-5 and 1 minute using DWS-10.

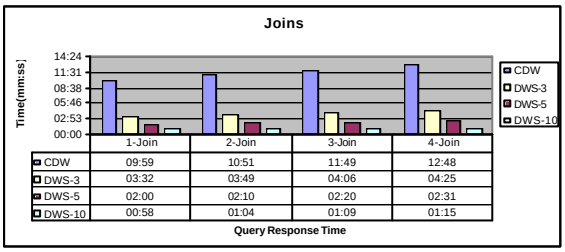
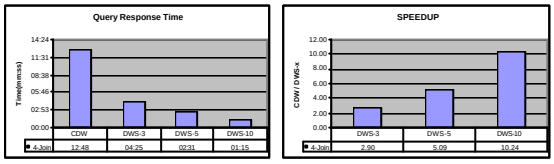



Figure 8: Query execution time with joins.

Figure 9a shows the execution time and Figure 9b the corresponding speedup for the 4-Join query. This query that takes 12:48 minutes in CDW is executed in 4:25 minutes using DWS-3, 2:31 minutes using DWS-5 and only 1:15 minute using DWS-10, correspond to speedups of 2.9, 5.09 and 10.24, respectively.




(a) (b)

Figure 9: Execution time and speedup using the 4-Join query.

These results confirms our hypothesis of optimal speedup of DWS when the computers are not running near the memory limits as is the case of DWS-5 and DWS-10. Thus, for computers with the same characteristics, how much bigger is the data warehouse better speedup we will obtain using DWS. This is due to the fact the data set we are working on is distributed in more manageable data sets that could be more easily computed using computers limited by memory space.

#### 4.2.3 Evaluation of dependent queries (DQ) with estimation procedures

In these experiments we quantify the error introduced by the estimation of intermediate results in subqueries, which restrict very much the number of rows that are selected.

Our experimental set-up is composed of the CDW, which calculates the exact result and both DWS-3 and DWS-5 systems. In the experiments we use the query Q0, defined in the previous section. A more restrictive query than Q0 is also defined, which computes the total number of units sold for a product whose sales are greater than the averages sales of the same product. We denote this as query Q3 and use it in the experiments with different product names. The SQL statement for query Q3 and product name 'GTMO4OPZVEDI ' is:

```
Q3:  select sum(unitssold)
     from actvars where
     product_level ='GTMO4OPZVEDI' and
     dollarsales >
          (select avg (dollarsales)
          from actvars where
          product_level ='GTMO4OPZVEDI')
```

This query selects 1,464 rows out of 24,786,000 from ACTVARS fact table. However, the query Q3 is modified to restrict even more the number of rows selected by the aggregation function, inserting an additional time level restriction. In the query Q4 (not shown) the time level restriction is inserted only in the subquery, while query Q5 also inserts the time level restriction in the main query, as illustrated by the following SQL statement:

```
Q5:  select sum(unitssold)
     from actvars where
     product_level ='GTMO4OPZVEDI' and
     time_level ='199506' and
     dollarsales >
          (select avg (dollarsales)
          from actvars where
          product_level ='GTMO4OPZVEDI'
          and time_level ='199505')
```

Query Q5 is a very selective query that only selects 103 rows from the fact table. Table 1 shows the results for the experiments done with the different queries. The columns in Table 1 show the names of the different products used by the queries, the number of rows that are selected by each query and the corresponding selectivity. The CDW column shows the correct final result of the query and the other columns DWS-3 and DWS-5 show the result using DWS with three or five computers and the relative error contained in these results.

The results show that when the selectivity of the query is very high, as Q0, the result of the query is estimated with very small error (0.06% for DWS-5 and no error for DWS-3). Using queries with very low selectivity, the error was always less than about 5% with DWS-5 and less than 1.2% with DWS-3.

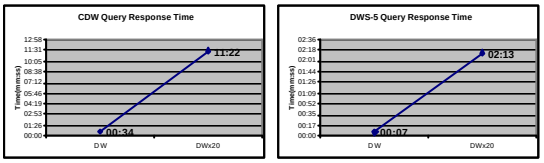
## 4.3 Scalability of DWS

To analyse the scaleup we keep a constant hardware configuration and increase the data warehouse size. In this experiment we increased the data size 20 times and we measured the increase in the query response time. If the configuration can scaleup by a factor of 20 then the query response

| Query | Product | No of rows sel | Selectivity | CDW result | DWS-3 | | DWS-5 | |
|---|---|---|---|---|---|---|---|---|
| | | | | | result | Error | result | error |
| Q0 | All | 8956764 | 36.13% | 90955602 | 90955602 | **0.00%** | 91009415 | **0.06%** |
| Q3 | GTMO4OPZVEDI | 1464 | 0.0059% | 15416 | 15323 | **0.60%** | 15729 | **2.03%** |
| Q3 | JNVFUDO2QKIE | 1397 | 0.0056% | 14223 | 14110 | **0.79%** | 14155 | **0.48%** |
| Q4 | GTMO4OPZVEDI | 1359 | 0.0055% | 14445 | 14489 | **0.30%** | 15178 | **5.07%** |
| Q4 | JNVFUDO2QKIE | 1345 | 0.0054% | 13839 | 13813 | **0.19%** | 14022 | **1.32%** |
| Q5 | GTMO4OPZVEDI | 103 | 0.0004% | 1060 | 1049 | **1.04%** | 1039 | **1.98%** |
| Q5 | JNVFUDO2QKIE | 100 | 0.0004% | 979 | 968 | **1.12%** | 973 | **0.61%** |

Table 1: Error for the different queries when Dependent Queries (DQ) are executed as Independent Queries (IQ).

time will not increase more than 20 times. As comparison, we use again ALL the aggregation functions simultaneously, and the query response time is shown in Figure 10 (a) and (b) using a CDW and DWS-5, respectively.




(a) (b)

Figure 10. Scaleup effect when the DW increases 20 times using CDW (a) and DWS-5 (b)

Calculating the respective scaleup, we obtain a scaleup of 20.06 for the CDW and 19.00 for DWS-5. For the CDW this value of scaleup means that when we increase the size of the DW the query response time is always slightly bigger than this increment. Using five computers (DWS-5) we see an interesting effect because the increment is only 19 times; less than then the theoretical increment of 20 times. This confirms our hypothesis of "the bigger the better" as increasing the size of the DW and distributing it by more computers the gain is even better.

# 5. CONCLUSIONS

The experimental results using aggregation and join queries show an optimal or even super linear speedup and also sow that DWS is scalable. This is due to the fact that, when we distribute the data, we are working with more manageable amounts of data that do not stress memory and computing resources so much. The manipulation of subqueries in an independent way is also possible with an error that can be negligible in most of the cases.

We believe that our modular approach can be incorporated into a commercial relational DBMS with little effort. However, there are still some open questions: one of the identified problems is big dimensions. Typically, dimensions are small in size but there are exceptions to this rule. In that case the space overhead of the strategy becomes more significant. Another future direction is to solve the problem of momentarily unavailability of one or more computer in a DWS.

# REFERENCES


[Agrawal00] S. Agrawal, S. Chaudhuri, V. R. Narasayya, "Automated Selection of Materialized Views and Indexes in SQL Databases", Proceedings of VLDB'2000, Cairo, Egypt, pp 496-505

[Albrecht98] J. Albrecht, H. Gunzel, W. Lehner, "An Architecture for Distributed OLAP", Int. Conf. on Parallel and Distributed Processing Techniques and Applications PDPTA'98, 1998.

[APB98] APB-1 Benchmark, Olap Council, November 1998. Available at www.olpacouncil.org.

[Bernardino00] J. Bernardino, H. Madeira, "A New Technique to Speedup Queries in Data Warehousing", ABDIS-DASFAA, Prague. 2000.

[Chauduri97] S. Chauduri and U. Dayal, "An overview of data warehousing and OLAP technology", SIGMOD Record, 26(1), March 1997, pp.65-74.

[DeWitt92] D. J. DeWitt and Jim Gray, "Parallel Database Systems: The future of high performance database systems", Com. of ACM, 35(6), June 1992, pp.85-98.

[Kimball96] "The Data Warehouse Toolkit", Ralph Kimbal, Ed. J. Wiley & Sons, Inc, 1996

[Lu94] Hongjun Lu, Beng Chin. Ooi, and Kian Lee Tan. Query Processing in Parallel Relational Database Systems. IEEE Computer Society, May 1994

[Oracle97] "Star queries in Oracle 8", White paper, Oracle, 1997

[RedBrick98] Red Brick Systems, Inc. "Star Schema Processing for Complex Queries", White Paper, 1998

[Srinivasan00] J. Srinivasan et al., "Oracle8i Index-Organized Table and Its Application to New Domains", Proc. of VLDB'2000, Egypt, pp.285-296

[Wu98] M. Wu, A. Buchmann, "Encoded Bitmap Indexing for Data Warehouse", ICDE Conf., 1998