Data Warehousing and OLAP: Improving Query Performance Using Distributed Computing

Jorge Bernardino
Instituto Superior de Engenharia de Coimbra (ISEC)
Dept. Eng. Informática e de Sistemas
Coimbra, Portugal
jorge@isec.pt

Henrique Madeira
Universidade de Coimbra
Departamento de Engenharia Informática
Coimbra, Portugal
henrique@dei.uc.pt

Abstract

Data warehouses are used to store large amounts of data. This data is often used for On-Line Analytical Processing (OLAP) where short response times are essential for on-line decision support. One of the most important requirements of a data warehouse server is the query performance. The principal aspect from the user perspective is how quickly the server processes a given query: "the data warehouse must be fast".
The main focus of our research is finding adequate solutions to improve query response time of typical OLAP queries and improve scalability using a distributed computation environment that takes advantage of characteristics specific to the OLAP context. Our proposal provides very good performance and scalability even on huge data warehouses.

1 Introduction

In recent years, there has been an explosive growth in the use of databases for decision support. This was mainly due to the fact that information, one of the most valuable assets of an organization, can assist in decision making and this way significantly improve the value of an organization. This phenomenon is a result of the increased availability of new technologies to support efficient storage and retrieval of large volumes of data, namely data warehousing and OLAP applications. Thus, we could define data warehouse as "*a repository of data that has been extracted and integrated from heterogeneous and autonomous distributed sources*" [Kimball96].
Data warehouses usually contain a vast amount of data that must be analyzed and, OLAP applications provide that analysis, helping in the organizational decision making process. The success of this kind of support depends heavily on database systems and correlated analysis tools. Data warehouses and OLAP applications differ significantly from the traditional database applications. On-line Transactional Processing (OLTP) systems are oriented towards the access of few

tuples for read/write access at any time. Data warehouses and OLAP provide a different context in which huge amounts of data must be processed efficiently and queries are often complex, but still require interactive response times. In data warehouse environments the data is used for decision support and large sets of data are read and analyzed *ad-hoc*. Most of today's OLAP applications work on data warehouses with a centralized structure in which a single database contains huge amounts of data, which is very expensive from the computation perspective.

Data warehouses tend to be extremely large – the data warehouse of General Motors, exceeds 1.5 terabytes in size, and contains a fact table with more than 2 billion rows [Gill00] in which queries can take hours to complete. Data warehouses also tend to grow quite quickly. Therefore, to handle this expansion in data and processing a highly scalable architecture is vital in a warehouse environment. In fact, this scalable architecture has to handle very large amounts of data and at the same time assure interactive response times to OLAP users.

The centralized data warehouse is very expensive because of the large setup costs and it is not very flexible due to its centralized nature. The use of small data warehouses (data marts) was the first attempt to solve the problems of space and performance but data marts are basically stand alone and have data integration problems in a global data warehouse context.

We propose a round-robin data partitioning approach specially designed for relational distributed data warehouse environments. This new technique takes advantage of the specific characteristics of star schemas and typical OLAP query profile to guarantee optimal load balance of query execution and assure high scalability. As will be shown in the paper, a huge data warehouse can be distributed by an arbitrary number of relatively small computers using this approach, guarantying a scalable and cost effective solution.

The remainder of the paper is organized as follows. Next section contains the main problems in the field of data warehouse research, and Section 3 outlines the current knowledge of the problem domain, as well as the state of existing solutions. Section 4 describes the proposed approach and preliminary results. Section 5 discusses the identified problems of the approach. Section 6 points out why our solution is better than existing approaches and Section 7 concludes the paper and suggests future work.

2 Significant problems in the field of Data Warehouse research

In this section we outline a number of research problems that arise in the warehousing approach. We classify the open research issues in the data warehouse field into five groups as identified in [Wu97]:

I. Data warehouse modelling and design

The relational model and the multidimensional model are the main data models used in data warehousing. However, new design methodologies and design tools for data warehouses are needed with the appropriate support for aggregation hierarchies, mapping between the multidimensional and the relational models, and cost models for partitioning and aggregation that can be used from the early design stages.

II. Data warehouse architectures

Research into future integration of legacy data warehouses is required, particularly as many companies are developing data marts of limited scope. Other interesting architectural issues concern the unbundling/streamlining of relational DBMSs, since data warehousing does not require full DBMS capabilities, such as, concurrency control and ACID properties. The data warehouse architecture also must provide services for efficient storage, retrieval and management of large amounts of data.

III. Data warehouse maintenance

Data warehouse maintenance issues include data cleansing, initial loading, data refreshing, data purging and metadata management. Support mechanisms and metadata management for the resolution of conflicts during data cleansing are required, as well as parallel loading utilities. Strategies for efficient data refreshing are also required, as well as clear criteria and cost models for data removal from the data warehouse.

IV. Operational issues

The data partitioning of a large table into small tables avoid scanning a large table improving query performance. Partitioning algorithms developed as fragmentation algorithms for distributed databases need to be re-evaluated for data warehousing. The set of relational operators must be extended to satisfy the user/application requirements. These extensions must include efficient mapping between the multidimensional and the relational worlds. These new operators must be efficiently implemented and the traditional operators, such as joins, must be re-implemented in an optimized manner for the warehouse requirements, particularly to support star joins and to take advantage of special indexing. Other operators that need re-implementation to take advantage of data warehouse characteristics are large table scans, which must be parallelized, and partial results must be usable by multiple queries.

V. Optimization

Research in query optimization that exploits the information available in aggregation hierarchies and join indexes, as well as issues of space/time trade-offs for computed views is needed. Additionally, data warehousing has special indexing requirements that are partially satisfied by bitmap indexes. These require extensions to deal better with sparsity, and to perform selective indexing, domain transformations and sub-domain indexing.

Our research work focuses mainly in the operational issues and optimization topics but is also related to data warehouse architectures and design. In our work we improve query response time by partititioning data among a set of computers using a distributed computing environment. This approach can potentially achieve linear speedup and can significantly improve query response times.

3 Related work

There are several strategies to improve query response time in the data warehouse context: indexing techniques, materialized views, parallelism, and partitioning of data.

Indexing techniques are among the first areas on which a database administrator will focus when good query performance in a read intensive environment is critical. Specialized indexing structures offer the optimizer alternatives access strategies for the time consuming full table scans. One of the most popular index structures is the B-tree and its derivatives [Comer79]. B^+tree indexes are the most common supported structures in RDBMS, but it is a well-known fact that tree structures have inconveniences when the cardinality of the attribute is small. Another class of index structures, the bitmap indexes, try to overcome the problem by using a bit structure to indicate the rows containing specific values of the indexed attribute [O'Neil97]. Although essential for the right tuning of the database engine, the performance of index structures depends on many different parameters such as the number of stored rows, the cardinality of the data space, block size of the system, bandwidth of disks and latency time, only to mention some [Jurgens99].

The use of materialized views (or summary tables) is probably the most effective way to speedup specific queries in a data warehouse environment. Materialized views pre-compute and store (materialize) aggregates from the base data [Chaudury97]. The data is grouped using categories from the dimensions tables, which corresponds to the subjects of interest (dimensions) of the organization. Storing all possible aggregates poses storage space problems and increases maintenance cost, since all stored aggregates need to be refreshed as updates are being made to the source databases. Many algorithms have been proposed for selecting a representative subset of the possible views for materialization [Harinarayan96, Meredith96, Ezeife98], corresponding to the most usual query patterns. But the main problems associated with materialized views are the difficulty to know in advance the expected set of queries, the problems of updating materialized views to reflect changes made to base relations and the large amount of space required to store the materialized views.

It is worth noting that the techniques mentioned above (indexes and materialized views) are general techniques that can (and should) be used in the data warehouse approach proposed in the present paper. In fact, in a distributed environment each individual machine must also be tuned and optimized for performance.

A large body of work exists in applying parallel processing techniques to relational database systems with the purpose of accelerating query processing [Lu94, DeWitt92]. The basic idea behind parallel databases is to carry out evaluation steps in parallel whenever possible, in order to improve performance. The parallelism is used to improve performance through parallel implementation of various operations such as loading data, building indexes and evaluating queries. One of the first works to propose a parallel physical design for the data warehouse was [Datta98]. In their work they suggest a vertical partitioning of the star schema including algorithms but without quantifying potential gains.

Partitioning a large data set across several disks is another way to exploit the I/O bandwidth of the disks by reading and writing them in a parallel fashion. User queries have long been adopted for fragmenting a database in the relational, object-oriented and deductive database models [Ozsu99, Ezeife95, Lim96]. That is, the set of user queries of a database is indicative of how often the database is accessed and of the portion of the database that is accessed to answer the queries. There are several ways to horizontally partition a relation. Typically, we can assign tuples to processors in a round-robin fashion (*round-robin partitioning*), we can use hashing (*hash partitioning*), or we can assign tuples to processors by ranges of values (*range partitioning*).

Most of today's OLAP tools require data warehouses with a centralized structure where a single database contains all the data. However, the centralized data warehouse is very expensive because of great setup costs and lack of structural flexibility in data storage.

More importantly, "the world is distributed'": world-wide enterprises operate in a global manner and do not fit in a centralized structure. Thus, a new paradigm is necessary. The first step in a new direction was the recent introduction of data marts, "small data warehouses" containing only data on specific subjects ([Informatica97, HP97]). But this approach doesn't solve our problems of space and performance. Data marts provide more flexibility in the distribution of data but they still consist of static, self-contained units with fixed locations. By distributing small static portions of data to fixed locations, the system becomes more flexible, but on the other hand new problems arise, related to intra data mart communication, especially in what concerns the processing of queries. Many of today's data marts are basically stand-alone, because of the unsophisticated and rudimentary integration in the global data warehouse context.

In spite of the potential advantages of distributed data warehouses, especially when the organization has a clear distributed nature, 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. Furthermore, each individual data mart is primarily designed and tuned to answer the queries related to its own subject area and the response to global queries is dependent on global system tuning and network speed.

The proposed approach 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 query processing load is uniformly distributed by all the available computers and, at the same time, the need of communication among computers during the query execution is kept to a minimum.

4 Proposed approach and preliminary results

Before presenting our approach let us briefly review the star schema and some key aspects of a data warehouse implemented over a relational repository. In OLTP environments a highly normalized schema offers superior performance and efficient storage because only a few records are accessed by a particular transaction. The star schema, provides similar benefits for data warehouses where most queries aggregate a large amount of data. A star schema, as illustrated in Figure 1, consists of a large central fact table and several smaller dimension tables that are related to the fact table by foreign keys.

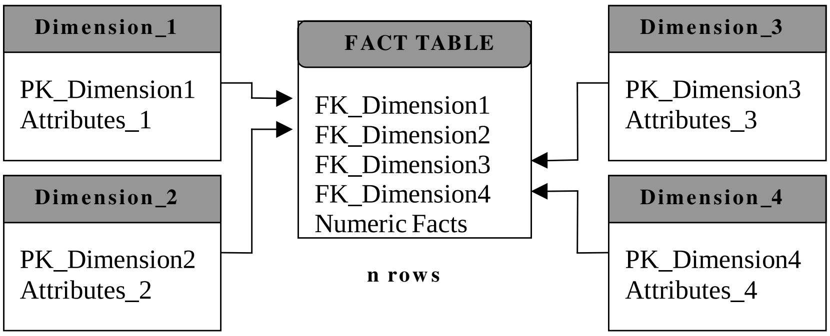

Figure 1: Example of a star schema.

The fact table is the center of a star schema, which has predominantly numeric data (the facts) such as counts and amounts. The dimension tables have descriptive attributes, providing information about each row in the fact table. Typical queries calculate sums, counts, maximums, minimums, etc, by aggregating the data facts according to attributes of one or more dimensions. Normally, one or more star schemas compose a data warehouse, each star corresponding to a business process.
The fact table accounts for most of the space occupied by all the tables in the star for most of the cases [Kimball96]. However, this table is highly normalized, which means that it really represents the most effective (concerning storage space) relational way to store the facts. The dimension tables are very denormalized, but these tables usually represent a small percentage of the space in the star. The star schema is also particularly optimized for the execution of complex queries that aggregate a large amount of data from the fact table.
A star schema provides an intuitive way to represent in a relational system the typical multidimensional data of businesses.

4.1 Data Warehouse Striping

To solve the problems associated with centralized data warehouses we propose a new approximation that distributes the data warehouse. Our approach is inspired in the RAID-3 [Patterson88] scheme, and that is why we call it data warehouse striping (DWS). In fact, the small size of the chunks (strips) used in RAID-3 assures that any disk access is uniformly distributed by all the disks of the array, improving the disk access speed by the number N of disks in the array. In a similar way, one major goal of data warehouse striping is to improve query execution speed by the number N of computers in a DWS system.
In the data warehouse striping approach, the data of each star schema is distributed by an arbitrary number of computers that have the same star schema of the equivalent centralized version (see Figure 2).

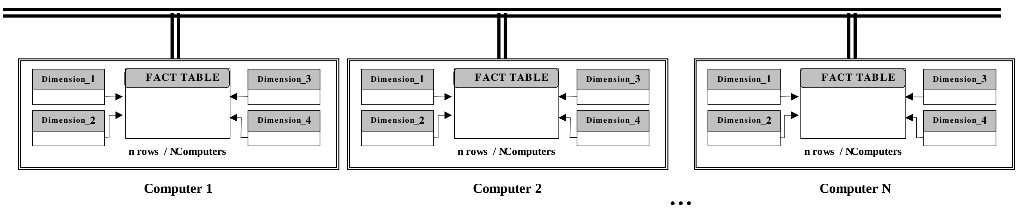

Figure 2: Data Warehouse Striping.

The usually much smaller dimension tables of the star schema are replicated in all computers that constitute the DWS system. The fact table of equivalent centralized version is distributed by the fact tables of each computer using a round-robin partitioning approach. That is, the fact rows are evenly distributed by the fact tables of each computer. Each computer has 1/N rows of the total amount of fact rows in the star, with N being the number of computers.
In data warehouse striping the fact tables are disseminated by an arbitrary number of computers and the queries are executed in parallel by all the computers, guarantying a nearly linear speedup and significantly improved query response times. Figure 3 illustrates this process using an example with five computers.

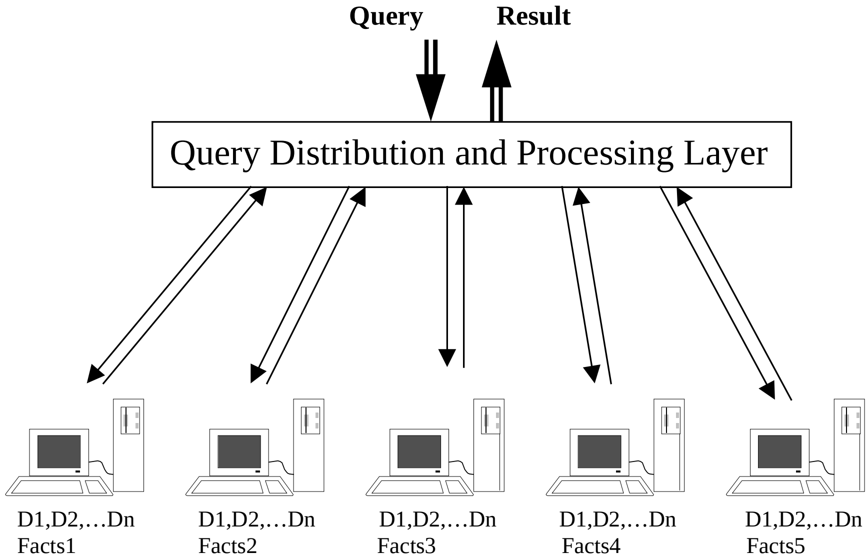

Figure 3: Distributed query execution using five computers.

Typical queries usually filter rows based on any number of dimensional attributes and aggregate the corresponding measures from the fact table. By distributing the often huge fact table (which usually represents more than 90% of the space occupied by all the tables in the star schema) by several computers, a significant gain can be achieved in the speed of the query.
The DWS system has the following phases:

1) ***Distributed loading phase***
   In this phase the dimensions are replicated in all computers and the fact tables rows are evenly distributed in a round robin fashion between all the computers.
2) ***Query distribution phase***
   A query is transformed into N partial queries that are executed in an independent way in each of the N computers. Generically, this phase distributes the same query to all computers but some types of queries require query re-writing.
3) ***Query processing phase***
   In this phase the partial queries are simultaneously executed in all computers; this distributed execution is one of the main reasons for the speedup and superior scalability offered by the approach.

4) Result merging phase

Generically, this phase collects the small partial results from the computers, “merges” them and delivers the final result to the end user.

The query distribution and processing layer is a software layer that implements the last three phases described above. The first phase, distributed loading phase, is independent from the others; it is executed once at the beginning of the load of the data warehouse data and only when more data need to be added it must be executed again.

4.2 Research methodology

We have used the APB-1 Benchmark of the OLAP Council [APB98] to validate our distributed data warehouse. The use of this benchmark allows other researchers to compare our results.
The APB-1 benchmark simulates a realistic On-Line Analytical Processing (OLAP) business situation that exercises server-based software and a simplified star schema is illustrated in figure 4.

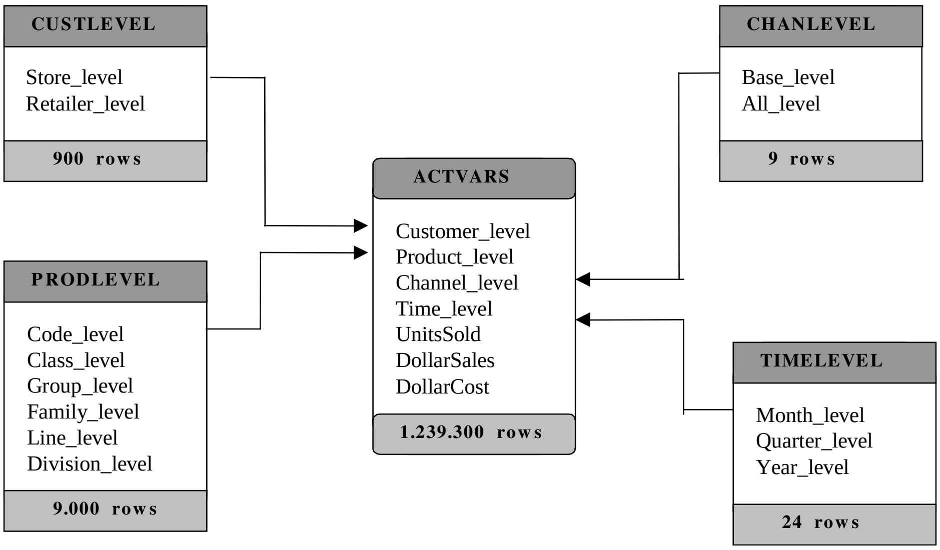

Figure 4: APB-1 simplified star schema.

The experimental setup was made in a PC environment, where all the computers have the same characteristics and are linked together in an Ethernet network. The fact table (ACTVARS) is evenly distributed by the five computers, where each one of them received 247,860 fact rows. The dimension tables (PRODLEVEL, CUSTLEVEL, TIMELEVEL and CHANLEVEL) are replicated in all five computers.

4.3 Preliminary results

We have made some experiments comparing the performance using one and five computers with the same characteristics. The results with one computer (1PC) simulate the centralized data warehouse and 5PC is data warehouse striping applied to five computers. The results of figure 5 illustrate the case of using any subset of the aggregate functions sum, count, max and min with this type of query:

select *aggregate_function*(unitssold) from actvars

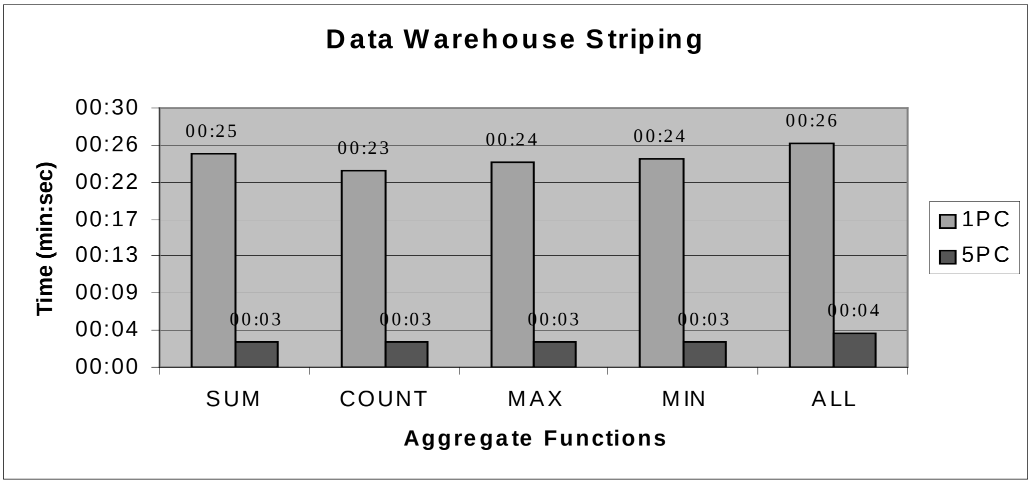

Figure 5. Results for simple fact aggregation.

From the figure we measure an average gain of 4.71 times between using one computer (1PC) and with DWS using five computers (5PC). This near optimal speedup is explained by the fact that the partial queries only access a small part of the fact table. We also tested data warehouse striping with more realistic queries like the following:

```
select p.division_level, count(dollarsales), sum(a.unitssold)
from actvars a, prodlevel p, timelevel, custlevel, chanlevel
where
< ... join conditions ... >
group by  p.division_level
order by p.division_level
```

Figure 6: Example of 4-join query.

This query makes a join of the fact table with all four dimensions. This is a more realistic decision support query and the figure 7 illustrates the results obtained.

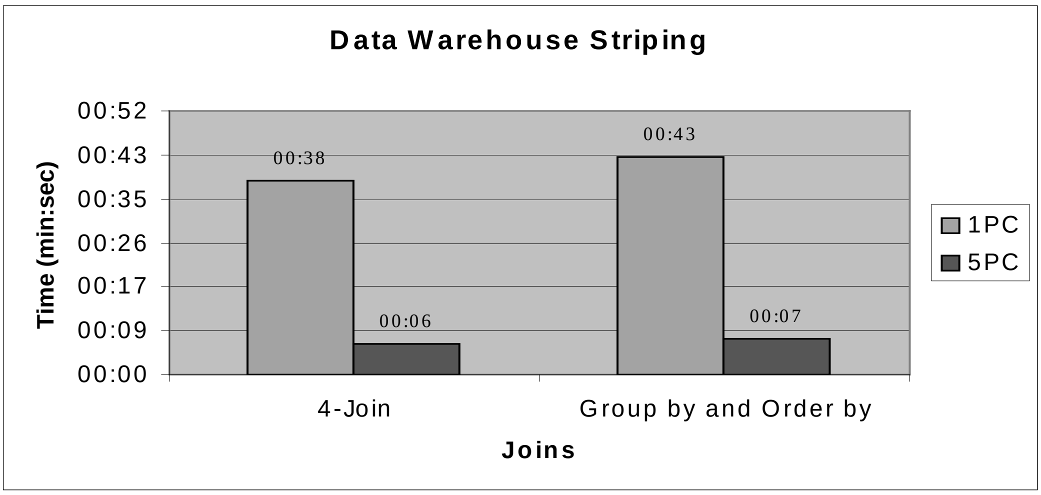

Figure 7: Results using a 4-join query.

The first two columns of figure 7 shows the result of the query of figure 6 excluding the group by and order by portion of the query, while the last ones depict the results of the query as is. In this case we have a potential gain of about 4.75 times using five computers, near the optimal value of 5. This could be explained by the gain obtained by distributing the 4-join between our five computers, meaning that when we distribute the data we are working on more manageable data sets.

5 Identified problems

The proposal seems to provide a very promising contribution to the scalability and efficient processing of huge data warehouses. However, we have identified some problems that must be dealt with:

- Query distribution phase
    - Queries that contain aggregation functions such as AVERAGE, STANDARD DEVIATION or VARIANCE must be transformed into new queries. For example an average function is transformed into SUM and COUNT functions.
- Result merging phase
    - For some simple aggregating functions such as sums, or counts the layer collecting the results simply has to sum the partial results. But, queries that returns more than one row (e.g with group by predicates) also require post-processing in the merging phase.
    - Correlated queries also pose difficulties.

These problems that were identified must be dealt within the Query Distribution and Processing software layer that is currently been developed.

- Big dimensions

  Typically, dimensions will be small in size. However, there are exceptions to this rule, in which case the space overhead becomes more significant.

6 Contributions

The proposed DWS architecture is scalable to support gigabytes or even terabytes of information, solving the traditional problems of data warehouses related to heavy storage loads and performance problems. As the need for additional data stores arises, only additional computers need to be added. The problem of scalability is pointed out by Jim Gray, in his Turing Lecture Award as the first long range IT systems research goals [Gray99].

We think that our proposal for distributed data warehousing affords several advantages to businesses, including suppliers and distributors. First and foremost, it is a cost-effective solution. By initially investing in a local Windows NT server, a department could build a small individual data store that could later be connected to additional ones, forming a “distributed” data warehouse. Instead of undertaking the painstaking task of integrating them all into one central store, the two or more could be virtually combined using network hardware and software.

The main contributions of this work are the following:

- A new approach for improving query response time in data warehouses named data warehouse striping. DWS can potentially achieve linear speedup and can significantly improve query response times.
- A simple but efficient algorithm for distributing the fact table solving the problems posed by its large volume.
- Takes advantage of the availability and low cost of microcomputers to interconnect these computers into networks which provide the distributed and parallel processing capabilities needed for handling very large data warehouses.
- This modular approach can be incorporated into a relational DBMS without significant modifications.

This technique also has the following advantages:

- Potential gains in performance;
- Large scalability;
- Approximately ideal load balancing between the processors;
- Reduction in equipment price because we could choose computers with the best cost/performance relation;
- Takes advantages of star schema characteristic of data warehouses;

7 Future work

The approach we have presented is preliminary, leaving open some issues. Future work in this area includes the implementation of software layer that handles the partial results and sends the result to the end user. We intend to test the approach increasing the size of the APB-1 Benchmark and using a different number of computers. It is also important to test the idea using another data set like benchmark TPC-H [TPC99].

Another future direction is to explore some peculiarities of the multidimensional schema to improve our partitioning. We are also working on other partition strategies such as vertical partitioning of the fact table according to its dimensions.

The availability of computers that form a distributed data warehouse is a factor of prior importance. We are studying strategies in order to make the DWS system robust enough to cope with the momentarily unavailability of one of more computers.

References

[Albrecht98] Jens Albrecht, Holger Gunzel, Wolfgang Lehner, "An Architecture for Distributed OLAP", International Conference on Parallel and Distributed Processing Techniques and Applications PDPTA'98, 1998.

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

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

[Comer79] D. Comer, "The ubiquitous B-tree", ACM Computing Surveys, 11(2):121-137, 1979.
[Datta98] Anindaya Datta, Bongki Moon and Helen Thomas, "A Case for Parallellism in Data Warehousing and OLAP ", Proceedings of the 9th International Conference on Database and Expert Systems Applications (DEXA'98), September 1998.
[DeWitt92] D.J. DeWitt, Jim Gray, "Parallel Database Systems: The future of high performance database systems", Comm. of the ACM, 35(6), June 1992, pp.85-98.
[Ezeife95] C.I. Ezeife, K. Barker, "A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System", Distributed and Parallel Databases, 1:247-272, 1995.
[Ezeife98] C.I. Ezeife. "A Partition-Selection Scheme for Warehouse Aggregate Views", Int. Conf. of Computing and Information, Manitoba, Canada, June, 1998.
[Gill00] Philipp J. Gill, "SIZE: Breaking the Warehouse Barrier", Oracle Magazine, January/February 2000, Volume IX, Issue 1, pp. 38-44.
[Gray99] Jim Gray, "What Next? A Dozen Information-Technology Research Goals", Turing Talk Award, June 1999, Technical Report MS-TR-99-50.
[Harinarayan96] Venky Harinarayan, Anand Rajaraman, and Jeffrey Ullman. Implementing Data Cubes Efficiently. In ACM SIGMOD Proceedings, June 1996.
[HP97] "HP Intelligent Warehouse", Hewlett Packard white paper, *http://www.hp.com*, 1997.
[Informatica97] "Enterprise-Scalable Data Marts: A New Strategy for Building and Deploying Fast, Scalable Data Warehousing Systems", Informatica white paper, *http://www.informatica.com*, 1997.
[Jurgens99] Marcus Jurgens and Hans-J. Lenz, "Tree Based Indexes vs. Bitmap Indexes: A Performance Study", Intl. Workshop DMDW'99, Heidelberg, Germany, 1999.
[Kimball96] The Data Warehouse Toolkit, Ralph Kimball, Ed. J. Wiley & Sons, Inc, 1996.
[Lim96] S.J. Lim and Y.-K. Ng, "A Formal Approach for Horizontal Fragmentation in Distributed Deductive Database Design", Int. Conference on Database and Expert Systems Applications (DEXA'96), pp234-243, Zurich, Switzerland, September 1996.
[Lu94] Hongjun Lu, Beng Chin. Ooi, and Kian Lee Tan, Query Processing in Parallel Relational Database Systems, IEEE Computer Society, May 1994.
[Meredith96] M.E. Meredith and A. Khader, "Divide and Aggregate: Designing Large Warehouses", Database Programming and Design, 9(6), June 1996.
[O'Neil97] Patrick O'Neil and Dallan Quass, "Improved Query Performance with Variant Indexes", ACM SIGMOD Proceedings, 1997, pp. 38–49.
[Ozsu99] M. Ozsu and P. Valduriez. Principles of Distributed Database Systems. Second edition, Prentice-Hall, New Jersey, 1999.
[Patterson88] D. A. Patterson, G. Gibson, R. H. Katz, "A case for Redundant Arrays of Inespensive Disks (RAID)", ACM SIGMOD Proceedings, June 1988.
[TPC99] TPC Benchmark H, Transaction Processing Council, June 1999. Available at *http://www.tpc.org/*.
[Wu97] M. Wu and A. Buchmann, "Research Issues in Data Warehousing", BTW'97, March 1997.