December 2001

Data Warehousing and OLAP in a Cluster Computer Environment

Amit Rudra
Curtin University of Technology

Raj Gopalan
Curtin University of Technology

DATA WAREHOUSING AND OLAP IN A CLUSTER COMPUTER ENVIRONMENT

Amit Rudra
Curtin University of Technology
rudraa@cbs.curtin.edu.au

Raj Gopalan
Curtin University of Technology
raj@cs.curtin.edu.au

Abstract

Decision oriented technologies, like data warehousing and on-line analytical processing (OLAP) systems store and handle very large volumes of data, requiring more efficient ways of dealing with them. Recent advances in parallel computing and high-speed networks using a cluster of PCs or workstations (COWs) offer a low cost solution for providing this scale up in performance by parallelism of data, and it's processing, in the data warehouse. This paper investigates how the star join and data cube operations can be performed in parallel on a cluster of Pcs.

Introduction

With the availability of increasingly powerful processors and large amount of memory (RAM), PCs are becoming more and more viable for solving serious problems on them. It is no wonder, therefore, that clusters of PCs are becoming quite serious contenders for the parallel systems market (Baker and Buyya 1999). When a number of PCs are connected by a high-speed LAN and a suitable software is put on them the ensuing cluster thus formed provides enormous potential for solving high-perfomance demanding computing problems. Lately, data warehousing as an application has generated considerable research interest both among academics and the industry (Chaudhuri and Dayal 1997). If properly implemented and used, it offers great advantages for business and industry. It also presents significant challenges to the research community as with growing gigabytes of data added to it every week or month, a data warehouse has the potential to quickly overflow the available storage space. In this paper, we look at investigate the issues involved in implementing a data warehouse in a cluster computer environment using PCs.

A data warehouse can be defined as an online archive of historical enterprise data that is aimed at enabling the knowledge worker (executive, manager, analyst) make better and faster decisions (Chaudhuri and Dayal 1997). The predominant reason for using a data warehouse is better performance in the management of historical data. Usually, a typical business analyst needs to perform complex analyses and reporting of data. To help with such analyses, the data in a warehouse is typically modeled multidimensionally. For example, in a product sales data warehouse, date, product code, region, salesperson and customer might be some of the *dimensions* of interest; detailed information regarding each sale (total price, quantity etc.) are termed *measures* and can be kept in a large fact table (Chaudhuri and Dayal 1997, Gray et al 1997). Figure 1 shows an example of such a database schema, called a *star schema* (Gray et al. 1997). The fact table (SALES) schema is in the middle, while the schema of dimension tables (PRODUCT, DATE and REGION) are shown on either side. The figures at the bottom of the table attribute list represent number of rows in them. It was adapted from the TPC Benchmark-R (Transaction Processing Council 2000) specifications for our test data warehouse.

Since data warehouses maintain predominantly data of historical nature, there is ever increasing demand for storage and processing speed. Of the three possible solutions to this problem (faster processors, faster/efficient algorithms and extra help in the form of parallel processing), parallel computing holds the most promise for scaleup needed by such applications (Baker and Buyya 1999, Pfister 1998). With recent advances in networking and the availability of increasingly powerful PCs and workstations, a cluster of workstations offer a solution that is worth exploring. Clusters, in general offer the following advantages (Pfister 1998):

- performance: adding more workstations to a cluster adds to the computing power;
- availability: a node in a cluster is not indispensable, as another can replace it;
- price/performance: a PC gives a better price/performance ratio than a comparable mainframe parallel system;

- incremental growth: the cluster can add nodes, and can grow as and when required;
- scalability: there is, virtually, no limit to how many machines can be clustered together;

However, there are some limitations in using clusters for large volume applications:

- Data Travel: As compared to the single and parallel processors, data in a cluster environment have to travel some distance over the network. Too frequent and voluminous data movement between the nodes could slow down an application.
- Update/Refresh: For applications requiring frequent updates of data from the source, it could lead to more network traffic over the LAN.
- Data Storage: There are still limitations to data that can be stored on the master or controlling node.

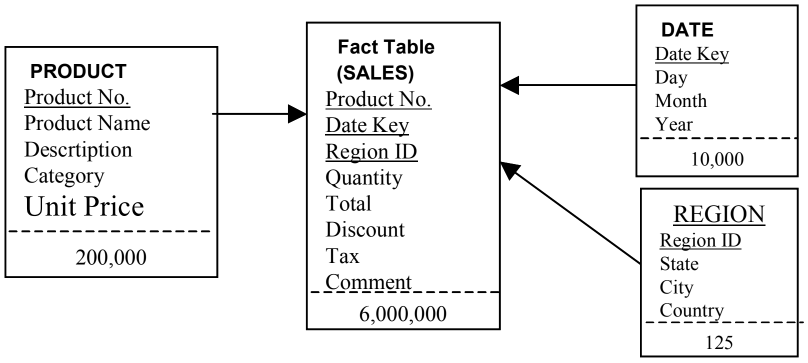

Figure 1. Star Schema of Sales Data Warehouse

While the above is a brief discussion of some of the factors that could affect the performance of a cluster in solving data warehousing and OLAP applications, these cannot be ignored and need closer investigation.

Research in the area of parallelism of data in a data warehouse has been very limited. The need for providing parallelism of a number of tasks for such an application has been highlighted in the work of Garcia-Molina et al (1998). There have also been some other reported research in the area (Datta et al. 1998, Goyal et al. 1999), but none of them deal with the cluster computing environment. In this paper, we propose a scheme for the *parallel star join* in such an environment. Star schema refers to a simplified relational schema where all dimension tables are related to a single fact table, thus giving it a logical star shape. In our example shown in Figure 1, the fact table SALES when combined with data of all the dimensions (REGION, TIME and PRODUCT) it is termed the star join. However, to affect a star join computation has to be done. This could be computationally expensive as the cardinality of the fact table and those of some of the dimension tables could be quite high. Performing this computation in parallel is called a parallel star join.

The main reason for performing a star join is to calculate what is called a *data cube* or simply the cube (Gray et al. 1997). For a given set of dimensions and measures the cube represents reporting the summaries of the measures for all combinations of the dimensions. Conceptually, the result of an OLAP query results in a multidimensional hypercube. In this paper, we propose parallel star join algorithm and also a parallel data cubing algorithm. To the best of our knowledge data warehousing in a cluster environment has not been studied previously.

The organisation of the rest of the paper is as follows: In Section 2 we describe briefly the concept of a cube and some of the other basic terms used in data warehousing and OLAP. The contributions of our research are presented in Section 3. Finally, conclusions and directions for further research are provided in Section 4.

Data Warehousing and OLAP

As pointed out in section 1, a data warehouse typically stores large amounts of data in a format suitable for decision-oriented (i.e. analytical) type of queries providing helpful visualisation of the data to the business analyst. Typical OLAP operations would not only need to retrieve large number of rows from different tables but may also need to match rows from one table with those from another. These matching operations in turn may require sorting the selected rows from different tables (O'Neil and Quass 1997, Graefe 1993). For speedy access to very large data, indexing is one of the most commonly used techniques for large data volume applications (Mumick et al. 1997, Wilschut et al 1995, Zhao et al. 1998). In the following sub-sections we provide a brief discussion of the indexing method used in our research.

Join Index

A join index is an index on one table for values that match a column value of another table (Goyal et al. 1999, O'Neil and Quass 1997). As data warehouses are usually updated periodically, most join indexes represent fully precomputed joins. This is done in batch mode, when the warehouse is unavailable for querying (O'Neil and Quass 1997). This implies that a batch update process can reorganise both data and indexes for efficient access, which would not be possible if the indexes were in continuous use as in an OLTP environment. It is therefore, possible to use specialised indexes in such an environment. We now describe one of these special techniques.

Join Value-list Index (JVI)

In a relational data warehouse information is organised in relational tables. The queries naturally involve relating two or more tables. Thus, a typical query on sales figures for different regions would involve joining the dimension table REGION with the fact table SALES. As is well known, such a join could be expensive as each row from REGION table needs to be matched against rows from the SALES table. A better way would be to precompute the related SALES rows for each row value of the REGION and store the resulting rowset as a list of Row-Ids (RIDs). Since, most of the OLAP queries are by one or more of the dimensions (e.g. list by year, product, region etc.), we consider it is more efficient storing the join RIDs in this fashion. We term this a JVI or Join Value-list Index. Physically, this can be stored as a B+-tree (O'Neil and Quass 1997). Also, variants of Bitmap indexes can be used to speed up small cardinality dimension tables. Figure 2 shows an example of a JVI.

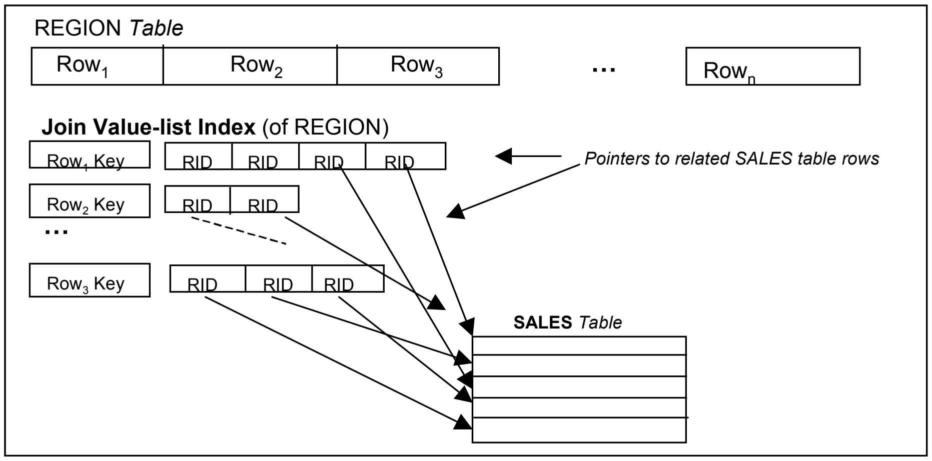

Figure 2. A Join Value-List Index (JVI)

OLAP

OLAP or online analytical processing is the technique or tool used to facilitate analytical queries on the data warehouse. Depending on the data structure used by the underlying data warehouse an OLAP tool is termed relational (ROLAP) or multidimensional (MOLAP). The requirements of these could be quite different from the normal SQL queries, as the decision maker may need to *drill* down from a level of aggregation by querying the details even further or go up to a higher level of aggregation of the data to perform the desired analysis (Gray et al 1997).

OLAP Operations: The operations of OLAP, the methods that allow users to efficiently retrieve data from a data warehouse, are quite different to those of on-line transaction processing (OLTP). Typical OLAP operations include *rollup* (increasing the level of aggregation; for example, sales for different categories rather than by items) and *drill-down* (decreasing the level of aggregation or increasing the detail; e.g. in Table 1, sales for each city within a state) along one or more dimension hierarchies, *slice-and-dice* (selection and projection) and *pivot* (re-orienting the multidimensional viewing of data) (Chaudhuri and Dayal 1997, Gray et al 1997). It is important to point out that due to these differences in the nature of applications and operations on a data warehouse, the typical operational database would not be able to provide an acceptable level of performance for OLAP queries. The data required for decision support may also be missing from such a database (for example, to show trends in sales for a certain state or city over the years we need historical data which is not available in a typical operational database).

The Data Cube

Besides the performance issues pointed out above, the standard SQL GROUP BY clause may not be appropriate for producing reports. Standard SQL lacks the commonly used data analysis facilities like: histograms, roll-up totals and sub-totals for drill-downs, and cross tabulations. Gray et al (1997) suggested using a CUBE operation on top of GROUP BY to facilitate such report formation. The difficulty of using SQL to perform the cube operation is illustrated by the following example:

```
SELECT  Year, 'ALL', 'ALL', SUM (Total)
     FROM  Sales
     WHERE Year = 1997
UNION
SELECT  Year, State, 'ALL', SUM (Total)
     FROM  Sales
     WHERE Year = 1997
     GROUP BY State
UNION
SELECT  Year, State, Category, SUM (Total)
     FROM  Sales
     WHERE Year = 1997
     GROUP BY State, Category;
```

This is a simple 2-dimensional rollup. Aggregating over N dimensions would require N such unions, which would not be so elegant a query. Using the CUBE operator to solve this problem, the query can be expressed as follows:

```
SELECT  Year, State, Category, SUM (Total)  AS  'Total Sales Amount'
     FROM  Sales
     WHERE Year = 1997
     GROUP BY  CUBE  State, Category;
```

Star Join and Data Cube in a Cluster Environment

In this section we discuss various issues of data warehousing in a typical cluster environment. A very important requirement is to divide or partition the data into several parts, sometimes referred to as declustering. First, we consider a data partitioning scheme and then explore how data can be propagated to various nodes of the cluster. It is followed by our algorithms for parallel star join and data cube operations.

Data Partitioning and Node Allocation

In order to exploit the power of a cluster viz. parallel computation, the data warehouse needs to be partitioned. Ideally, to utilise the storage (both primary and secondary) and processing power of each node or PC on a cluster, the data and processing on all nodes should be balanced. We assume that the nodes in the cluster have the same processing speed, memory and secondary storage. First, we group the dimension tables into groups viz. small or large. The smaller dimension tables are those that can be stored on a single node. As explained in the research of O'Neil and Quass (1997), this may be determined based on the cardinality of the table. If it is below a certain threshold, it is termed small. Dimension tables that are large and the fact table are partitioned horizontally among the rest of the available nodes of the cluster. In order to speed up joins we use the join value-list index or JVI as discussed in the previous section. A JVI stores the RIDs or row numbers of the fact table that are related to a given dimension row. Storing JVI along with the dimension rows would be beneficial in accessing the related information for performing GROUP BY operations typical of OLAP. However, for small dimension tables we use bitmap indexes as this is much more efficient for performing AND and OR operations. Thus, when a dimension table is partitioned horizontally for placement on a number of nodes, the corresponding JVI will also be partitioned. Figure 3 shows the scheme for partitioning the dimension table among the nodes.

Algorithm 1 specifies how the tables are partitioned for placement on different processor nodes. After allotting k nodes to smaller dimension tables (Step 1), the remaining n-k dimensions and the fact table are allotted n-k+1 groups of nodes from the remaining N-k nodes of the cluster. Thus, we allot *floor*((N-k)/(n-k+1)) nodes to each of the n-k large dimension tables and the remainder nodes, i.e. *floor*((N-k)/(n-k+1)) + *remainder*((N-k)/(n-k+1)), are allotted to the fact table (Step 2). Next, we partition horizontally the large dimension tables and load them (Step 3). The fact table is partitioned next (Step 4), into *floor*((N-k)/(n-k+1)) + *remainder*((N-k)/(n-k+1)) horizontal partitions and the data loaded. To keep track of the allotments we need to maintain a mapping between the table partitions and cluster nodes (Step 5).

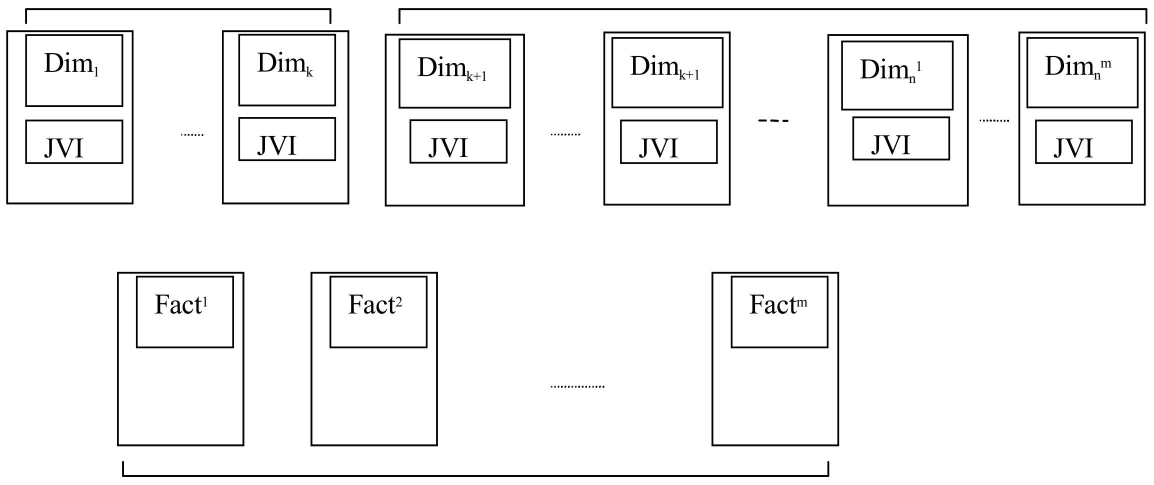

Figure 3. Data Partitioning Scheme

Algorithm 1. Allocation of Processors and Dimensions to Processor Nodes

Input: 1. $D = \{D_1, D_2, \ldots D_n\}$: set of dimensions of the data warehouse ($n < N$ the no. of nodes)
2. F : fact table
3. J: set of join value-list indexes of all dimension tables
4. $P = \{P_1, P_2, \ldots P_N\}$: set of processor nodes

Output: 1. Dimension tables allocated and stored on cluster nodes
2. M: Mapping of the set of dimensions and fact table to nodes

1. Allot nodes $P_1, P_2, \ldots P_k$ of the cluster to the small dimensions: $D_1, D_2, \ldots D_k$.
   (Assume the first k dimensions are small).
2. Allot rest of the available nodes to n-k large dimensions and the fact table such that (N-k)/(n-k+1) processor nodes are allotted to each dimension i (k< i <=n). In case (N-k)/(n-k+1) has a remainder i.e. does not divide integrally into k groups, allot *floor* ((N-k)/(n-k+1)) no. of nodes to each dimension. Allot remainder nodes to the fact table.
3. Partition horizontally each large dimension table and corresponding JVIs into *floor* ((N-k)/(n-k+1)) partitions and load them.
4. Partition horizontally the fact table into *floor*((N-k)/(n-k+1)) + *rem*((N-k)/(n-k+1)) partitions and load them.
5. Form M: mapping array of table allotments to nodes. (Determines which node(s) a dimension or fact table is mapped to).

Star Join

To compute the star join, we first find out the rows of the fact table that will be participating in the final cube grouping. Since restriction predicates reduce the number of rows in the final result table, we utilise them first. The following query shows how restriction predicates limit the number of rows in the result table:

```
SELECT  Year, State, Product_Name, SUM (Quantity)  AS  'Total Quantity'
    FROM  Product P, Date D, Sales S, Region R
    WHERE P.Product_No = S. Product_No  AND  D.Date_key = S.Date_Key  AND
        R.Region_ID = S. Region_ID  AND
        D.Year IN (1998, 1999)  AND  P.Category IN ('Printer', 'Scanner')
    GROUP BY  CUBE  Year, State, Product_Name;
```

Assuming the data warehouse (Figure 1) maintains sales records of 10 years, we can straight away notice that only the last two years' sales rows will be picked from the fact table. The second restriction predicate on product category effects a further reduction.

We use a simple method viz. to form rowsets (of the fact table) for one restriction predicate. The restriction predicates are applied to their corresponding dimensions in parallel. The JVIs that satisfy each predicate are passed on to a coordinator node where an intersection of the RIDs of all the restricted dimensions is performed. The resulting rowset (set of row IDs) thus formed is the result of the star join. The dimensions for which no restriction predicates are present do not participate in this operation as they cannot cause any reduction to the resulting rowset. While operating on the fact table we try to distribute equally to each node available as much processing as is possible, so that I/O and computation could be performed efficiently in parallel. Algorithm 2 shows how the parallel star join is performed.

Algorithm 2. Parallel Star Join

```
Input:  1. $P = \{P_1, P_2, \ldots P_N\}$:  nodes with dimension and fact tables
        2. $M$:        Mapping array of dimensions to nodes
        3. $P_\sigma$:       set of restriction predicates
        4. $J$:        JVIs
Output: 1. $R_{DC}$:     resulting rowset (of fact table) from the star join that participate in data cubing.
        2. $O_{DC}$:     aggregate operations (sum, average etc.) and metric attributes

1. Pick the dimensions with restrictions, if any. Else, if none, pass 'all rows' indication to master and skip steps 2-6.
2. For each node corresponding to restriction dimensions perform steps 3-4 in parallel.
3. If dimension is large, determine coordinator node that will form the resulting rowset from this dimension.
4. Build a list of JVIs for those rows that satisfy the restriction condition(s) on the dimension in parallel.
5. For partitioned dimensions, pass the resulting rowset to the coordinator node for each restriction predicate and
   consolidate.
6. Accumulate resulting rowset for each restriction on a coordinator node.
7. Consolidate – form resulting matrix from arrays of rowsets sent by nodes corresponding to restriction dimensions.
8. Form rowset of RIDs common to all restrictions at the coordinator node.
```

The Cube Operation

Since the whole computation of the data cube needs to be well coordinated, we follow a coordinator oriented scheme in designing the parallel programs. For example, distribution of the resulting rows from *star join* needs to be coordinated so that they could be collected by one node in forming the final data cube. Algorithm 3 shows the scheme for computing the cube. The rowset of fact table resulting from the star join is distributed to the nodes holding the fact table partitions. The aggregation required are performed at these nodes in parallel. The results are then passed on to the coordinator for consolidation and presentation.

Algorithm 3. Parallel Data Cube

```
Input:  1. $R_{DC}$: resulting rows (of fact table) from the star join that participate in data cubing.
        2. $O_{DC}$:     aggregate operations (sum, average etc.)
Output: 1. Query report

1. Distribute rowset of fact table to be worked on to each participating node.
2. For each node, retrieve rows from fact table if not already in memory.
3. Perform required aggregation.
4. Pass result of aggregation to coordinator.
5. Sort results by the required dimensions.
6. Consolidate final ordering of result and present it.
```

Conclusion

In this paper we have presented three significant algorithms for implementing data warehousing and OLAP on a cluster of PCs. These include partitioning and distribution of data to different nodes of the cluster, the computation of star join and cube for answering OLAP queries. At present we are in the implementation stage and expect the results to be available for reporting soon. Our future research will involve further refinement of the star join and cube algorithms; and comparison of their performance with other existing sequential and parallel implementation schemes. We also plan to test the different indexing schemes: projection indexes, bitmap indexes and bit-sliced indexing strategies for accessing the data, with a view to comparing their relative performance in the computation of both star join and cube operations.

References

Baker, M., Buyya, R. Cluster Computing: The Commodity Supercomputing. In *Software-Practice and Experience, Vol. 1 (1), 1-4,* Jan., 1999.

Bruck, J., Dolev, D., Ho, C-T., Rosu, M-C., Strong, R. Efficient Message Passing Interface (MPI) for Parallel Computing on Clusters of Workstations. *Papers of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures.* 1995, pp. 64 - 73.

Chaudhuri, S., Dayal, U. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record,* March 1997.

Datta, A., Moon, B., Thomas, H. A Case for Parallelism in Data Warehousing and OLAP. *DEXA '98 - 9th International Workshop on Database and Expert Systems Applications, Vienna, Austria* ., 1998.

Garcia-Molina, H., Labio, W. J., Weiner, J. L., Zhuge, Y. Distributed and Parallel Computing Issues in Data Warehousing. Invited talk in *Proceedings of the Seventh Annual Symposium on Principles of Distributed Computing (PODS '98)*, 1998.

Goil, S., Choudhary, A. Design and Implementation of a Scalable Parallel System for Multidimensional Analysis and OLAP. *Proceedings of International Parallel Processing Symposium*, Puerto Rico, 1999.

Goyal, K., Ramamritham, K., Datta, A., Thomas, H. Indexing and Compression in Data Warehouses. *Proceedings of the International Workshop on Design and Mangement of Data Warehouses (DMDW'99),* Heidelberg, Germany, 1999.

Graefe., G. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys, Vol. 25(2),* 1993, pp. 73-170.

Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F., Pirahesh, H. Data Cube: A Relational Operator Generalising Group-By, Cross-Tab and Sub-Totals. *Data Mining and Knowledge Discovery, Vol 1,* Kluwer Academic Publishers, The Netherlands, 1997, pp. 29-53.

Mumick, I. S., Quass, D., Mumick, B. S. Maintenance of Data Cube and Summary Tables in a Warehouse. *Proceedings of the VLDB Conference*, 1997.

O'Neil, P, Quass, D. Improved Query Performance with Variant Indexes. *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data* , 1997, pp. 38-49.

Pfister, G. F. *In Search of Clusters.* 2nd Ed. Prentice Hall PTR, NJ, 1998.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J. *MPI: The Complete Reference* Vol.1, 2nd Ed. The MIT Press, Cambridge, MA, 1998.

Transaction Processing Performance Council, 2000 – http://www.tpc.org [Accessed May 2001].

Wilschut, A. N., Flokstra, J., Apers, P. M. G. Parallel Evaluation of Multi-join Queries. *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data ,* 1995, pp. 115 -125.

Zhao, Y., Deshpande, P. M., P., Naughton, J. F., Shukla, A. Simultaneous Optimization and Evaluation of Multiple Dimensional Queries. *Proceedings of the VLDB Conference*, 1998.