Database Query Distribution over Intelligent Networks*

Naphtali Rishe, Ouri Wolfson+, Scott Graham, Wei Sun

High Performance Database Research Center
School of Computer Science
Florida International University
11200 SW 8TH ST, Miami, FL 33199 USA
Tel: (305) 348-1706, Fax: (305) 348-1705, E-mail: {rishen, grahams, weisun}@cs.fiu.edu

+Department of Electrical Engineering and Computer Science
University of Illinois at Chicago
851 S. Morgan St., Chicago, IL 60612, USA.
Tel: (312) 996-6770, Fax: (312) 413-0024, E-mail: wolfson@eecs.uid.edu

Abstract: *The main objective of a Query Distribution Algorithm is to find efficient ways to evaluate global queries. In traditional distributed database systems based on proprietary WAN infrastructures, efficiency of query processing is typically measured by query response time: the time it takes to finish a query. The monetary cost of network usage is not explicitly considered in the cost model since the overall dollar cost of maintaining the proprietary network is a fixed figure. This has placed the focus on how to reduce necessary data transfer across the network as much as possible for each query (and thus reduce transfer time), in order to maximize the effective use of the available network bandwidth. The adoption of intelligent networks in a distributed database system reveals two fundamental changes from a traditional one based on proprietary networks. First, the connections among the distributed sites are dynamic rather than static. In other words, the network links are established (through call setup) only when needed and the bandwidth can be allocated/de-allocated on-demand. Second, the dollar cost of the connections is charged on a per-connection basis. These new factors make the direct adoption of traditional distributed query processing and optimization techniques unsuitable as they do not consider network dollar cost (which can vary between any two sites) and query response time in an integral manner. This paper presents a Query Distribution Algorithm that takes these factors into consideration.*

KEYWORDS: Distributed Databases, Intelligent Networks, Query Distribution

INTRODUCTION

The problem of query processing and optimization in a distributed database environment has attracted much attention from the database research community, and numerous publications have been devoted to this subject. The most expensive database operation is the join operation. Considerable research effort has been applied to minimize the cost of join queries. A common technique to reduce the communication cost in a distributed join is to use semi-join. The

* This research was supported in part by NASA (under grants NAG5-9478, NAGW-4080, NAG5-5095, NAS5-97222, and NAG5-6830), NSF (CDA-9711582, IRI-9409661, HRD-9707076, and ANI-9876409), and ONR (N00014-99-1-0952).

semijoin strategy was justified by the assumption that network transmission cost is the dominant component in overall query processing cost.

There have been many theoretical studies in database integration. [1], [2], [3], and a number of prototype systems such as Information Manifold [4], Multibase [5], MRDSM [6], Pegasus [7], Carnot [8], SIMS [9], TSIMMIS [10,11], and SEMHDB [13,14] each represent a different methodology. One common assumption in the previous research work is that the ‘‘communication cost rates are constant and equal to transmission rate” [12] and thus the monetary cost of network usage is usually not explicitly considered. This greatly simplifies the problem of distributed query optimization as there is only one optimization metric: time. This assumption and therefore the various techniques developed based on it do not apply to our proposed distributed database environment using intelligent networks. First, it is clear that constant inter-node bandwidth cannot be assumed in networks made up of, for example, ISDN links, where a varying number of ISDN links may be established between two nodes. Second, with ISDN being charged based on usage, we now have two objectives for minimization: query response time and ISDN cost (in real dollars). Our proposed Query Distribution Algorithm (QDA) aims to close this gap and advance the query processing and optimization technologies to the new horizon of intelligent networks.

1. SYSTEM OVERVIEW

Our research is based on a heterogeneous distributed database system interconnected via an ISDN network. The techniques are applicable to other intelligent network topologies. We adopt the concentrated management mode. AMIP, the Application Management Interface Program, is the concentrated manager server of whole system. LIP, the Local Interface Program, is the client program offering a user interface. The heterogeneous DBMS at each local site communicates with AMIP and LIP through ODBC. Figure 1.1 depicts the system’s physical architecture. Further details on both AMIP and LIP can be found in [15].

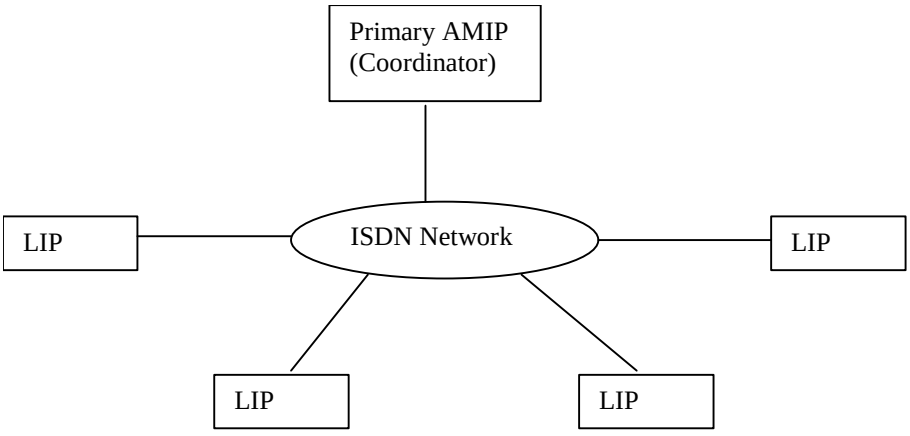

Fig 1.1 - Physical Architecture of QDA

The user queries the global schema using standard SQL statements. To reduce communication cost, the SQL parser module resides in AMIP. LIP only sends the user’s input query to AMIP. It is up to AMIP to parse the queries based on SQL syntax, generating a query tree. If the syntax generates a parser error, AMIP will notify LIP to ask the user to check the query syntax and reenter the query. AMIP then produces an

execution plan for the query, based on the global schema. This procedure is the focus of this paper. AMIP then executes the EP, transferring the resulting table to a dedicated LIP site if necessary and notifies the originating LIP to display it. According to the execution plan, the coordinator module will control LIPs to execute sub-queries and synchronize the data transfer from LIP to LIP. This query processing model is depicted in Figure 1.2.

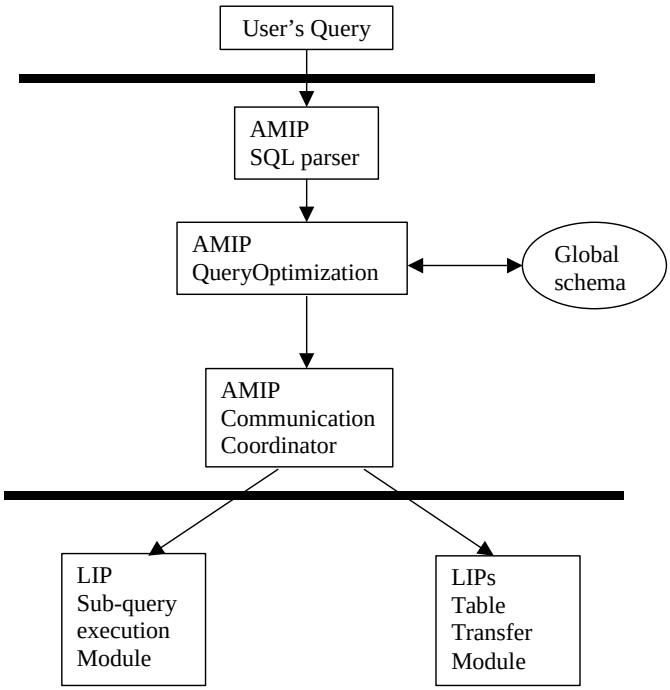

Fig1.2 - Query processing

2. QUERY EXECUTION PLAN

A query execution plan is a sequence of operations that retrieves data from databases and composes the answer set for a query. In our distributed database system, which provides access to heterogeneous data sources, an execution plan is a sequence of mixed sub-query executions and table transfer operations. A sub-query is a query that is to be executed against a certain data source at a certain LIP site; a table transfer operation sends a table from one LIP site to another. The goal of the Query Distribution Algorithm is to construct a cost-effective execution plan for a distributed query. A widely adopted rule of thumb when constructing an execution plan is to perform selections and projections as early as possible. This is usually referred to as the "push-down" strategy for selections and projections. Reflected in our system, we should perform the query's selections and projections present for the involved tables at their local sites before sending them over the network for more processing. The rationale behind this is to reduce the size of the tables in order to save communication cost.

We now look into the issues pertinent to finding a cost-effective execution plan for a query. We divide the discussion into two types of queries: (1) *selection queries* - which access only one table, and (2) *join queries* - which combine tuples from more than one table. For a selection query, the selection must be performed at the local site first (according to the push down rule). The only communication cost incurred is for sending the restricted table to the destination site where the result of the query is to be placed. To minimize the cost, we need to find the least-cost ISDN call path to transfer the data. We maintain an up-to-date graph of the network topology

and potential network topology with costs for each link. The problem of finding the least-cost path can be formulated as the classic "Shortest Path Problem" for which an efficient algorithm is well known. In addition to minimizing communication cost for selection queries, the shortest path algorithm is also useful in constructing execution plans of low communication costs for join queries, which will be discussed next.

The execution plan for a distributed join query comprises a sequence of two-way (2-way) joins, each of which joins two tables from different sites. Several decisions need to be made to select appropriate strategies while constructing the execution plan.

- *join method*: Two join methods, pure join and semi-join, are widely adopted in traditional distributed database systems. A pure join between tables *A* and *B* is performed by sending one of the tables (subject to restriction by selections and projections) to the other site for a regular join. We use *A* ||X| *B* to denote a pure join where *B* is the table being sent. Notation *A* |X|| *B* can be understood similarly. A semi-join, denoted *A* |X *B*, performs the following steps sequentially: (1) send *A*'s join column(s) to *B*, (2) restrict *B* by selecting only those tuples whose join attribute values have a match in the join columns received from *A*, (3) send the restricted table of *B* back to *A*'s site and perform a regular join with *A*. Semi-join *A* |x *B* requires less data transferred than a pure join when the size of the projected join column(s) of *A* consists of only a small portion of the total data transferred and table *B* is highly restricted by *A*'s join column(s).
- *join order*: When joining two tables (either by pure join or semi-join), there are two directions to perform the join (*A* ||X| *B* or *A* |X|| *B*, *A* |X *B* or *A* X| *B*). When joining more than two tables, the order of performing the individual 2-way joins also has an impact on the final communication cost. For example, to perform a three-way join (*A* join *B* join *C*), we can do either ((*A* join *B*) join *C*) or (*A* join (*B* join *C*)), with each two-way join subject to the options of pure join or semi-join.
- *data transfer paths*: There could be multiple paths to transfer a table from one site to another. The least expensive one must be sought in order to save communication cost. Note that the least cost path between two sites need not to be the direct connection (namely the edge connecting the two nodes). It could be an indirect path which travels through other node(s) to relay the data (namely, a path that consists of more than one edge). For example, if a semi-join *A* |x *B* is to be performed, the least cost paths to send data from *A* to *B* and from *B* to *A* must be found. And the paths can go through a third site *C*.
- *assembly sites*: Assembly sites are the places where the intermediate or final query results are assembled. Some algorithms only allow one assembly site, i.e., the destination site, and defer the assembly until the last stage. In those strategies, no intermediate join results, except for semi-join restricted tables, are allowed to be generated. Some allow more flexible choices on the sites to place intermediate join tables.

Given all the options in constructing execution plans, the number of possible execution plans for a join query increases exponentially with the number of involved tables. Thus, it is cost-prohibitive to perform an enumerated search on the entire space of execution plans to find the optimal one. In the next section, we describe an algorithm that finds an efficient execution plan from a selected subset of the search space.

3. QUERY DISTRIBUTION ALGORITHM (QDA)

3.1 Description

QDA is the kernel part of the AMIP. As shown in Figure 3.1, it consists of two parts: pre-order traverse and post-order traverse. The former receives the SQL objects (a data structure of query trees) produced by the AMIP's SQL parser, which processes SQL syntax, checks it, and produces SQL objects. Then pre-order traverse processes the query tree, makes a global semantic check of each table, does some preparation work such as assigning each source table a global exclusive internal table name and processing horizontally fragmented tables. We call this procedure "pre-order traverse query tree" or simply "pre-order traverse" because it traverses the query tree from root to leaves. The second step is post-order traverse, which accepts the result of pre-order traverse as input: a modified query tree and a global table schema object. Post-order traverse generates a distributed query Execution Plan (EP) and executes it by sending the command messages to LIP one by one to control and complete the query process. We call this procedure "post-order traverse query tree" or simply "post-order traverse" because it traverses the query tree from leaves to root.

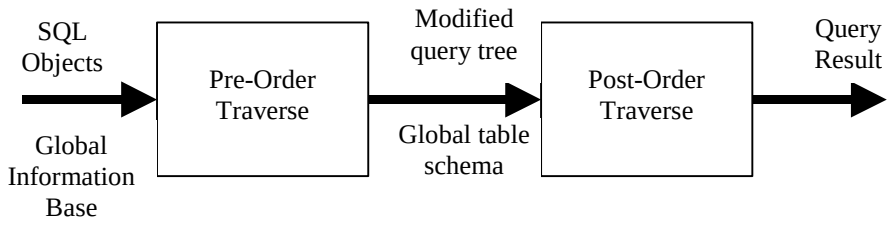

Fig 3.1 - QDA Procedure

3.2 QDA Algorithm

The QDA is the central module of the distributed system. The main task of QDA is processing complicated join queries. QDA decomposes chain queries into sub-queries according to least weight cost of communication cost and response time. Join decomposition adopts LPT (Linear Processing Tree) or Bushy Tree. The search space complexity of the former is O(n^3) and the latter is O(n^5) (where n is number of tables in the join). The dynamic channel allocation of ISDN is also considered in QDA as an important parameter in the shortest path search. To compress the search space and decrease the complexity of QDA processing, we adopt LPT search, and many heuristic rules are also introduced. The dynamic channel allocation of ISDN is considered as an important parameter in the shortest path search used by QDA. Last, but not least, the optimizing object function is decided by a weighted total of communication cost and response time. This weighted cost offers users a simple way to optimize the query processing to the user's preference.

When AMIP receives a query statement from LIP, it first calls upon its SQL parser to analyze the query statement and generate a query tree. The QDA process at the AMIP module performs three steps:

A. Pre-Order traverse the query tree and perform four tasks:
    1. For each node in the query tree, generate the corresponding result table schema and insert it into **EPGblSchema.** (**EPGblSchema** records the schema of all tables included in a SQL query, depending on a special thread.)

2. For each **SELECT** node of the query tree, perform a semantic check on the tables and their related attributes according to the information stored in the knowledge base.
3. For each **SELECT** node of the query tree, if there is an **OR** operator in the **WHERE** list, convert **OR** to **UNION ALL** and modify the query tree accordingly.
4. For each table in the **FROM** list of a **SELECT** node, generate the corresponding table and insert its schema into **EPGblSchema**.

B. Post-Order traverse the query tree and perform nine tasks:
1. For each **UNION**, **UNION ALL**, and **BRACKET** node of the query tree, generate the result table according to the **EPGblSchema**.
2. For each local query of a **SELECT** node, process the query locally then transfer the results to the destination site.
3. For each distributed query of a **SELECT** node, shrink each table in the **FROM** list by executing another query that is relatively simpler and that is only related to the local table itself and the projections.
4. For the tables generated in step 3, produce new **SELECT** nodes with only **GlobalName** tables.
5. From the new **SELECT** nodes, generate a new **Join Graph**.
6. From the new **Join Graph**, call the **QDA** kernel algorithm and generate a new Execution Plan (EP).
7. For the new EP, call EPInterpreter to execute the EP and produce the resultant table on the destination site.
8. With the resultant table and the queries of the original **SELECT** node, make a new query with all operations at the local site.
9. Evaluate the new local query, generate the final result table, and insert the result table's schema into **EPGblSchema**.

C. After the Post-Order traverse, we can produce the final result table with the query tree and **EPGblSchema**. We then delete all temporary tables on each local site.

The following provides further detail on how an Execution Plan is generated from a Join Graph.

For optimization, a Shrink Table procedure is introduced to do some simple condition query and projection operations to minimize the size of tables to be joined. Meanwhile some accessory work is done for the upcoming processing of the distributed join query. With the pre-processed simplified internal table produced above, we can focus on processing the join operation (including Cartesian product) and producing a Join Graph, each node of which indicates the internal tables taking part in join operation and each edges of which indicate join relations between two tables.

The next step is to produce a linear tree that can be turned into an execution plan. We can get a great number of linear trees from a Join Graph. If we consider the Cartesian product as a join relation between two vertices of the Join Graph, the number of linear trees produced is C(n,2)*(n-2)!=n!/2. When n is not a small number, the search space is very large. To compress the search space, we use some heuristic rules, which are described below:

- Select the minimal size table as the first table in Linear Tree.
- The priority of join is greater than Cartesian product.
- When there exists more than one join relation table (Adjacent Vertex list has more than one element) or Cartesian product table, select the table that has minimal size.

From a Linear Tree we get to know the order of tables in a join sequence and the operations between two tables: join(|X|) or Cartesian product(X). Now we have to decide the join method

(semi-join or pure join) and the result table location in each join operation step. Since the search space is still very large (2^(n-1) if there are n source tables and we do not consider the join method), we must introduce some heuristic rules to optimize the whole search process. The first rule is called destination pruning, and the other is called of k-stage decision.

The destination pruning technique is a pre-processing step toward the production of an execution plan. The pruning technique is based on the fact that in each join step we can decide the result table location immediately if the location of one of the operand table is equivalent to the destination site. Another aspect is that we also do not have to search the intermediate result table location, since we always put the intermediate table on the destination site.

K-stage decision is introduced to offer users the option to limit the depth of the binary search tree according to practical conditions, such as computer's processing ability or a user's need for optimization accuracy, to compress the searching space from 2^(n-1) to 2^k. When variable parameter k (k<n) is set, the searching space is compressed to 2^k. At the end of the k^{th}-stage, we can get a partial optimal conclusion to decide the intermediate table location and join method of each join step in the stage. When k>=n-1 we scan the whole searching space and can acquire the optimal EP.

The introduction of the k-stage decision method compresses the search space greatly when n is not a small number. But under a multi-query concurrent environment the search space can still be excessive and very difficult to compute even if n is not too large. In practice we adopt the "k=1 local optimal" rule to achieve the trade-off. When processing one distributed query we have to predict the variation of the ISDN Network Topology Graph (NTG) to get the k-stage optimal cost. When there exist multiple concurrent distributed query requests, a simple solution is to set the current real NTG as the reference NTG and to not generate any predicted NTG to estimate local optimal cost. To acquire a k-stage optimal cost with concurrent query requests, we must adopt a dynamic prediction method to predict the real variations of the NTG in a multi-threaded environment.

4. CONCLUSION

We have presented the outline of a Query Distribution Algorithm that is optimized for databases distributed across intelligent networks. Traditional query distribution algorithms do not take the costs associated with setting up and maintaining network connections into account. Since these costs were typically fixed on a monthly or yearly fee schedule, instead of a per-minute or per-second schedule, they did not need to. With today's intelligent networks, these issues do become factors in any optimization method for query distribution. Our Query Distribution Algorithm has been implemented in a prototype system and has proven to be effective, allowing optimizations to be made for both response time and communication cost. Future work along these lines includes optimizations for hybrid networks, and for varying Quality of Service levels.

REFERENCES

[1] B. E. Reddy, P. G. Reddy, and A. Gupta: "A methodology for integration of heterogeneous database." *IEEE Transactions on Knowledge and Data Engineering*, 6 (6): 920-933, December 1994.

[2] A. Sheth and V. Kashyap: “So far (schematically) yet so near (semantically).” *Interoperable Database Systems*, pages 283--312, Elsevier Science Publishers B. V. (North-Holland) 1993.
[3] A. Sheth and J. Larson: “Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Database.” *ACM Computer Surveys*, 22 (3):183-235, Sept. 1990.
[4] A. Levy, et al: “Querying Heterogeneous Information Sources Using Source Descriptions” In *Proceedings of the 22nd VLDB Conference*, 251-262, 1996, Bombay, India.
[5] W. Litwin and A. Abdellatif: “Multidatabase interoperability.” *IEEE Computer*, pages 10-18, December 1986
[6] W. Litwin and A. Abdellatif: “An overview of the multi-database manipulation language MDSL.” *Proceedings of the IEEE*, 75(5):621-632, May 1987.
[7] R. Ahmed et al: “The Pegasus heterogeneous multidatabase system.” *IEEE Computer*, 24 (12):19-27, December 1991
[8] C. Collet, M. N. Huhns, and W. Shen: “Resource integration using a large knowledge base in Carnot.” *IEEE Computer*, pages 55--62, December 1991.
[9] Y. Arens, C. Chee, C. Hsu, and C. Knoblock: “Retrieving and integrating data from multiple information sources.” *International Journal on Intelligent and Cooperative Information Systems*, 2(2), June 1993.
[10] H. Garcia-Molina et al: “The TSIMMIS approach to mediation: data models and languages” In *Next Generation Information Technologies and Systems*, June 1995. Nahria, Israel.
[11] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina: “Object fusion in mediator systems” In *Proceedings of the 22nd VLDB Conference*, 1996, Bombay, India.
[12] A. Segev: “Strategies for distributed query optimization,” Information Sciences, 54 (1-2), Mar. 1992.
[13] N. Rishe, et al: “Semantic Access: Semantic Interface for Querying Databases,” Proceedings of the 26th International Conference on Very Large Databases, 2000, pp. 591-594
[14] N. Rishe, et al: “SemWrap: A Semantic Wrapper over Relational Databases with Substantial Size Reduction of User’s SQL Queries,” EDBT 2000: Seventh International Conference on Extending Database Technology, pp. 13-14.
[15] N. Rishe, et al: “On Database Integration over Intelligent Networks,” Proceedings of the ISCA 2nd International Conference on Information Reuse and Integration (IRI-2000), pp. 70-75.