Distributed Query Optimization: Use of mobile Agents
Kodanda Kumar Melpadi
M.Tech (IT) GGS Indraprastha University Delhi
mk_kumar_76@yahoo.com

Abstract  DDBS adds to the conventional centralized DBS some other types of processing expenses, because of the additional design (hardware & software) to handle the distribution. These expenses present as the cost of data transfer over the network. Data transferred could be, intermediate files resulting from local sites, or final results need to be sent back to the original site that issued the query. Minimizing the quantity of data transferred is a desirable optimization criterion. The goal of DQP is to execute such queries as efficiently as possible in order to minimize the response time. The use of mobile agents in distributed query processing improves response time.

Keywords: Distributed Query Optimization, Distributed databases, Mobile agents

1 Introduction

Data retrieval from different sites in a DDB is known as distributed query processing (DQP). So a distributed query is one that selects data from databases located at multiple sites in a network and distributed processing performs computations on multiple CPUs to achieve a single result. Query processing is much more difficult in distributed environment than in centralized environment because a large number of parameters affect the performance of distributed queries, relations may be fragmented and/or replicated, and considering many sites to access, query response time may become very high. It is quite evident that the performance of a DDBS is critically dependant upon the ability of the query optimization algorithm to derive efficient query processing strategies. DDBMS query optimization algorithms attempts to reduce the quantity of data transferred. Minimizing the quantity of data transferred is a desirable optimization criterion. The distributed query optimization has several problems relate to the cost model, larger set of queries, optimization cost, and optimization interval.

2 Query Optimization Process

The most common queries are Select-Project-Join queries. In this paper, we will focus on the join-ordering problem since permutations of the join order have the most important effect on performance of relational queries. The query optimization process shown in Fig.2.1, consists of getting a query on n relations and generating the best Query Execution Plan (QEP). For a given query, the search space can be defined as the set of equivalent operator trees that can be produced using transformation rules. The example below illustrates 3 equivalent join trees in figure 2.2, which are obtained by exploiting the associative property of binary operators. Join tree (c) which starts with a Cartesian product may have a much higher cost than other join trees.

SELECT ENAME, RESP
FROM EMP, ASG, PROJ
WHERE EMP.ENO=ASG.ENO
AND ASG.PNO=PROJ.PNO

Regarding different search spaces, there would be different shape of the join tree. In a linear tree (figure 2.3(a)), at least one operand of each operand node is a base relation. However, a bushy tree (figure 2.3(b)) might have operators whose both operands are intermediate operators. In a distributed environment, bushy trees are useful in exhibiting parallelism. The query optimizer is usually seen as three components: a search space, a cost model, and a search strategy. The last layer in figure 2.1 is performed by all the sites having fragments involved in the query. Each sub-query executing at one site, called a local query, is then optimized using the local schema of the site. At this time, the algorithms to perform the relational operations may be chosen. There are three optimization

Input
Query

Search space
Transformation Rules

Search
Cost
Static query optimization: is performed at query compilation time and uses the database statistics to estimate the sizes of the intermediate relations.

Dynamic query optimization: proceeds at query execution time. For any execution, the choice of the best next operation can be supported accurate results of the operations executed previously. Therefore, database statistics are not needed to estimate the size of intermediate results.

Hybrid query optimization: is combined static and dynamic query optimization.

Three most common types of algorithms for join-ordering optimization are deterministic, Genetic and randomized algorithms. Deterministic algorithm, also known as exhaustive search dynamic programming algorithm, produces optimal left-deep processing trees with the big disadvantage of having an exponential running time. Due to the very large time and space complexity of this algorithm for plan enumeration, iterative dynamic programming approach was proposed which produces reasonable plans with reasonable running time for most network topologies. However, its complexity is not much more than classical DP algorithm. Genetic and randomized algorithms on the other hand do not generally produce an optimal access plan. But in exchange they are superior to dynamic programming in terms of running time. Experiments have shown that it is possible to reach very similar results with both genetic and randomized algorithms depending on the chosen parameters. Still, the genetic algorithm has in some cases proved to be slightly superior to randomized algorithms. Layers of distributed query optimization have been depicted in Fig.2.4.
the total time, or the response time. The total time is the sum of all times and the response time is the elapsed time from the initiation to the completion of the query. The total time (TT) is computed as below, where TCPU is the time of a CPU instruction, T/I/O the time of a disk I/O, TMSG the fixed time of initiating and receiving a message, and TTR the time it takes to transmit a data unit from one site to another.

\[
TT = TCPU \ast \#\text{insts} + T/I/O \ast \#\text{I/Os} + TMSG \ast \#\text{msgs} + TTR \ast \#\text{bytes}
\]

This response time (RT) is calculated as below:

\[
RT = TCPU \ast \text{seq}_{\#}\text{insts} + T/I/O \ast \text{seq}_{\#}\text{I/Os} + TMSG \ast \text{seq}_{\#}\text{msgs} + TTR \ast \text{seq}_{\#}\text{bytes}
\]

Most early distributed DBMSs designed for wide area networks have ignored the local processing cost and concentrate on minimizing the communication cost.

**Database Statistics:** The main factor affecting the performance is the size of the intermediate relations that are produced during the execution. When a subsequent operation is located at a different site, the intermediate relation must be transmitted over the network. It is of prime interest to estimate the size of the intermediate results in order to minimize the size of data transfers.

**Join:** The upper bound of the join cardinality is the cardinality of the Cartesian product.

Two main approaches exist to order joins in fragment queries:

1) Direct optimization of the ordering of joins (e.g. in the Distributed INGRES algorithm).
2) Replacement of joins by combination of semi-joins in order to minimize communication costs.

Let R and S are relations stored at different sites and query \( R \bowtie S \) be the join operator. The obvious choice is to send the smaller relation to the site of the larger one.

If \( \text{size}(R) < \text{size}(S) \)

If \( \text{size}(S) < \text{size}(R) \)

The objective of the join ordering algorithm is to transmit smaller operands. Since the join operations may reduce or increase the size of intermediate results, estimating the size of joint results is mandatory, but difficult. Several researchers proposed mobile agent contributions to the distributed dynamic query optimization. F. Morvan studied the contribution of the execution model based on mobile agents to the distributed dynamic query optimization. They proposed and evaluated three methods allowing cooperation between the agents participating in the query evaluation process. S. Das presented a mobile agent based engine, Agent-based Complex QUerying and Information Retrieval Engine (ACQUIRE), for retrieving data from heterogeneous, distributed data sources. ACQUIRE acts as an interface agent by accepting a high-level user query and decomposing this query into a series of subqueries. Mariposa system proposed a large scale distributed and static query optimization.

4. Proposed Query Optimization Algorithm

Input: MRQ: multi-relation query
Output: result of the optimized multi-relation query

**Begin**

for each detachable OVQi in MRQ do

run (OVQi) ------------------- (1)
end-for

for each database location sites (s-1) do

\{ s is the number of databases located at different sites \}

Construct all possible query plans [QPs] - (2)
end-for

while n 0 do \( n \) is the number of mono-relation queries) --------(3)

begin

for each pair (MQ, S) in Mono-relation query-site list by using mobile agent do

Move mono-relation query MQ to site S --(3.1)

Get size of intermediate relation [SIR] -- (3.2)
end-for

\( n \leftarrow n -1 \)
end-while

for each [QPs] do

compute the estimate cost of all monorelation queries using [SIR] ------ (4)
end-for

choose the best execution plan (BEP)----(5)
run (BEP) ----------------------------- (6)

End. \{QOA\}

In our proposed query optimization algorithm, firstly detaches a given multi-relation query (MRQ) to obtain one variable query (OVQ) called mono-relation queries (MQ) (step 1). Then construct all possible query plans (step 2). In (step
3.1), mono-relation query is moved to corresponding site S by using mobile agent. Get size of intermediate relation (SIR) in (step 3.2). Compute the estimate cost of all mono-relation queries using SIR (step 4). In (step 5), chooses the best query execution plan. (Step 6) executes that optimized plan.

5. Case Study

Consider the expression of the query in relational calculus using the SQL syntax is:

```sql
SELECT ENAME
FROM EMP, ASG, PROJ
WHERE EMP.ENO=ASG.ENO
AND ASG.PNO=PROJ.PNO
AND PNAME= 'CAD/CAM';
```

Assumption: There are no fragmented databases and each database locates at different sites. Figure 5 shows the join graph of distributed query. According to the proposed algorithm, given multirelation query is detached into three monorelation queries. They are as follows:

- The relational algebra query for mono-relation query1: \( \Pi_{ENO, ENAME}(EMP) \)
- The relational algebra query for mono-relation query2: \( \Pi_{PNO, ENO}(ASG) \)
- The relational algebra query for mono-relation query3: \( \sigma_{PNAME= "CAD/CAM"}(PROJ) \)

Then construct all possible query execution plans. Four possible execution plans obtained. They are

![Figure 5 Join Graph of Distributed Query](image)

**Plan 1:** The relational algebra query for plan 1 is:

\[ \Pi_{ENAME}(EMP \bowtie_{ENO} (ASG \bowtie_{PNO} PROJ)) \]

Where \( PROJ'= \sigma_{PNAME= "CAD/CAM"}(PROJ) \)

Step 1: Site3 \( \rightarrow \) Site2
Site 3 computes \( PROJ'= \sigma_{PNAME= "CAD/CAM"}(PROJ) \)

Step 2: Site2 \( \rightarrow \) Site1
In step1, mobile agent sends PROJ’ to site2.
Site 2 computes \( ASG' = ASG \bowtie_{PNO} PROJ' \)

Step 2: \( ASG' \rightarrow Site1 \)
In step2, mobile agent carries the intermediate relation ASG’ to site1.
Site1 computes \( ASG' \bowtie EMP \)

**Plan 2:** The relational algebra query for plan 2 is:

\[ \Pi_{ENAME}(EMP \bowtie_{ENO} (ASG \bowtie_{PNO} PROJ')) \]

Where \( PROJ'= \sigma_{PNAME= "CAD/CAM"}(PROJ) \)

Step 1: Site3 \( \rightarrow \) Site2
In step1, mobile agent is used to send relation ASG to site3.
Site3 computes \( ASG'=ASG \bowtie PROJ' \)

Step 2: \( ASG' \rightarrow Site1 \)
In step2, mobile agent carries intermediate relation ASG’ to site1.
Site1 computes \( ASG' \bowtie EMP \)

**Plan 3:** The relational algebra query for plan 3 is:

\[ \Pi_{ENAME}(PROJ' \bowtie_{PNO} (EMP \bowtie_{ENO} ASG)) \]

Where \( PROJ'= \sigma_{PNAME= "CAD/CAM"}(PROJ) \)

Step 1: Site2 \( \rightarrow \) Site3
In step1, mobile agent carries relation EMP to site2.
Site2 computes \( EMP'=EMP \bowtie ASG \)

Step 2: \( EMP' \rightarrow Site3 \)
In step2, mobile agent is used to send the intermediate relation EMP’ to site3.
Site3 computes \( EMP' \bowtie PROJ' \)

**Plan 4:** The relational algebra query for plan 4 is:

\[ \Pi_{ENAME}(EMP \bowtie_{ENO} (PROJ' \bowtie_{PNO} ASG)) \]

Where \( PROJ'= \sigma_{PNAME= "CAD/CAM"}(PROJ) \)

Step 1: Site2 \( \rightarrow \) Site3
In step1, mobile agent sends the mono-relation query PROJ’ to site2.
Step 2: \( EMP' \rightarrow Site2 \)
In step2, another mobile agent carries relation EMP to site2.
Site2 computes \( PROJ' \bowtie ASG \bowtie EMP \)

To select one optimized plan from these possible strategies, sizes of relations and intermediate relations are estimated using mobile agents. Assume the sizes of relations as size (EMP) =8, size (ASG) =10 and size (PROJ) =4 and size (PROJ’) =1. Strategy 4 is the least total
6. **Proposed Mobile Agent Co-operation in Distributed Query Optimization**

In order to measure the performance of mobile agent cooperation in query optimization, the prototype system is built. Figure 6 shows Overall Design of Proposed Query Optimization Model. In our proposed model, we use hybrid query optimization. We use database statistic information and results of mono-relation query. This prototype system contains three main modules: Query Decomposition, Optimization Plan and Execution Plan. In Query Decomposition module, the input query is decomposed into mono-relation query using detachment and substitution techniques. Optimization Plan step uses all mono-relation queries as input and constructs all possible query plans. Then compute the estimate cost of each query execution plan. In order to estimate cost of query plan, mobile agent is used to collect the statistical database information of distributed databases. Each mobile agent carries mono-relation query and is sent to corresponding databases located at different sites. Agents assign the monorelation query to local database which execute the input query and returns result to agent. Agent then sends back the size of immediate result of mono-relation query. When all the necessary information have been received, estimate cost of all mono-relation queries are computed and optimizer chooses the best execution plan. Execution Plan accepts the best execution plan and decides whether the input optimized plan can be performed in serial or parallel plan. Executes optimized plan using mobile agent and sends results back to client.

7. **Conclusions & Future Scope**

We discuss the distributed query processing steps, agent-based query optimization, and our proposed system architecture. The most vital part of distributed query system is decomposition of the input query into multiple fragment queries to retrieve the information at different sites according to data location. Another important step to consider is how to execute the algebraic operations such as JOIN and send the partial result data to accumulate final data and return to the user. In our proposed system, we use hybrid query optimization to get the statistical database information of distributed databases, carry mono-relation query to corresponding site and compute all possible query execution plans by cooperating mobile agents. We construct our new algorithm that make use of mobile agent to accumulate the data fragment information to get the optimize query method.

8. **References**


