Query Optimization Concepts in Distributed Database

Rudrani Sharma, Er. Pushpneel Verma, Er. Sachin Chaudhary

Department Of Computer Science & Engineering
Bhagwant Institute of Technology, Muzaffarnagar

Abstract
Query optimization is an important part of database management system. In this paper, through the research on query optimization technology, based on a number of optimization algorithms commonly used in distributed query, it aims to arrive at an optimal query processing plan for a given distributed query. As per the approach, the query plans having the required data residing close to each other are considered more efficient and, therefore, these generated query plans would result in efficient query processing.

Keywords: Database, query processing, distributed query strategy, system model, query processing cost, cost measures.

Introduction
In recent years, with the development of computer network and database technology, distributed database is more and more widely used; with the expanding application, data queries are increasingly complex, the efficiency requests are increasingly high, so query processing is a key issue of the distributed database system. In a distributed database environment, data stored at different sites connected through network. A distributed database management systems (DDBMS) support creation and maintenance of distributed database. The research literature proposes a wide variety of query optimization algorithms. Yu/chang give comprehensive overviews on various query optimization techniques for distribute database management system [20]. However, these overviews do not attempt to develop a model of query optimization that explains and presents the algorithms in a uniform way. This understanding in case we want to change or extend existing algorithms to adapt them to new requirements. In this research we consider query processing algorithms for a Distributed Database system. There has been many research done on distributed query processing methods (see [2],[3]). Increased reliability and performance can also be attained with a distributed database.

All database systems must be able to respond to requests for information from the user—i.e. process queries. How a DBMS processes queries and the methods it uses to optimize their performance are topics that will be covered in this paper. In certain sections of this paper, various concepts will be illustrated with an example. Since many optimization algorithms differ in their computational behavior while reflecting aspects of the implementation environment at the same time, it is the purpose of this paper to understand all of them by few simple concepts. Finally, we summaries our findings and discuss future work.

General aspects of optimization
To provide a better understanding of what we mean by the term query, query processing and query optimization. Further we discuss the algorithms of query optimization that can found in all optimization algorithms described in the papers.

Definitions And Examples
A. What Is A Query?
A database query is the instructing a DBMS to update or retrieve specific data to/from the physically stored medium. The actual updating and retrieval of data is performed through various “low-level” operations. Examples of such operations for a relational DBMS can be relational algebra operations such as project, join, select,Cartesian product, etc.

B. The Query Processor
There are three phases that a query passes through during the DBMS’ processing of that query:
1. Parsing and translation
2. Optimization
3. Evaluation

Most queries submitted to a DBMS are in a high-level language such as SQL. During the parsing and translation stage, the human readable form of the query is translated into forms usable by the DBMS. These can be in the forms of a relational algebra expression, query tree and query graph. Consider the following SQL query:

```
SELECT make
FROM vehicles
WHERE make = "Toyota".
```

Query Graph After parsing and translation into a relational algebra expression, the query is then transformed into a form, usually a query tree or graph that can be handled by the optimization engine. The optimization engine then performs various analyses on the query data, generating a number of valid evaluation plans. From there, it determines the most appropriate evaluation plan to execute. After the evaluation plan has been selected, it is passed into the DMBS’ query-execution engine (also referred to as the runtime database processor), where the plan is executed and the results are returned.

B.1 - Parsing and Translating the Query

The first step in processing a query submitted to a DBMS is to convert the query into a form usable by the query processing engine. High-level query languages such as SQL represent a query as a string, or sequence, of characters. Certain sequences of characters represent various types of tokens such as keywords, operators, operands, literal strings, etc. Like all languages, there are rules (syntax and grammar) that govern how the tokens can be combined into understandable (i.e. valid) statements. The primary job of the parser is to extract the tokens from the raw string of characters and translate them into the corresponding internal data elements (i.e. relational algebra operations and operands) and structures (i.e. query tree, query graph). The last job of the parser is to verify the validity and syntax of the original query string.

B.2 - Optimizing the Query

In this stage, the query processor applies rules to the internal data structures of the query to transform these structures into equivalent, but more efficient representations. The rules can be based upon mathematical models of the relational algebra expression and tree (heuristics), upon cost estimates of different algorithms applied to operations or upon the semantics within the query and the relations it involves. Selecting the proper rules to apply, when to apply them and how they are applied is the function of the query optimization engine.

B.3 - Evaluating the Query

The final step in processing a query is the evaluation phase. The best evaluation plan candidate generated by the optimization engine is selected and then executed. (Note that there can exist multiple methods of executing a query. Besides processing a query in a simple sequential manner, some of a query’s individual operations can be processed in parallel— either as independent processes or as interdependent pipelines of processes or threads. Regardless of the method chosen, the actual results should be same.)

C. Query Processing

Query processing is defined as the activities involved in parsing, validating, optimizing and executing a query. The main aim of query processing is Transform query written in high-level language (e.g. SQL), into correct and efficient execution strategy expressed in low-level language (implementing Relational Algebra) and to find information in one or more databases and deliver it to the user quickly and efficiently.

D. Query Optimization

Query optimization is defined as the activity of choosing an efficient execution strategy for processing a query. Query optimization is a part of query processing. The main aims of query optimization are to choose a
transformation that minimizes resource usage, Reduce total execution time of query and also reduce response time of query. Distributed Query Processing Methodology:

It contains four stages which are as follows:

**Query decomposition**
- Normalization
  - manipulate query quantifiers and qualification
- Analysis
  - detect and reject “incorrect” queries
  - possible for only a subset of relational calculus
- Simplification
  - eliminate redundant predicates
- Restructuring
  - calculus query $\rightarrow$ algebraic query
  - more than one translation is possible
  - use transformation rules

**Data localization**
Input: Algebraic query on distributed relations
- Determine which fragments are involved
- Localization program
  - substitute for each global query its materialization program
- optimize

**Global query optimization**
Fragment query
- Find the best (not necessarily optimal) global schedule
- Minimize a cost function
- Distributed join processing
> Bushy vs. linear trees
> Which relation to ship where?
> Ship-whole vs ship-as-needed
> Decide on the use of semijoins
> Semijoin saves on communication at the expense of more local processing.
> Join methods
> nested loop vs ordered joins (merge join or hash join)

**Data localization**
Input: Algebraic query on distributed relations
- Determine which fragments are involved
- Localization program
- substitute for each global query its materialization program
- optimize

**system model**
Three types of models
Physical models: capture the hardware composition of a system in terms of computers and other devices and their interconnecting network;
Architecture models: define the main components of the system, what their roles are and how they interact (software 2 system, what their roles are and how they interact (software architecture), and how they are deployed in a underlying network of computers (system architecture);
Fundamental models: formal description of the properties that are common to architecture models. Three fundamental models: – interaction models, failure models and security models

**Cost based optimization**
Main Consideration for Query Optimization
- Communication cost
- If there is several copies of a relation, decide which copy to use
- Amount of data being shipped
- Relative processing speed at each site
- Site selection

Global plan includes
- several local plans (subqueries)
- If response time is critical, subqueries can be carried out in parallel
- Local plans constructed by optimizer of each site

**Cost measures**
Cost is generally measured as total elapsed time for answering query
- Many factors contribute to time cost
  - disk accesses, CPU, or even network communication
- Typically disk access is the predominant cost, and is also relatively easy to estimate. Measured by taking into account
  - Number of seeks * average-seek-cost
  - Number of blocks read * average-block-read-cost
  - Number of blocks written * average-block-write-cost
  - Cost to write a block is greater than cost to read a block
  - data is read back after being written to ensure that the write was successful
- Assumption: single disk
  - Can modify formulae for multiple disks/RAID arrays
Or just use single-disk formulae, but interpret them as measuring resource consumption instead of time

For simplicity we just use the number of block transfers from disk and the number of seeks as the cost measures

\[
\begin{align*}
\text{Cost} & = b \times t_r + S \times t_s \\
& \text{where } b \text{ is the number of block transfers, } S \text{ is the number of seeks, } t_r \text{ is the time to transfer one block, and } t_s \text{ is the time for one seek.}
\end{align*}
\]

We ignore CPU costs for simplicity. Real systems do take CPU cost into account.

We do not include the cost to writing output to disk in our cost formulae.

Several algorithms can reduce disk IO by using extra buffer space.

Amount of real memory available to buffer depends on other concurrent queries and OS processes, known only during execution.

We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available.

Required data may be buffer resident already, avoiding disk I/O.

But hard to take into account for cost estimation.

**Conclusion**

Distributed query optimization is more complex that centralized query processing, since (i) bushy query trees are not necessarily a bad choice, (ii) one needs to decide what, where, and how to ship the relations between the sites. Query optimization searches the optimal query plan (tree) For N relations, there are O(N!) equivalent join trees. To cope with the complexity heuristics and/or restricted types of trees are considered. There are two main strategies in query optimization: randomized and deterministic. (Few) semi-joins can used to implement a join. The semi-joins requires more operations to perform, however reduces the data transfer rate. INGRES, System R, and Hill Climbing algorithms are used to optimize queries.

**References**

[23] LA Fortune, Wong, “Query Optimization”.