International Journal of Enhanced Research in Management & Computer Applications, ISSN: 2319-7471
Vol. 2 Issue 6, June-2013, pp: (1-4), Available online at: www.erpublications.com

Selectivity & Cost Estimates in Query Optimization in Distributed Databases

Ridhi Kapoor[1], Dr. R. S.Virk[2]
Department of Computer Science & Engineering, Guru Nanak Dev University Amritsar, Punjab, India

Abstract: Query optimizers are critical to the efficiency of modern relational database systems. If a query optimizer chooses a poor query execution plan, the performance of the database system in answering the query can be very poor. This study describes that there are numerous alternative ways to execute a query. These are so called execution plans. A component in the database management system called the Query Optimizer decides how to pick an efficient execution plan. For this the optimizer deploys cost-based optimization. Approximate execution costs are calculated for various plans, and one with low cost is chosen. The execution cost is a weighted function of the system resources needed to execute the query. Examples of such system resources are the CPU time or the number of I/O operations. In order to come up with reasonable cost estimates, the optimizer needs to estimate the size of sub-queries. This is important, for instance, when choosing the join order of the relations. To estimate the sizes of sub-queries, the optimizer needs to know the selectivity of the query predicates.

Keywords: Query Optimization, Distributed Databases, Cost Based Query Optimizers, Selectivity, Response Time, Total Time.

1. Introduction

A DDB query is answered by joining tables. In a distributed database, tables reside on different nodes of a computer network; to join tables, data must be moved between nodes. Consequently, the cost of a distributed query includes a processing cost (the joins) and a transmission cost [1]. In a query, the order of joins is not specified. Distributed query optimization involves finding an efficient order for the required joins. Before moving a large table across the network, it may be possible to reduce its size by restricting it to just those rows that are related to the table to which it will be joined. However, to effect this reduction, another table must be sent across the network and an additional join performed. As the number of tables in a query increases, the number of possible *join* schedules grows at least exponentially; an exhaustive search for the minimum cost schedule is not feasible. Optimizer needs, the selectivity of a query, i.e., the number of records that qualifies to a query, in order to generate an efficient query execution plan. The query optimizer can generate several execution plans for the same query. To choose the execution plan having the response time close to the optimal, the optimizer is based on a cost model.

Distributed Database [2]**:** A database that consists of two or more data files located at different sites on a computer network.

Query consists of operations on tables. Most commonly performed operations are Select (σ): Returns tuples that satisfy a given predicate Project (π): Returns attributes listed Join (⋈): Returns a filtered cross product of its arguments Set operations: Union, Intersect, and Difference.

Query Processing: It is defined as the activities involved in parsing, validating, optimizing and executing a query. The main aim of query processing is to transform a query written in high level language(eg.SQL) into efficient and correct strategy expressed in low level language(implementing low-level language).

High level user query -> Query Processor ->low-level data manipulation commands.

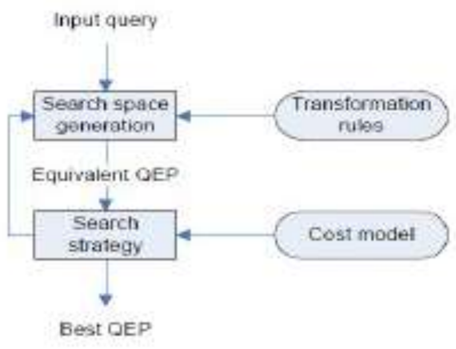

Fig. 1: Cost Based Estimation

Query Optimization: It refers to the process by which the best execution strategy for a given query is found from a set of alternatives. 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.
Cost Based Query Optimization: It assigns an estimated "cost" to each possible query plan, and chooses the plan with the smallest cost [1]. Costs are used to estimate the runtime cost of evaluating the query, in terms of the number of I/O operations and CPU requirements, and other factors determined from the dictionary. The set of query plans examined is formed by examining the possible access paths (e.g. index scan, sequential scan) and join algorithms (e.g. sort-merge join, hash join, nested loop join). The search space can become quite large depending on the complexity of the SQL query.

II. DISTRIBUTED COST MODEL

One of the hardest problems in query optimization is to accurately estimate the costs of alternative query plans. Optimizers cost query plans using a mathematical model of query execution costs that relies heavily on estimates of the cardinality, or number of tuples, flowing through each edge in a query plan[3]. Cardinality estimation in turn depends on estimates of the Selection factor of predicates in the query. An optimizer cost model includes cost functions to predict the cost of operators, and formulas to evaluate the sizes of results.
Cost functions can be expressed with respect to either the Total Time, or the Response Time [7].

- **Total Time** is the sum of all times.

Where **TCPU** is the time of a CPU instruction, **TI/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 * #insts +TI/O * #I/Os +TMSG * #msgs +TTR * #bytes.

- **Response Time** is the elapsed time from the initiation to the completion of the query.

RT=TCPU*seq_#insts+TI/O*seq_#I/Os+TMSG*seq_#msgs +TTR*seq_#bytes.
where **#seq x** (x in instructions, I/O, messages, bytes) is the **maximum number** of x which must be done sequentially.

III. SELECTIVITY

The selectivity factor is defined as the ratio of output to input tuples i.e.
N (output)/N (input)
Here we consider selectivity factors for restrictions, projections, joins; semi-joins and anti-joins. Since the output of each such operator is a new relation, we indicate the selectivity factor of an operator by the name of the relation(s) on which it is applied.

fR - Selectivity factor for restriction f on R
R - Selectivity factor for projection, including duplicates' removal, over R
J(R,S) - Selectivity factor for the classic join (as opposed to anti-join) on R, S.

Query Predicate: - Given a relation R, a query predicate π on R is a Boolean function

$$\Pi: \underset{\rightarrow}{R} \rightarrow \{true, false\}$$

Thus, for each tuples in the relation the predicate is either true or false. We also denote the set of all tuples that satisfy the predicate as π(R):

$$\Pi(R) = \{t€R\ (t) = true\}$$

Predicate Cardinality and Selectivity: Let R be a relation and π is a predicate on R. The cardinality of π is the number of tuples from R that satisfy π:

$$card\ (\pi) = |\pi(R)| = |\{t|t€R \wedge \pi(t) = true\}|.$$

The selectivity of π is its cardinality divided by the number of tuples in R:

$$sel\ (\pi) = card\ (\pi)/|R|$$

The cardinality is the number of tuples satisfying the condition. The selectivity is the portion of tuples satisfying the condition.
The selectivity of the predicate is the portion of the tuples from the base relation which satisfy the given predicate. Now consider a predicate π and the random variable Xπ, defined as follows:

$$X\pi = \{1;\ \pi\ (t) = true \\ 0;\ otherwise\}$$

The probability that a randomly selected tuples satisfies a predicate π equals the selectivity of π.
P (Xπ = 1) = sel (π)

Types of Selectivity

Index selection: The optimizer has to choose between alternative plans which use different indexes. In addition, skipping indexes and scanning the whole data set can be an option too.

Two-table join: When joining two tables, it is often best to read the smaller table into the main memory and leave the bigger table to the disk. The assumption is that the larger table does not fit into the main memory completely, and we want to read it from the disk only once.

Multi-table joins: When joining more than two tables, the order of the join affects the cost. In order to choose the optimal join order, we have to estimate the selectivity of predicates which filter the tables.

IV. QUERY EXECUTION COST

Query execution cost is a function of the resources used for the execution. An actual cost model is defined by the resources considered and the cost function, i.e. how these resources are weighted in the final cost calculation.

Cost Components for Query Execution

- **Access Cost:** Cost of searching for, reading, writing data blocks that resides on secondary storage disk.
- **Storage Cost:** Cost of storing intermediate files that are generated by execution strategy for the query.
- **Computation Cost:** Cost of performing in-memory operations on the data buffers during query execution. It includes searching for, sorting, merging, records, computing field values.
- **Communication Cost:** Cost of shipping the query and its results from database site to the site or terminal where it originated.
- **Memory Usage Cost:** Cost pertaining to number of memory buffers needed during query execution.
- Main focus of Cost Based Optimization is minimizing Communication Cost and Computation Cost.

V. CONCLUSION

Accurate cost estimation is very important in distributed databases, because errors can have a huge impact on actual execution cost. The cost-based approach generally chooses an execution plan that is as good for large queries with multiple joins or multiple indexes [10][11]. The cost-based approach also improves productivity by eliminating the need to tune database statements on our own. The most important task for the CBO is to design an execution plan for an SQL statement. The CBO takes an SQL statement and tries to weigh different ways (plan) to execute it. It assigns a cost to each plan and chooses the plan with the smallest cost. The cheapest plan is the one that will use the least amount of resources (CPU, Memory, I/O, etc.) to get the desired output.

VI. PROPOSED WORK

We will perform query optimization using the MATLAB to choose a good execution strategy for a given query [5]. It can be observed from previous studies that MATLAB is usually better in choosing a good execution strategy than a traditional query optimization approach that uses a crisp cost model. Using MATLAB we can further explore other methods to establish a good cost model **[6][7**]for Distributed Environment and by investigating existing algorithms based on cost model. The mathematical equations can easily be modelled using MATLAB GUI interface in which a Graphical interface is provided to the developer which is helpful to save the time in cost calculations.

REFERENCES

[1]. Mishra, Ms Anju, Ms Gunjan Nehru, and Mr Ashish Pandey, "Dynamic Programming Solution for Query Optimization in Homogeneous Distributed Databases." International Journal of Engineering 1.6 (2012).
[2]. Liu, Mengmeng, "Efficient optimization and processing for distributed monitoring and control applications", Proceedings of the on SIGMOD/PODS 2012 PhD Symposium. ACM, 2012.
[3]. Xu, Zichen, Yi-Cheng Tu, and Xiaorui Wang. "PET: reducing database energy cost via query optimization", Proceedings of the VLDB Endowment 5.12 (2012): 1954-1957.
[4]. Golshanara, Ladan, Seyed Mohammad Taghi Rouhani Rankoohi, and Hamed Shah-Hosseini, "A multi-colony ant algorithm for optimizing join queries in distributed database systems." Knowledge and Information Systems (2013): 1-32.
[5]. Bausch, Daniel, Ilia Petrov, and Alejandro Buchmann, "Making cost-based query optimization asymmetry-aware", Proceedings of the Eighth International Workshop on Data Management on New Hardware. ACM, 2012.
[6]. Chandramouli, Badrish, et al., "Accurate latency estimation in a distributed event processing system", Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011.
[7]. P. Stone, P., P. Dantressangle, G. Bent, A. Mowshowitz, A. Toce, and B. Szymanski, "Relation algebra-coar segrained query cost modelsfor ddfds", In Proceedings of the Fourth Annual Conference of ITA, 2010.
[8]. Catania, Barbara, and Lakhmi Jain, "Advanced Query Processing: An Introduction", Advanced Query Processing, Springer Berlin Heidelberg, 2013. 1-13.
[9]. G. Bent. Hyperd, "Analysis and performance evaluation of a distributed hypercube database databases, In Proceedings of the Sixth Annual Conference of ITA, 2012.
[10]. A. Toce, A. Mowshowitz, P. Stone, P. Dantressangle, and G. Bent, "Hyperd: A hypercube topology for dynamic distributed federated databases", In Proceedings of the Fifth Annual Conference of ITA, 2011.
[11]. Görlitz, Olaf, and Steffen Staab, "Federated data management and query optimization for linked open data", New Directions in Web Data Management 1. Springer Berlin Heidelberg, 2011. 109-137.