ISSN 2229-5518

Cost-Based Query Optimization with Heuristics

Saurabh Kumar,Gaurav Khandelwal,Arjun Varshney,Mukul Arora

Abstract— In today's computational world,cost of computation is the most significant factor for any database management system.Searching a query from a database incurs various computational costs like processor time and communication time.Then, there are costs because of operations like projection, selection, join etc.DBMS strives to process the query in the most efficient way (in terms of 'Time') to produce the answer.In this paper we proposed a novel method for query optimization using heuristic based approach. In the proposed algorithm,a query is searched using the storage file which shows an improvement with respect to the earlier query optimization techniques. Also, the improvement increases once the query goes more complicated and for nesting query.

Index Terms—Heuristic,query,optimization,usage factor,storage file,magic tree,cost,weighted.

1 INTRODUCTION

DATABASE Management Systems(DBMS) have become a standard tool for shielding the computer user from details of secondary storage management.They are designed to improve the productivity of application programmers and to facilitate data access by computer-naïve end users. There have been two major areas of research in database systems.One is the analysis of data models into which the real world can be mapped and on which interfaces for different user types can be built. Such conceptual models include the hierarchical network, the relational and a number of semantics-oriented models that have been reviewed in a large number of surveys[14]. A second area of interest is the safe and efficient implementation of the DBMS. Computerized data has become the central resource of most organizations.Each implementation meant for production use must take into account by guaranteeing the safety of the data in the cases of concurrent access[9],recovery[11] and reorganization[12].

Imagine yourself standing in front of an exquisite buffet filled with numerous delicacies. Your goal is to try them all out, but you need to decide in what order. What exchange of tastes will maximize the overall pleasure of your palate? Although much lesspleasurable and subjective, that is the types of problem that query optimizers are called to solve. Given a query, thereare many plans that a Database Management System(DBMS) can follow to process it and produce its answer. All plans are equivalent in terms of their final output but vary in their cost, i.e., the amount of time that they need to run. What is the plan that needs the least amount of time? Such query optimization is absolutely necessary in a DBMS. The cost difference between two alternatives can be enormous. The path that a query traverses through a DBMS until its answer is generated is shown in Figure 1. The system modules through which it moves have the following functionality:

The Query Parser checks the validity of the query and then translates it into an internal form, usually a relational calculus expression or something equivalent.The Query Optimizer examines all algebraic expressions that are equivalent to the given query and chooses the one that is estimated to be the cheapest.The Code Generator or the Interpreter transforms the access plan generated by the optimizer into calls to the query processor. The Query Processor actually executes the query.

Research in query optimization has quickly acknowledged the exponential nature of the problem. While certain special cases can be solved in polynomial time (e.g., chain queries with no cross-products [1] or acyclic join queries under ASI cost models [2]), the general case is NP-hard (see [3], [2]).

- *Saurabh Kumar, Gaurav Khandelwal, Arjun Varshney, BITS-PILANI, HYDERABAD CAMPUS, HYDERABAD, INDIA {f2008446,f2008437,f2008320}@bits-hyderabad.ac.in*
- *Mukul Arora, BITS-PILANI, PILANI, RAJASTHAN, INDIA, f2007070@bits-pilani.ac.in*

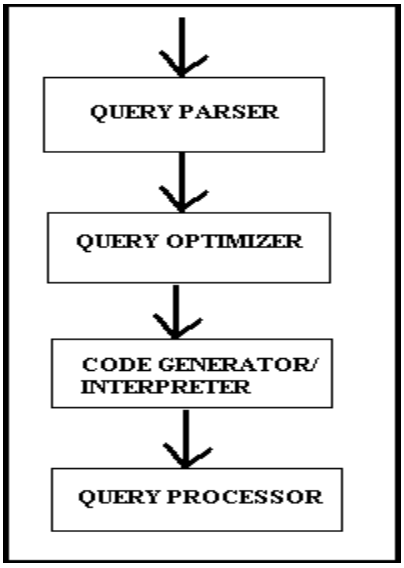

Figure 1: Query flow through a DBMS

Despite the inherent complexity of query optimization, algorithmic research has traditionally focused on exhaustive enumeration of alternatives (see [4] for the classical dynamic programming approach and [5], [6] for a transformation based approach). As queries become more complex, exhaustive algorithms simply cannot enumerate all alternatives in any reasonable amount of time. For instance, enumerating all join orders for a 15 table star query takes several minutes in commercial systems (but we have seen scenarios with queries that join more than 50 tables together). To be able to cope with such complex queries, several heuristics have been proposed in the literature[1]. However, previous work is limited to joins operators (i.e., they do not consider other relational operators like group-by clauses), do not consider the presence of indexes (which, as we will see, can drastically change the landscape of optimal plans), and can still be inefficient or inadequate in certain scenarios.

For the past twenty years, query optimization has been an intensively studied area of database system research. Most modern optimizers are cost-based in that they decide between execution plans by minimizing the estimated cost of evaluating the query. A fundamental technique used in cost estimation is cardinality estimation – optimizers take as input the cardinalities of tables at the leaves of a query tree, and then use selectivities of operators in the tree to estimate the cardinality of the input to operators further up in the tree. To convert cardinalities to costs, optimizers use functions that estimate the cost per tuple of each operator. While this approach is not perfect, it is very effective in most traditional DBMS applications. However, as we move to the Internet domain, this approach, in its current form, may not even apply. The reason for this is that if the leaves of the query tree correspond to incoming network streams, not only is their cardinality often not known, in some cases it may not even be well defined (e.g., in the case of infinite streams.)

2 RELATED WORK

The seminal paper on cost-based query optimization is [15]. Other optimization models have been proposed, especially in the areas of parallel query optimization, using cost models that arenot cardinality-based but instead deal with resource scheduling and allocation [7], [13]. The Britton-Lee optimizer could optimize for the first result tuple [17], while in Mariposa [16] the optimization criterion was a combination of execution time and resource utilization. Modeling streaming behavior through input rates and modeling network traffic as Poisson random processes have appeared in many contexts, including [3], although to our knowledge it has not been applied in the context of query optimization. A lot of work has been carried out in the areas of non-blocking symmetric join algorithms [2], [18], [19], which aim at producing plans that do not block their execution because of slow input streams.Framework in these indicates that with variable rate sources it is beneficial to employ such algorithms. In the same context, methodologies aiming at avoiding blocked parts of an execution plan at runtime [23] can benefit from framework of rate optimization by starting with and/or switching to plans for which the predicted output rate is maximized.

Some has worked on the basic concepts of query processing and query optimization [20] in the relational database domain. How a database processes a query as well as some of the algorithms and rule-sets utilized to produce more efficient queries was also presented along with the implementation plan using join ordering to extend the capabilities of Database. Some worked on SQoUT Project[21], which focuses on processing structured queries over relations extracted from text databases

Some related areas of work come from the adaptive query execution and dynamic re-optimization frameworks of [2] and [10]. In these frameworks, the main concern is to dynamically monitor an execution plan and identify points of sub-optimal performance. Once such points are identified, the system can choose to reorganize the plan in a way that is expected to yield better performance. In [10], such points are detected by incrementally measuring the cardinalities of partial outputs and comparing them to the optimizer's estimates. If the measured and estimated

cardinalities differ by a substantial amount, the optimizer is called to generate a better execution plan under the new information. In [2], the objective is to dynamically adapt and improve performance by rerouting inputs to particular operators thus improving overall performance. They initially choose an execution plan through a heuristic pre-optimizer and then continuously monitor the executing plan's performance. They also use runtime deviations from selectivity estimates as a criterion to identify sub-optimal performance. Work has also been done in the context of continuous queries over data streams in two directions: the first one aims at characterizing the behavior of these queries with respect to their memory requirements [1], [4]. Additionally, [6] deals with identifying and maintaining stream statistics for sliding window queries.Finally, some work also fits into the re-optimization frameworks of [12], [15], which focus on identifying performance bottlenecks of an already executing plan and ways to overcome them. Moreover, in a re-optimization framework like the one of [12], the performance crossing points,framework identified aids in scheduling when re-optimization should take place.

3 PROPOSED IDEA

As we know, the order of execution of the steps changes the cost of the execution. In query optimizer, a binary tree is obtained. In the proposed idea, all the dependent variables are set to one side of the branch of the tree. Each variable is assigned some weight in our proposed algorithm and for simulation purpose,we calculated the cost on the basis of the total weight. Weight is assigned according to how much time that variable or operation takes. Higher the computational time of the execution of the operation, higher will be the weight of that operation. When a search query is triggered, it initiates the search of the requested item. All the items will be at leaf. As we have discussed earlier, all dependent variables will be on same side of the branch of the tree obtained from the query from algorithm 1.In the proposed idea we have reordered certain variables and eliminated certain variables. Suppose there is a nested query, so running cost of let us say projection is some "a" units. So if in nested query, we run projection for ten times, then the cost will be 10*"a" units ,while in our proposed idea we shifted the projection to a state and from there we need to run the projection in the query for single time and the cost incurred will be only "a" units.

ALGORITHM 1

Function: creating magic tree.

a) Parsing of the query.
b) Building the tree.
c) Selection operation moves at the head node of the tree.
d) All subsequent selections are removed.
e) Projection operation moves next to the selection.
f) All subsequent projections are removed.
g) All dependent groups are formed and will branch to the same side of the tree.
h) Leaf node is a relation,so once we reach leaf,operation is terminated.
i) Search query is initiated.
j) As soon as the target is found,it moves to the projection and appropriate functions are performed.

Heuristics has always proved to be a useful tool.Sometimes the result may not show an improvement in early stage,but after sometime it will show an improvement as soon as its memory is filled with the usage information.In any organization or in any system of a database,generally the same query is executed after certain time.So every-time a tree must be built and then a new search query is executed. So, if we know in advance where the search is,then we can directly go there and it will save the time and hence decreases our cost.

ALGORITHM 2

Function: heuristics query search.

a) Once a query is run,a storage file is created.
b) Counter is set to company usage factor for each storage file.
c) Magic tree is stored in the storage file.Maximum number of storage files to be created is equal to company usage factor(c.u.f).
d) When the next query is run,it first checks the equivalence of the tree with any tree in any storage file.

e) If the trees are equal,then it will go to the path of search to the branch directly as stored in the storage file.
   1) If the search is successful than the appropriate actions are performed.
   2) If the search fails,it will search in the magic tree.

f) If the trees are not equivalent,then it will form its own magic tree as described in algorithm 1 and the counter is increased.

g) Refresh the storage file if counter>c.u.f.

4 RESULT ANALYSIS

Simulation:We executed our code on Linux machine in GCC compiler.Code is written in c and uses the concept of file handling.Tree data structure is used with dynamic memory allocation using linked list.

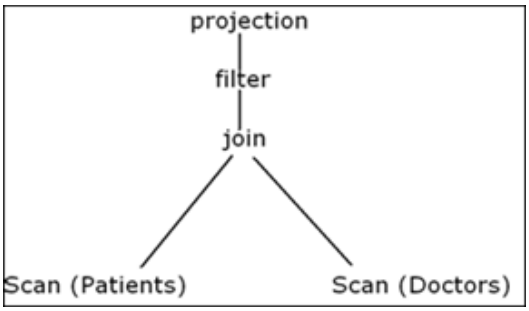

Figure 2: Query tree

Analysis: The result of the proposed idea is better as compared with the original query search. If we had the following query-

SELECT p.pname, d.dname

FROM Patients p, Doctors d

WHERE p.doctor = d.dname

AND d.gender = 'M'.

The initial tree of the following query is shown in figure 2.When this tree is converted into magic tree as shown in figure 3,it will incur some cost,but the cost of search in the magic tree will be less.Also the computational cost in the magic tree decreases because of the decrease in the number of operations.

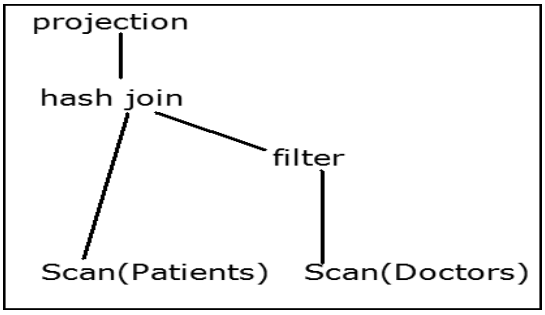

Figure 3: Query Tree

The estimated cost of the simple tree is 100 units whereas; the cost of magic tree is 50 units.But at the same time,there will be some cost incurred while converting simple tree into magic tree.Now, as we move towards the heuristics approach,magic tree is already there in the storage file,which will save the time of conversion and hence reducethe cost.

Figure 4 shows the Cost vs Time graph of the old simple query processing and with our proposed heuristic approach. As we can see, initially the cost is high, but later on as the time increases cost decreases. This can be explained as follows:

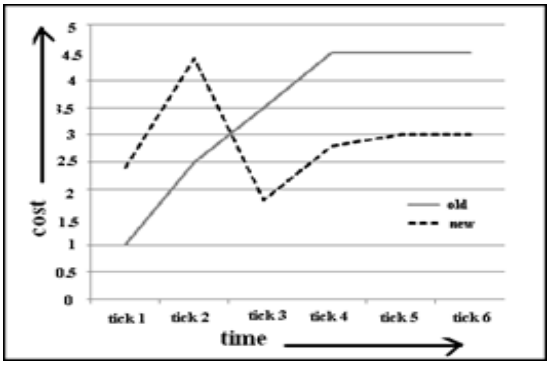

Figure 4: Cost vs Time

Initially, when in database some new query comes,then all steps of tree matching etc. are run which waste the time and hence increasesthe cost.But later,some standard query or duplicate query is run which matches our already built magic tree and saves the computational time.Figure 5 shows as the query becomes nested,the result improves significantly in our proposed heuristic approach.

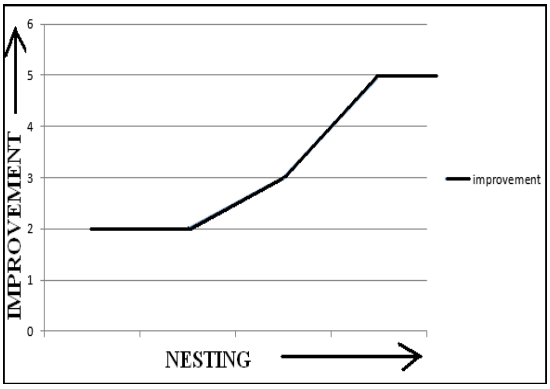

Figure 5: Performance vsComplexity

5 CONCLUSION AND FUTURE WORK

In this paper we described a novel approach of using a heuristic function to evaluate the efficiency of a query search in the database operations.Our simulation results indicate the improvement in query search as against the traditional query search. Therefore, we safely assume that heuristic based query optimization is a better approach to query optimization as compared to earlier query optimization techniques that are extensively used in the literature. With simulation run of our algorithm, further properties of join like left join,right join etc. can be extended.Along with these we can add some security in optimization step if feasible.

REFERENCE

[1] A. Arasu, B. Babcock, S. Babu, J. McAlister and J. Widom, Characterizing Memory Requirements for Queries over Continuous Data Streams, Stanford Techinical Report, November 2001,
http://dbpubs.stanford.edu/pub/2001-49

[2] R. Avnur and J. M. Hellerstein. Eddies: Continuously Adaptive Query Processing, Proceedings of the 2000 ACM SIGMOD Conference.

[3] D. Bertsekas and R. Gallager. Data Networks, Prentice Hall, 2nd edition, 1991.

[4] S. Babu, and J. Widom, Continuous Queries over Data Streams, SIGMOD Record, Sept. 2001.

[5] J. Chen, D. J. DeWitt, F. Tian and Y. Wang. Niagara-CQ: A Scalable Continuous Query System for Internet Databases, Proceedings of the 2000 ACM SIGMOD Conference.

[6] M. Datar, A. Gionis, P. Indyk and R. Motwani, Maintaining Strea m St tistics over Sliding Windows 2002 Ann al ACM SIAM

[7] M. N. Garofalakis and Y. E. Ioannidis. Parallel Query Scheduling and Optimization with Time- and Space-Shared Resources,
Proceedings of the 23rd International VLDB Conference.

[8] Polynomial Heuristics for Query Optimization Nicolas Bruno, C´esar Galindo-Legaria, Milind Joshi Microsoft Corp., USA

[9] Bernstein, P. A., And Goodman, N. 1981a. The Power of Natural Semijoins.SIAMJ.Comput. 10,4, 751-771.

[10] Z. G. Ives, D. Florescu, M. Friedman, A. Levy and D. S. Weld.An Adaptive Query Execution System for Data Integration,Proceedings of the 1999 ACM SIGMOD Conference

[11] Verhofstad, J. S. M. 1978. Recovery Techniques for Database Systems.ACM Comput.Surv. 10, 2 (June), 167-195.

[12] Sockut, G. H., And Goldberg, R. P. 1979. Database Reorganization--Principles and practice.ACM Comput.Surv. 11, 4 (Dec.) 371-395.

[13] C. Lee, C.-H. Ke, J.-B.Chang and Y.-H. Chen. Minimization of Resource Consumption for Multidatabase Query Optimization,Proceedings of the 3rd IFCIS Conference.

[14] Brodie, M., Mylopoulos, J., And Schmidt, J. W.,Eds. 1984. On Conceptual Modelling.Perspectives from Artificial Intelligence, Databases, and Programming Languages. Springer, New York

[15] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie and T. G. Price. Access Path Selection in a Relational Database Management System, Proceedings of the 1979 ACM SIGMOD
Conference.

[16] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin and Andrew Yu. Mariposa: A Wide-Area Distributed Database System, VLDB Journal, 1996, (5) 1:48-63.

[17] G. Schumacher. GEI's Experience with Britton-Lee's IDM, IWDM,1983, pp. 233-241.

[18] T. Urhan and M. J. Franklin.Xjoin: A Reactively-Scheduled Pipelined Join Operator, IEEE Data Engineering Bulletin, June 2000, (23) 2:27-33.

[19] A. N. Wilschut and P. M. G. Apers. Pipelining in Query Execution,Conference on Databases, Parallel Architectures and their
Applications, Miami, 1991.

[20] Prof.M.A.Pund, S.R.Jadhao, P.D.Thakare : A Role of Query Optimization in Relational Database,International Journal of Scientific & Engineering Research, Volume 2, Issue 1, January-2011.

[21] Alpa Jain, PanagiotisIpeirotis, Luis Gravano,Building Query Optimizers for Information Extraction:The SQoUT Project