Journal of Computer Sciences and Applications, 2014, Vol. 2, No. 2, 31-35
Available online at http://pubs.sciepub.com/jcsa/2/2/3
© Science and Education Publishing
DOI:10.12691/jcsa-2-2-3

Evaluating Query Execution Plans by Implementing Join Operators using Particle Swarm Optimization

Sambit Kumar Mishra[1,*], Srikanta Pattnaik[2], Dulu Patnaik[3,*]

[1]Department of Computer Sc.&Engg, Ajay Binay Institute of Technology, Cuttack, Odisha, India
[2]S.O.A. University, Bhubaneswar, Odisha, India
[3]Government College of Engineering, Bhwanipatna, Odisha, India
*Corresponding author: sambit_pr@rediffmail.com

Received September 01, 2014; Revised September 07, 2014; Accepted September 11, 2014

Abstract The nested structured queries as well as nested iteration as operator in both the logical and physical query algebra have been sometimes neglected in research. Interesting issues arise if multiple invocations of the same nested computation affect each other, e.g., the first invocation warms up the I/O buffer for subsequent ones. Other interesting issues arise if different nested computations compete for resources, e.g., I/O buffer or memory for sort and hash operations within inner queries. Nested computations are very important in practice, both because queries are authored using nested structured queries and because nested iteration based on index-to-index navigation often is the best execution plan. Therefore, nested computations could be a very fruitful research topic, both execution and optimization, and could probably also benefit from more dynamic and adaptive techniques than those in use today. While most resource issues have relatively little impact on optimal plan choices (even if they affect the ranking among different plans of fairly similar costs), one issue that is crucial in practice but usually ignored in academic research is the effect of buffer hits and faults in complex query plans. However, a conceptual model may be needed of nested queries that are substantially simpler, e.g., based on algebra expressions with a table of parameter values. In this paper it is aimed to find location of local minima of particle, random velocities of particles considering the relation schemes. The query plans related to relation schemes may be represented as particles. The query is optimized at compile time by that the complete query execution plans may be generated. The function evaluation of particles represented in terms of query plans in the relation schemes is planned to be done by considering random population of continuous values and velocities.

Keywords: *query, plan, OLAP, OLTP, tuple, swarm, Query Scrambling*

Cite This Article: Sambit Kumar Mishra, Srikanta Pattnaik, and Dulu Patnaik, "Evaluating Query Execution Plans by Implementing Join Operators using Particle Swarm Optimization." *Journal of Computer Sciences and Applications*, vol. 2, no. 2 (2014): 31-35. doi: 10.12691/jcsa-2-2-3.

1. Introduction

Query processing can be divided into query optimization and query execution. For queries with 10, 20, or 100 operators, this clear-cut distinction which is no longer applicable. If each operator's selectivity is consistently underestimated by a mere 10%, the error after 10 operators is a factor of 2.8; the error after 20 operators is a factor of 8.2; and after 100 operators, it is a factor of 37,649. Beyond a level of query complexity somewhere between 10 and 20 operators, selectivity and cost estimates are more guesses than predictions, and plan choices are more gambles than strategic decisions. Startup time-dynamic query evaluation plans are dynamic plans implementing decisions typically made at compile time that are delayed until the start of a query's evaluation. Run-time-dynamic query evaluation plans are plans that make decisions between alternative, potentially optimal, algorithms, operator ordering, and plan shapes based upon additional knowledge obtained while evaluating the plan. In this case Classical query processing, based on the distinction between compile-time and run-time, may be used. The query is optimized at compile time, thus resulting in a complete query execution plan (QEP). At runtime, the query engine executes the query, following strictly the decisions of the query optimizer. This approach has proven to be effective in centralized systems where the compiler can make good decisions. However, the execution of an integration query plan produced with this approach can result in poor performance because the mediator has limited knowledge of the behavior of the remote sources. Query Scrambling may be termed as a reactive approach to query execution; it reacts to data delivery problems by on-the-fly rescheduling of query operators and restructuring of the query execution plan. Query Scrambling is aimed at improving the response time for the entire query, and may actually slow down the return of some initial results in order to minimize the time required to produce the remaining portion of a query answer once all necessary data has been obtained from all of the remote sources.

The most exciting recent development in database query processing is usually the commercial use of materialized views. In a way, materialized views take the concept of early binding a step further than compile-time optimization. The benefit can be tremendous, as many online analytical processing (OLAP) tools and applications demonstrate every day with sub-second response times even for very large volumes of detail data. In effect, when materialized views work really effectively, complex query processing is reduced to index navigation very similar to online transaction processing (OLTP) processing, except for differences in the update load. Fancy techniques for large queries, e.g., shared scans, bitmap indexes, hash joins, and parallel query processing might once again be of little importance, just as they were when databases were used only for business operations (OLTP), not data analysis and business intelligence. The transaction processing has many benefits. It allows sharing of computer resources among many users. It shifts the time of job processing to when the computing resources are less busy. The online transaction processing facilitates and manages transaction oriented applications, typically for data entry and retrieval transaction processing. It has two key benefits, simplicity and efficiency.

There are many issues with respect to building redundant indexes on materialized views as well as to coherency, updates, invalidation, re-computation, incremental updates, etc.; probably the simplest policy (certainly from the perspectives of the application developer and the end user) is to treat materializations of view results similar to indexes, meaning instant updates within the original transaction, or even simply to index views in addition to tables, supporting both non-clustered and clustered indexes (the latter contain all columns in the table or the view).

In this paper a complementary approach using a non-blocking join operator is presented which is based on two fundamental principles:

1. It is optimized for producing results incrementally as they become available. When used in a fully pipelined query plan, answer tuples can be returned to the user as soon as they are produced. The early delivery of initial answers can provide tremendous improvements in the responsiveness observed by the users.

2. It allows progress to be made even when one or more sources experience delays. There are two reasons for this. First, the join requires less memory, which allows for bushier plans. Thus, some parts of a query plan can continue while others are stalled waiting for input. Second, by employing background processing on previously received tuples from both of its inputs, the Join operator can produce results even when both inputs are stalled simultaneously.

2. Review of Literature

Kristina Zelenay et.al [1] have discussed Deterministic algorithm, which is also known as exhaustive search dynamic programming algorithm, produces optimal left-deep processing trees with the big disadvantage of having an exponential running time. This means that for queries with more than 10-15 joins, the running time and space complexity explodes.

S. R. Madden et.al [2] have explained the query evaluation techniques to identify opportunities for sharing between operators, and to modify parts of the query plan to exploit these opportunities. The predicate indexing multiple query optimization technique shares work among them by indexing the selection predicates of the operators. For each incoming stream tuple this index is probed. It returns all satisfied predicates at a much lower cost than the naive strategy of evaluating each selection predicate individually one-by-one.

R. Avnur et.al [3] have elaborated in their paper that while query optimizers do a remarkably good job of estimating both the cardinality and cost of most queries, many assumptions underlie their mathematical models, such as the currency of the database statistics and the independence of predicates. Outdated statistics or invalid assumptions may cause significant estimation errors in the cardinality, and hence the cost of a plan, causing sub-optimal plans to be chosen. One proposed solution is to continually reoptimize the plan as each row, but this incurs impractically large re-optimization costs.

P.G. Selinger et.al [4] have discussed in their paper that most modern query optimizers determine the best plan for executing a given query by mathematically modeling the execution cost for each of many alternative query evaluation plans and choosing the one with the cheapest estimated cost. The execution cost is largely dependent upon the number of rows that may be processed by each operator in the query evaluation plan in the query.

As explained by Ganguly, S et.al [5], the general problem of the query optimization can be expressed as follows.

Assume a query q, a space of the execution plans E, and a cost function cost (q) associated to the execution of p ϵE. To find the execution plan calculating q such as the cost (q) is minimum an optimizer may be decomposed into three elements a search space, corresponding to the virtual set of all possible execution plans corresponding to a given query, a search strategy generating an optimal (or close to the optimal) execution plan and a cost model.

J. Chen et.al [6] have discussed in their paper that the plans of multiple queries are grouped together if they have common expression signatures, i.e., their plans have common syntactic characteristics. They have considered non-boolean queries and apart from syntactic similarities they have calculated the communication cost by computing and merging different plans.

A. Halevy et.al [7] have focused on query translation through mappings between the sources. They used data-level mapping between heterogeneous sources. It is understood that a distributed query may be involved with multiple sources, instead of a single source each time.

F. Neven et.al [8] have proposed a system to monitor various categories of data. The setting of the queries is centralized one where every source communicates only with a central source. Furthermore, it might not the amount of data transfer that is minimized but the total number of different database accesses.

J. Bleiholder et.al [9] have focused on single queries where the largest set of answers, following alternate paths through the graph connecting the sources, has to be computed at the lowest cost.

G. M. Thomas et.al. [10] have proposed an algorithm that combines greedy heuristics with dynamic programming in order to reduce the running time and space complexity of dynamic programming, while still producing good plans. The algorithm is capable of optimizing a query using dynamic programming by enumerating over the minimal number of connected subsets of the given join graph.

T. Neumann et.al [11] have discussed in their paper that while considering the join between two relations in a distributed setting there are a number of situations that must be taken into consideration when determining the most efficient means of carrying out the join. Situations involving possible shipping of complete relations between sites have been focused. They have aimed to reduce the network useage when performing cross site joins include the use of Semi-joins and Bloomjoins.

AlinDeutsch et.al [12] have introduced in their paper about tuple generating & equality generating dependencies and discussed embedded dependencies with disjunction and non equalities.

G. Tan et.al [13] have discussed sub-problems in join enumeration which depend on all preceding levels, whereas sub-problems. It is seen that the dynamic programming depend on only a fixed number of preceding levels. Thus, existing parallel dynamic programming algorithms cannot be readily applied to dynamic programming query optimization to achieve linear speedup.

G. Moerkotte et.al [14] have emphasized upon reduction of compilation times for large data warehouse OLAP queries which are typically star-shaped. OLTP queries are not our focus, since they can be optimized very fast. They have conducted experimental study on a real DB2 query workload and verified that compilation time is dominated by the number of join re-orderings.

W.-S. Han et el [15] have proposed the first framework to parallelize the size-based serial enumeration in a multi-core architecture. Extensive work has been done on heuristic or randomized query optimization to reduce query optimization time for large join queries.

N. W. Paton et.al [16] have discussed in their paper about dependency-aware parallel enumeration algorithm called DPEGeneric. The main thread invokes DPE-Generic. It then invokes the subroutine Enum And Build Partial Order to convert a fixed number of join pairs into a partial order.

3. Methods and Techniques Used

In this case the particle swarm optimization technique (PSO) is used to evaluate the query execution plans and to implement join operators. The particle swarm optimization (PSO) is a robust stochastic optimization technique which is based on the movement and intelligence of swarms. PSO applies the concept of social interaction to problem solving. It was developed in 1995 by James Kennedy and Russell Eberhart. It uses a number of agents or particles that constitute a swarm moving around in the search space looking for the best solution. Each particle is treated as a point in a N-dimensional space which adjusts its “flying” according to its own flying experience as well as the flying experience of other particles. Each particle keeps track of its coordinates in the solution space which are associated with the best solution (fitness) that has achieved so far by that particle. This value is called personal best, pbest. Another best value that is tracked by the PSO is the best value obtained so far by any particle in the neighborhood of that particle. This value is called gbest. The basic concept of PSO lies in accelerating each particle toward its pbest and the gbest locations, with a random weighted accelaration at each time step. Unlike in genetic algorithms, evolutionary programming and evolutionary strategies, in PSO, there is no selection operation. All particles in PSO are kept as members of the population through the course of the run.

4. Problem Formulation

A. Example

Assume that P(P) and Q(Q) are two union compatible relations.

The union of P(P) and Q(Q) is the set theoretic union of P(P) and Q(Q). The resultant relation R=PỤQ has tuples drawn from P and Q such that

R={ t| tϵP ỤtϵQ} and max(|P|, |Q|) <=|R|<=|P|+|Q|.

Similarly the Cartesian product of two union compatible relations, P and Q is nothing but the concatenation of tuples belonging to the two relations.

The resultant relation R= P X Q, has tuples drawn from P and Q such that R={t1||t2| t1ϵP ∩ t2ϵQ}, where tuple rϵR, and t1,t2 ϵ r.

The redundancies may be eliminated by decomposing the relation into several relations in higher normal form. The decomposition may be lossless if it may be recovered.

For example, a relation R may be in Boyce-codd normal form if for every non trivial functional dependencies X->Y in R where X,Y ϵ R, and X is a super key. X->Y is a violation if it is non trivial and X does not contain any key of R. So based on a Boyce-codd normal form violation X->Y, relation R may be decomposed into two relations, one with XỤY as its attributes and other with XỤ(attr(R)-X-Y) as its attributes.

B. Algorithm

```
Step 1:   The size of the swarm, i.e., popsize
          is set to 10.
Step 2:   The dimension of the problem, i.e.
          nrelations is set to 2;
Step 3:   The maximum number of iterations,
          i.e. maxquery is set to 100;
Step 4:   The number of swarm variables,
          npar=2;
Step 5:   The cognitive parameter, c1 = 1;
Step 6:   The social parameter, c2 = 4-c1;
Step 7:   nitialize the swarm and velocities and
          generate random population of
          continuous values
          par=rand(popsize,npar);
Step 8:   Generate random velocities, vel =
          rand(popsize,npar);
Step 9:   Evaluate the prime function,
          ff=par+vel/c2;
Step 10:  Calculate the population cost using
          the prime function
Step 11:  Calculate the min cost,
          minc=min(cost);
```

```
Step 12:    Calculate the mean of the cost,
            meanc=mean(cost);
Step 13:    Initialize the global minimum for
            each particle
Step 14:    Initialize the local minimum for
            each particle
Step 15 :   Retrieve the location of local
            minima
Step 16 :   Evaluate the cost of local minima
Step 17 :   Retrieve the best particle in initial
            population
Step 18 :   Initiate the iteration
            iter = 0;
            while iter < maxquery
            iter = iter + 1;
Step 19 :   Update the velocity, vel and
            evaluate inertia weight index
            considering maxquery and
            iterations
Step 20 :   Generate random numbers,r1,r2 by
            considering the popsize and
            number of warm variables
Step 21 :   Update the particle position
Step 22 :   Evaluate the new swarm
Step 23:    Evaluate the cost of swarm
Step 24:    Update the best local position for
            each particle
Step 25 :   Update the index
Step 26 :   Evaluate the best min for the
            iterations
Step 27:    Evaluate the average cost for the
            iterations
```

Table 1. Location of local minima of particle

Relation(R1)	Relation(R2)
0.99327	0.21992
0.054167	0.26077
0.49369	0.77139
0.99051	0.89836
0.3117	0.57969

Table 2. (Vel) Random velocities

Relation(R1)	Relation(R2)
0.508	0.35572
0.094959	0.38302
0.5713	0.45707
0.61635	0.30588
0.70711	0.71084

Table 3. Estimated cost of plan

Relation (R1)	Relation (R2)
2.1559	0.55841
0.13999	0.64921
1.1778	1.6951
0.40356	1.8987
0.8591	1.3963

Table 4. ff (Function evaluation by considering random population of continuous values and velocities)

Relation(R1)	Relation(R2)
1.1626	0.33849
0.08582	0.38844
0.68412	0.92374
0.3045	1.0003
0.5474	0.81663

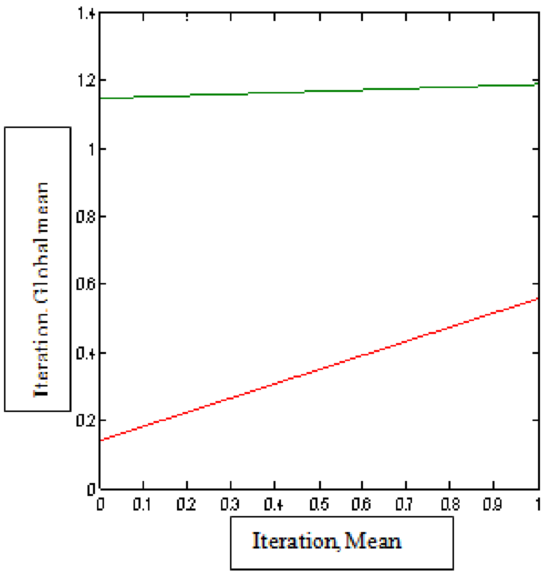

Figure 1. Mean VS Global Mean

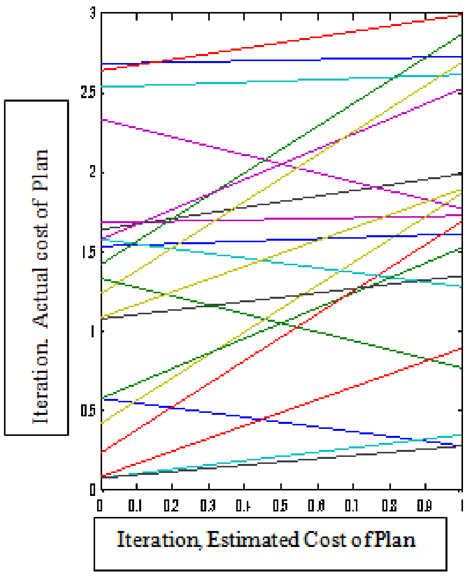

Figure 2. Estimated cost of plan VS Actual cost of plan

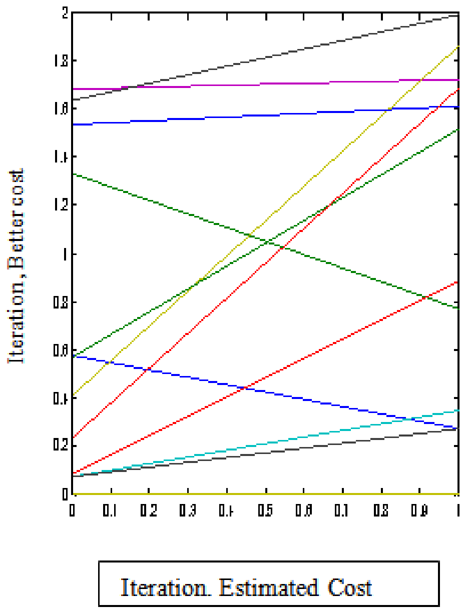

Figure 3. Estimated cost VS Better cost

5. Discussion and Future Direction

As the area of data management for the Internet has gained in popularity, recent work has focused on effectively dealing with unpredictable, dynamic data volumes and transfer rates using adaptive query processing techniques. Important requirements of the Internet domain include: (1) the ability to process the web data as it streams in from the network, in addition to working on locally stored data; (2) dynamic scheduling of operators to adjust to I/O delays and flow rates; (3) sharing and re-use of data across multiple queries, where possible; (4) the ability to output results and later update them. Data may often be pre fetched and cached by the query processor, but the system may also have to provide data freshness guarantees. If the domain includes large numbers of similar queries being posed frequently, the query processor should generate query plans with a focus on materialization of partial results for future reuse, and it should make use of common sub expressions. It is understood that the better plans may be achieved by executing a set of queries by modeling the execution cost for each of alternative query evaluation plans and choosing the one with the cheapest estimated cost. The execution cost is largely dependent upon the number of rows that may be processed by each operator in the query evaluation plan in the query.

6. Conclusion

During the process of implementation, the techniques to evaluate query execution plans have been classified and compared. The opportunities offered by such techniques to modify a query plan have been identified. In addition, while implementing dynamic plans in the dynamic query processing environments, the systems are required to be classified according to the problem focused on along with the objectives, the nature of feedback and the frequency of adaptation.

The survey reveals the inadequacy of existing techniques to adapt to environments where the pool of resources is subject to changes. In particular, the paper provides evidence to the need for research into dynamic query environment where resource availability, allocation and costing are not, by definition, decidable at compile time.

References

[1] Kristina Zelenay, “Query Optimization”, ETH Zürich, Seminar Algorithmen für Datenbanksysteme, June 2005

[2] S. R. Madden, M. A. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. in Proc. SIGMOD, 2002.

[3] R. Avnur and J. M. Hellerstein, Eddies: Continuously Adaptive Query Optimization, SIGMOD 2000.

[4] P.G. Selinger et al. Access Path Selection in a Relational DBMS. SIGMOD 1979.

[5] Ganguly, S., Hasan, W., Krishnamurthy, R.: Query Optimization for Parallel Execution. In: Proc. of the 1992 ACM SIGMOD int’l. Conf. on Management of Data, vol. 21, pp. 9-18. ACM Press, San Diego (1992).

[6] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: a scalable continuous query system for internet databases.In SIGMOD, p. 379-390. ACM, 2000.

[7] A. Halevy, Z. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The piazza peer data management system. IEEE Trans. Knowl. Data Eng., 16(7):787-798, 2004.

[8] F. Neven and D. Van de Craen. Optimizing monitoring queries over distributed data. In EDBT, p. 829-846, 2006.

[9] J. Bleiholder, et al. Query planning in the presence of overlapping sources. In EDBT, p. 811-828, 2006.

[10] G. M. Thomas Neumann. Analysis of two existing and one new dynamic programming algorithm for the eneration of optimal bushy join trees without cross products. In Proceedings of the 32nd international conference on Very large data bases, pages 930-941. VLDB Endowment, 2008.

[11] T. Neumann. Query simplification: Graceful degradation for join-order optimization. In C. Binning and B. Dageville, editors, SIGMOD-PODS09: Compilation Proceedings of the International Conference on Management of Data 28th Symposium on Principles of Database Systems, pages 403-414, Providence, USA, June 2009. Association for Computing Machinery (ACM).

[12] AlinDeutsch, Bertram Ludaascher, and Alan Nash. Rewriting queries using views with access patterns under in tegrit y constraints. The or.Comput.Sci, 371(3):200-226, 2007.

[13] G. Tan, N. Sun, and G. R. Gao. A parallel dynamic programming algorithm on a multi-core architecture. In SPAA, 2007.

[14] G. Moerkotte and T. Neumann. Dynamic programming strikes back. In SIGMOD, 2008.

[15] W.-S. Han, W. Kwak, J. Lee, G. M. Lohman, and V. Markl. Parallelizing query optimization. In VLDB, 2008.

[16] N. W. Paton, V. Raman, G. Swart, and I. Narang. Autonomic query parallelization using non-dedicated computers: An evaluation of adaptivity options. In ICAC, 2006.