A Cost Evaluator for Parallel Deductive Database Systems

A. Hameurlain, E. Ceccato, F. Morvan,

Université Paul Sabatier, Lab. IRIT
118 Route de Narbonne -31062 Toulouse - France
Tél :+33 61 55 67 65 ; Fax : +33 61 55 62 58 ; E-mail : hameur@irit.fr

Extended Abstract

1. Introduction and motivations

New applications such as office information system, geographical information systems, CAD, expert systems and other applications which require large knowledge bases, need improved efficiency in response time from advanced database management systems (DBMS). This goal can be achieved by optimizing SQL query (eventually recursive [Ban86]) compilers and using optimal parallel execution strategies for query processing [Val88].

The conception of an optimizer is fundamental to obtain high performance. The rôle of the optimizer for recursive or non recursive SQL queries for parallel execution is to generate an efficient execution plan to get the information requested by the user. Thus, the optimizer must be capable of selecting one of several execution plans with highest performance. To estimate query processing costs, the optimizer needs another system called cost evaluator. However, the specificities of the hardware and software environment are usually not taken into account by the optimizer with classical cost models. Indeed, each DBMS contains a specific module for estimating query costs, which is not compatible with other systems. An elegant solution consists in separating the optimizer and the cost evaluator [And91]. The advantage of this solution is to provide an *extensible, flexible and adaptable* cost evaluator. The cost evaluator is seen as an intelligent black box by the optimizer and can therefore be used independently by other DBMS. This design leads to the parametrization of the cost evaluator with libraries describing the environment (Architecture, database profile, cost models). The cost evaluator proposed previously by [And91] only applies to the optimization of SQL queries for parallel execution. We therefore extended this evaluator to take into account the optimization of recursive queries as follows:

(i) We define and integrate a new parameter which evaluates the number of new tuples deduced by the specialized operators (i.e. Transitive Closure [Ioa88], External Clo-

sure,...). With knowledge of the number of tuples, the number of processors can be estimated [Ham92] for the optimal parallel execution of a relational or specialized operator.
(ii) We extend the database profile by adding the *canonical data structures* associated with each referenced base or deduced relation into premises of the recursive rules of the intentional database.
(iii) We integrate a new parameter, called *acceleration factor* Ω, to evaluate the benefit from the propagation of the attributes and the number of processors during the parallelization process.
(iv) We include in the cost model a *decision table* to choose the best of sequential or parallel algorithms for each execution strategy.

In this paper we briefly recall the various steps in the optimization of recursive or non recursive SQL queries. This points out the contribution of the cost evaluator to the optimizer. Then we describe the functions of a cost evaluator for parallel deductive systems. Finally, we explain the advantage of a multi-environment oriented cost evaluator working independently from the optimizer.

2. Query Compiler Architecture for a Parallel Deductive System

The compilation process consists of several stages up to the generation of an optimal program, which may be parallel, to be executed on the algebraic machine (Cf. Appendix). Construction of such a program proceeds as follows:

(a) Analyser and Resolution Process: The SQL query is analysed syntactically and semantically then transformed in an execution plan in the shape of resolution graph [Ham90].

(b) Optimization: The resolution graph is restructured, during the logical optimization, by applying the transformation rules, as well as the algorithm for selection propagation [Agr89][Ham90][Kif86] in the recursive queries sub-graphs. The joins are ordered, during the physical optimization by using the keys and indexes stored in the *database profile*. Thereafter the resolution graph is enriched with new information such as the *number of processors*, the *local response time* calculated by the cost evaluator during the parallelization phase of relational and specialized operations (Transitive Closure, External Closure). The last phase consists in scheduling in an eventually parallel fashion all the operations of the execution plan in order to minimize the query response time.

(c) Code generation: The optimal resolution graph enriched with the control and communication operations is translated by the code generator into a code "object" which is executed by the algebraic machine on the various nodes of the parallel architecture.

3. Functional aspects of the cost evaluator

Let us first define the various *metrics*, i.e. costs, to be calculated by the evaluator. Some representation of these costs is required and the evaluator must know the value of each parameter in the algebraic machine environment. These data are stored in *libraries* we will describe later.

The different metrics are:

(i) The number of processors NP: For each resolution graph operation, the evaluator must determine the best number of processors to minimize response time. But response time is mainly function of the size of the operand relations. First the evaluator must calculate the number of deduced tuples. Other parameters such as tuple production, distribution or transfer times are stored in libraries.

(ii) The local response time LRT: As a function of the algorithm chosen for the operation, the evaluator may determine the local response time thanks to a set of cost functions (communication, distribution and effective time). Amongst the parameters implicated in these functions, some are stored in the libraries (architecture, BD profile), others have been determined previously (numbers of processors and of deduced tuples).

(iii) The acceleration factor Ω*:* During the parallelization phase, the optimizer must determine whether propagation is beneficial or not. The acceleration factor is calculated by the evaluator and defined as the ratio between local response times with and without propagation.

The values of these parameters are stored in the following libraries:

(1) Architecture: This data structure contains the information related to all types of architecture (memory, cpu, network) even if only a given architecture is being used at a given moment. This ensures the evaluator may be used in any environment.

(2) EDB&IDB Profile: This contains information on the extentional database EDB (relations, attributes, keys, indexes,...) and intentional IDB one concerning the canonical data structure of the base or deduced relations used in the definition of the recursive relations. This canonical data structure is represented by a set of parameters which are: Fin, Fout, expansion factor.

(3) Cost models: This contains, on one hand, the set of cost functions which allow the evaluation of the metrics defined above and on the other hand, a decision table which allows selection of the best algorithm for each operation as a function of some data (size of the relations...)

4. Conclusion

In this paper we presented a multi-environment cost evaluator, independent from the optimizer. The main advantages of separating the optimizer and the cost evaluator are:

(i) Extensibility: new environments (architecture, DB_profil, access methods, new algorithms) can be taken into account simply by extending the libraries.

(ii) Flexibility: the evaluator has a structure well dissociated from the optimizer. The volume of information it contains can therefore be increased without any modification of the optimizer. Furthermore, the internal structure of the cost evaluator is also flexible since new parameters can be integrated and defined all along without modifying its structure.

(iii) Adaptability to various database Machines. To use the cost evaluator it suffices to call it with its environment. Thus, the environment can be changed without having to rewrite the software.

Appendix

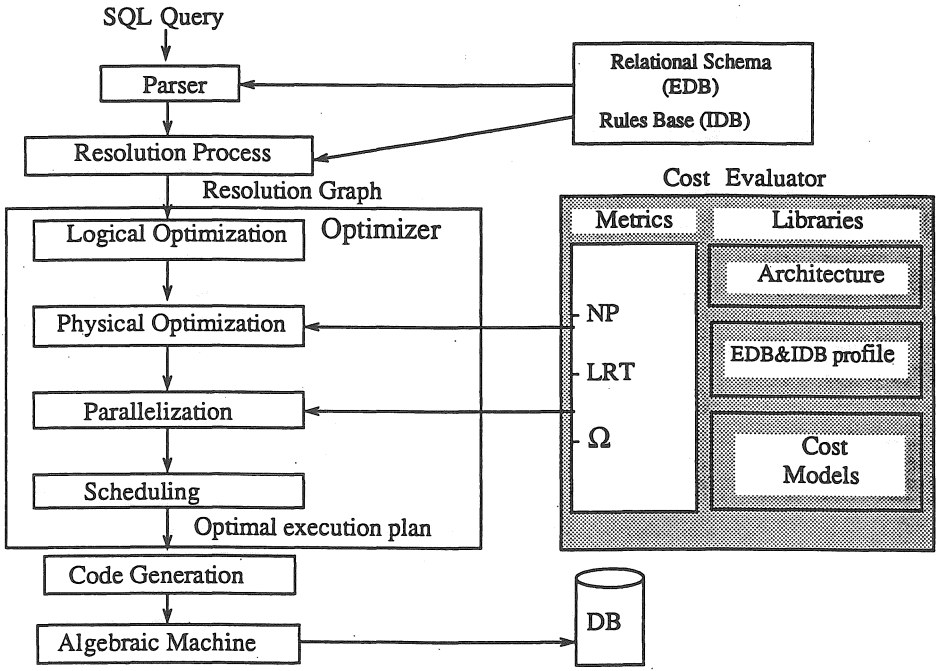

Figure 1 : Architecture of the SQL query compiler

References

[Agr89] R. Agrawal, P. Denabu, "Moving Selections into Linear Least Fixpoint Queries", IEEE Transaction Knowledge and Data Engineering. Vol. 1, No. 4, Dec 89, pp. 424-432.

[And91] F. Andrès et al., "A Multi-Environment Cost Evaluator for Parallel Database Systems", 2nd Intl. Symp. on Database Systems for Advanced Applications, DASFAA'91, Tokyo, April 1991.

[Ban86] F. Bancilhon, R. Ramakrishan, "An Amateur's Introduction to Recursive Query Processing Strategies", Proc. of ACM- SIGMOD, Washington, May 1986, pp. 16-52.

[Ham90] A. Hameurlain, F. Morvan, "An Algorithm For Selection Operator Propagation in Resolution Graph", Intl. Conf. On Database and Expert Systems Applications, Vienna, Aug. 1990, pp. 550-553

[Ham92] A. Hameurlain et al., "Processor Number Estimation for Optimal Parallel Execution of recursive Queries", European Workshops on Parallel Computing, Barcelona, March 1992.

Ioa88] Y.E. Ioannidis, R. Ramakrishan, "Efficient Transitive Closure Algorithm", Proc. of the 14th VLDB Conf. Los Angeles, California, pp. 382-394, 1988.

[Kif86] M. Kifer, E. Lozinskii, "Filtering Data Flow in Deductive Database", Proc. of the Intl. Conf. on Database Theory, Lecture Notes in computer Science, No. 243, Springer-Verlag, Rome, Sept 1986.

[Val88] P. Valduriez, S. Khoshafian, "Parallel Evaluation of the Transitive Closure of a Database Relation", Intl. Journal of Parallel Programming, Vol. 17, No. 1, Feb. 1988.