On the Efficient Construction of Query Optimizers for Distributed Heterogeneous Information Systems
A Generic Framework

Tianxiao Liu, Dominique Laurent and Tuyêt Trâm Dang Ngoc
ETIS, CNRS, ENSEA - Cergy-Pontoise University, 95000 Cergy-Pontoise, France

Keywords: Distributed Heterogeneous Information Systems, Query Optimization, Search Strategy, Cost Model.

Abstract: It is now common practice to address queries to Distributed Heterogeneous Information Systems (DHIS). In such a setting, the issue of query optimization becomes crucial, and more complex than in centralized homogeneous approaches. Indeed, the optimization processing must be as flexible as possible so as to apply to different database models, and integrate different cost models. In this paper, we present a generic framework for query optimization in the context of DHISs, with the goal of *facilitating* the implementation of *efficient* query optimizers. To this end, we identify all necessary components for building such a query optimizer and we define the basic functions that have to be implemented. Moreover, we report on experiments showing that our approach allows for an efficient query optimization in the context of DHISs.

1 INTRODUCTION

It is well known that one of the main features of relational database systems is to allow for *query optimization*. When optimizing a query Q, Q is first transformed into an initial execution plan, which is then transformed into other equivalent plans using transformation rules. These candidate plans form a search space that is explored by the query optimizer module in order to find an optimized execution plan (*i.e.*, an execution plan having a lower execution cost). As the size of the search space is generally huge, a search strategy is used to efficiently find such an optimized execution plan. Query optimization processing in a given database is based on the following information:

- **Meta-data Model:** database schema, data location, data accessibility, etc.
- **Data and Query Model:** relational, object oriented, semi-structured, services, etc.
- **Optimization Goals:** minimize runtime, minimize money cost, minimize the access to networks, etc.
- **A Search Strategy:** exhaustive, incremental, genetic, dynamic, etc.

When queries are addressed to a single database for which the information above is known in advance, query optimization allows for efficiently minimizing the computation cost. However, it is well known that changing a piece of the information mentioned above requires significant efforts in source code writing.

On the other hand it is now common practice to address queries to Distributed Heterogeneous Information Systems (DHIS). In this setting, the evaluation of a given query requires accessing different heterogeneous data sources, and so, query optimization becomes more complex. Indeed, in a DHIS, the optimization processing must be as flexible as possible so as to (1) consider databases located in different sites, (2) apply to different data models, and (3) integrate different cost models. The contribution of this paper is to propose a *generic framework* for integrating various optimization techniques in order to build efficient optimizers in the context of DHISs. In this framework, we consider the following components:

- Plug-in modules dealing with meta-data, data models, queries, search strategies and transformation rules, respectively.
- Basic functions for an easy and efficient implementation of search strategies.

We have implemented our generic framework, and the experiments reported in this paper show that our approach offers the necessary *flexibility* when designing *efficient* query optimizers for DHISs. More precisely, we show that, by using our approach:

1. the number of code lines for implementing a new

Liu T., Laurent D. and Trâm Dang Ngoc T..
On the Efficient Construction of Query Optimizers for Distributed Heterogeneous Information Systems - A Generic Framework.
DOI: 10.5220/0003987801870190
In *Proceedings of the 14th International Conference on Enterprise Information Systems* (ICEIS-2012), pages 187-190
ISBN: 978-989-8565-10-5

strategy or designing a new optimizer is drastically reduced, as compared with an implementation from scratch;

2. the generated optimized execution plan reduces the processing time by 28 times.

The paper is organized as follows: In Section 2 we describe our generic framework, in Section 3, we report on experiments, in Section 4, we overview related work, and in Section 5, we conclude the paper and offer research directions for future research.

2 THE GENERIC FRAMEWORK

Figure 1 shows the main components of our generic framework, namely:

1. *Source Description*, such as data schema, source location, cost model, etc.
2. *Transformation Rules*, used to transform an execution plan into another equivalent execution plan.
3. *A Collection of Search Strategies*, which are metaheuristic algorithms used for optimization.
4. *A Toolbox of Five Basic Functions*, implemented *only once* and reused to combine optimization processes for different search strategies.

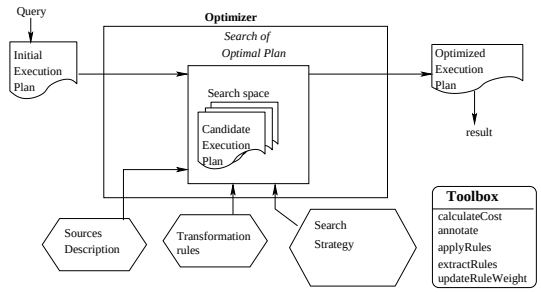

Figure 1: The main components of our framework.

2.1 Generating Execution Plans

We implemented a description module called GSD (standing for *Generic Source Description* and formerly called TGV in (Dang-Ngoc and Travers, 2007)) for annotating any kind of information about data sources. However, any module collecting data source information can be used, as long as it implements the following function, taking an execution plan as input and returning the annotated execution plan:

annotate(exec_plan) → annotated_exec_plan

Given an annotated execution plan, the optimizer should be able to calculate the cost of this execution plan. In our generic framework, this step is achieved through a call to the following function:

calculateCost(exec_plan) → cost_value

Clearly, the implementation of this function depends on the characteristics of the data source where the execution plan is to be run. This information is provided by the annotation returned by the annotate function.

On the other hand, in order to generate a new execution plan from a given one, *transformation rules* are applied. Usually, the following kinds of transformation rules are considered: (1) Logical rules, that reflect basic properties of the underlying algebra; (2) Physical rules that specify the best way a given operation can be computed; (3) User defined rules, such as specific commutation rules.

In our generic framework, such a rule manager is implemented through the following two functions:

extractRules(exec_plan) → set_of_rules
applyRule(rule, exec_plan) → exec_plan

A call to the function extractRules for a given execution plan generates all rules that can be applied for transforming this execution plan into another one. On the other hand, a call to the function applyRule, for a given rule and a given execution plan, applies the given rule to generate a new execution plan.

We emphasize that these four functions do *not* depend on the optimization strategy. Therefore, they are implemented only once for a fixed DHIS, and changing the optimization strategy does *not* require any additional implementation work in this respect.

2.2 Search Strategy

We recall that, in order to avoid searching the *whole* set of execution plans of a given query, search strategies are used. These strategies are based on well known algorithms such as Dynamic Programming (Selinger et al., 1979), Simulated Annealing (Kirkpatrick et al., 1987), or Genetic Algorithm (Goldberg, 1989). Of course each strategy has its own characteristics, and thus, changing from one search strategy to another requires some implementation work.

However, in our generic framework, only the following three functions have to be reconsidered when changing the strategy:

chooseRules(exec_plan, integer) → set_of_rules
updateRuleWeight(rule) → value
getOptimizedPlan(exec_plan) → exec_plan

The function chooseRules returns a set of rules (whose cardinality is the value of the second parameter) chosen from the set of rules returned by a call to extractRules. These rules are applied to the current execution plan, to generate new execution plans

whose costs are computed using the function calculateCost. Finally, the function getOptimizedPlan allows to compute the execution plan with lowest cost, using a given search strategy.

3 EXPERIMENTS

We now present experimental results on the development of our generic framework. The considered DHIS contains 12 distributed heterogeneous data sources (relational, xml, object-relational and Web service), and we have chosen typical queries with inter-site binary operators to be optimized.

Search strategies are compared in the following two respects: We first compare their quality with respect to the *absolute optimal* plan obtained by searching the whole search space, and second, we compare the runtimes needed by each search strategy for the computation of its output optimized execution plan. Figure 2 shows the comparison of various search strategies: Simulating Annealing, Genetic Algorithm, Incremental (Nahar et al., 1986), Dynamic Programming (Selinger et al., 1979), Random (Swami, 1989) and Ant Colony (A. Colorni, 1991).

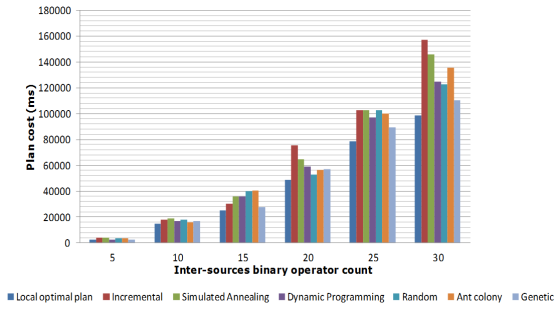

Figure 2: Quality of optimization strategies.

For queries with 10, 15 and 20 inter-site operators, we compare the quality of execution plans found by each strategy with the absolute optimal. We can see that for most strategies, the optimized execution plan is very close to the absolute optimal execution plan.

Figure 3 shows that the time spent for finding the optimized plan increases almost linearly with respect to the complexity of the query being optimized. It can also be seen that the average time for computing the optimized execution plan is in the order of 2 seconds, whereas we found that computing the absolute optimal execution plan takes about 2 minutes in average. This clearly shows that any search strategy is much more efficient than the exhaustive strategy.

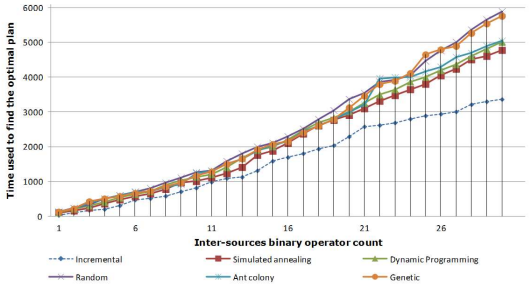

Figure 3: Runtime for the computation of the optimal plan.

We also note from Figure 3 that the Incremental strategy is the most efficient in terms of runtime, whereas the other strategies show similar and acceptable performance. However, Figure 2 shows that, among all other strategies, the efficiency of the Incremental strategy is obtained at the cost of computing the worst execution plan, in terms of quality. More generally, our experiments have shown that the runtime of the optimized execution plan is reduced by up to 28 times, as compared with the runtime of the non optimized initial execution plan. We refer to (Liu, 2011) for more details in this respect.

Regarding now the efforts in coding search strategies, we recall that applying a new search strategy only requires to modify the implementation of two the functions getOptimizedPlan and chooseRules. Our experiments show that the Exhaustive search strategy can be implemented with about 260 lines of Java code. Knowing that the total number of code lines of the whole implementation of the query optimizer is 5000, implementing the Exhaustive search strategy represents only 5.2% of these 5000 lines.

Assuming that the query optimizer has already been implemented using the Exhaustive strategy, changing from the Exhaustive search strategy to the Incremental search strategy requires to replace the 260 code lines of the Exhaustive strategy with the 280 code lines for implementing the Incremental strategy. This represents only 5.2% of the 5000 code lines of the whole implementation. This example clearly shows that our generic framework is flexible enough to allow query optimization in various environments.

4 RELATED WORK

Although search strategies are the core component of query optimization, most mediation systems use exhaustive search strategies on a portion of the search space (System R* (Daniels et al., 1982), DIOM (Liu and Pu, 1997), DISCO (Naacke et al., 1998), Garlic (Roth et al., 1999)) or dynamic programming (Star-

Burst (Haas et al., 1989), Garlic (Roth et al., 1999)). In (Josifovski and Risch, 2002), the authors propose an approach using the AMOS*II* mediator database system, in which a given query is transformed into an executable object algebraic execution plan. The optimization process is based on built-in algebraic operators and a built-in cost model for local data.

Moreover, in (Josifovski and Risch, 2002), Dynamic Programming, Simulated Annealing, or Random can be used as search strategies and so, this approach is the only system that offers the choice between different search strategies, as we do. However, the strategies proposed in (Josifovski and Risch, 2002) are hard-coded, which offers less flexibility than in our approach.

We finally mention that, in (Stonebraker 2008), the author stresses that database systems are now more and more specialized (e.g. OLPT vs. OLAP systems), and thus, that there is no hope in designing efficient common optimization techniques for all these systems. We therefore conclude that our approach of providing a generic framework for the integration of different optimization techniques can contribute in the design of *efficient* and *flexible* query optimizers in DHISs.

5 CONCLUSIONS

In this paper, we have proposed a generic framework for query optimization in the context of DHISs. Our framework is composed of a set of basic functions, whose implementation takes into account all aspects of query optimization such as transformation rule, cost estimation, construction and annotation execution plans, and search strategy. The experimental results reported in this paper show the high flexibility of our framework used to create or upgrade easily optimizers with high performance. Moreover, our experiments also show that using our generic framework allows for significant gains of runtime.

Regarding future work, we plan to investigate the following issues: (i) implementing a cache system in order to optimize the cost computation of a given execution plan, (ii) consider cost models in the context of cloud computing, and (iii) incorporate multi-criteria optimization techniques in our framework.

REFERENCES

Colorni, A. et al (1991). Distributed optimization by ant colonies. In *Conférence européenne sur la vie artificielle*, pages 134–142.

Dang-Ngoc, T.-T. and Travers, N. (2007). Tree graph views for a distributed pervasive environment. In *International Conference on Network-Based Information Systems (NBIS)*.

Daniels, D., Selinger, P. G., Haas, L. M., Lindsay, B. G., Mohan, C., Walker, A., and Wilms, P. F. (1982). An introduction to distributed query compilation in r*. In *DDB*, pages 291–309.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Haas, L. M., Freytag, J. C., Lohman, G. M., and Pirahesh, H. (1989). Extensible query processing in starburst. In *ACM SIGMOD*, pages 377–388.

Josifovski, V. and Risch, T. (2002). Query decomposition for a distributed object-oriented mediator system. *Distributed and Parallel Databases*, 11:307–336.

Kirkpatrick, S., Gelatt, Jr., C. D., and Vecchi, M. P. (1987). Readings in computer vision: issues, problems, principles, and paradigms. chapter Optimization by simulated annealing, pages 606–615. Morgan Kaufmann.

Liu, L. and Pu, C. (1997). An adaptive object-oriented approach to integration and access of heterogeneous information sources. *Distributed and Parallel Databases*, 5:167–205.

Liu, T. (2011). *Proposition d'un cadre générique d'optimisation de requêtes dans les environnements hétérogènes répartis*. PhD thesis, Université de Cergy-Pontoise, France. (French).

Naacke, H., Gardarin, G., and Tomasic, A. (1998). Leveraging mediator cost models with heterogeneous data sources. In *ICDE*, pages 351–360.

Nahar, S., Sahni, S., and Shragowitz, E. (1986). Simulated annealing and combinatorial optimization. In *23rd Design Automation Conference*.

Roth, M. T., Ozcan, F., and Haas, L. M. (1999). Cost models do matter: Providing cost information for diverse data sources in a federated system. In *VLDB*, pages 599–610.

Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., and Price., T. G. (1979). Access path selection in a relational database management system. In *ACM SIGMOD*, pages 23–34.

Stonebraker, M. (2008). Technical perspective - One size fits all: an idea whose time has come and gone. *CACM*, 51(12):76.

Swami, A. (1989). Optimization of large join queries: Combining heuristics and combinatorial techniques. In *ACM SIGMOD*, pages 367–376.