Expert Systems with Applications 41 (2014) 6934–6944

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Query join ordering optimization with evolutionary multi-agent systems

Frederico A.C.A. Gonçalves [a,c], Frederico G. Guimarães [a,*], Marcone J.F. Souza [b]

[a] Department of Electrical Engineering, Federal University of Minas Gerais, Belo Horizonte, Brazil
[b] Department of Computer Science, Federal University of Ouro Preto, Ouro Preto, Brazil
[c] IT Center, Federal University of Ouro Preto, Ouro Preto, Brazil

ARTICLE INFO

Article history:
Available online 17 May 2014

Keywords:
Join ordering problem
Query optimization
Multi-agent system
Evolutionary algorithm
Heuristics

ABSTRACT

This work presents an evolutionary multi-agent system applied to the query optimization phase of Relational Database Management Systems (RDBMS) in a non-distributed environment. The query optimization phase deals with a known problem called query join ordering, which has a direct impact on the performance of such systems. The proposed optimizer was programmed in the optimization core of the H2 Database Engine. The experimental section was designed according to a factorial design of fixed effects and the analysis based on the Permutations Test for an Analysis of Variance Design. The evaluation methodology is based on synthetic benchmarks and the tests are divided into three different experiments: calibration of the algorithm, validation with an exhaustive method and a general comparison with different database systems, namely Apache Derby, HSQLDB and PostgreSQL. The results show that the proposed evolutionary multi-agent system was able to generate solutions associated with lower cost plans and faster execution times in the majority of the cases.

1. Introduction

Database Management Systems (DBMS) are very complex software systems designed to define, manipulate, retrieve and manage data stored in a database. DBMS have an essential role in the information based society and represent critical components of business organization. Relational Database Management Systems (RDBMS) are those based on the relational model, which is the focus of this work. The information recovery depends on a database query and, as in Fig. 1, this query can be formed by many relations, which can be filtered and/or connected by different relational operators (Garcia-Molina, Ullman, & Widom, 2008) such as: selection ($\sigma_{\langle condition \rangle}$), projection ($\pi_{\langle attributes \rangle}$), intersection ($\bigcap$), union ($\bigcup$), set difference ($\setminus$), join ($\bowtie_{\langle condition \rangle}$) and others.

When processing a query, many steps can be executed by the RDBMS (Garcia-Molina et al., 2008; Elmasri & Navathe, 2010) until the delivery of a result: (i) scanning/parsing/validation, (ii) query optimization, (iii) query execution and (iv) result. The query optimization step, focus of this work, has a very important optimization task, which is ordering the relational operations of the query. Specifically in the case of the join operations, the most time-consuming operation in query processing (Elmasri & Navathe, 2010), the optimization task can be viewed as a combinatorial optimization problem commonly known as join ordering problem. The problem has similarities to the Traveling Salesman Problem (TSP) and according to Ibaraki and Kameda (1984), it belongs to the NP-Complete class. It is worth noting that after the optimization phase, the executor component receives a plan with all the instructions to its execution. Execution plans with relations ordered and accessed in a way that can cause high I/O and CPU cycles (high cost solutions) will impact directly in the query response time and affect the entire system. Besides, some costly plans can make the query execution impractical, because of their high execution times. Therefore, the use of techniques capable of finding good solutions in lower processing time is extremely important in a RDBMS. For instance, in one case reported in our experiments (Section 4) the proposed optimizer (Section 3.2) was capable to find a solution with a lower estimated cost, which allowed the plan to be executed almost 23% faster than the solution provided by the official optimizer used in H2 in the same experiment.

In this paper we propose an approach to query optimization that can be classified as a non-exhaustive one. We describe an evolutionary multi-agent system (EMAS) (see Section 3.1) for join ordering optimization running in the core of a real RDBMS named H2[1] in a non-distributed environment. The main feature of the

* Corresponding author. Tel.: +55 (31) 3409 3419.
E-mail addresses: fred@nti.ufop.br (F.A.C.A. Gonçalves), fredericoguimaraes@ufmg.br (F.G. Guimarães), marcone@iceb.ufop.br (M.J.F. Souza).

[1] http://www.h2database.com/.

http://dx.doi.org/10.1016/j.eswa.2014.05.005
0957-4174/

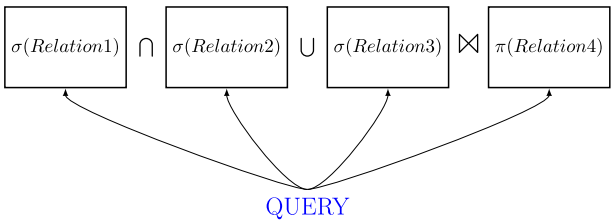

Fig. 1. Example query.

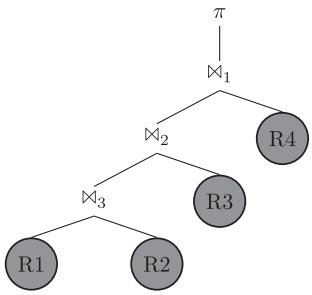

Fig. 2. Left-deep tree.

algorithm is having a team of intelligent agents working together in a cooperative or competitive way to achieve the solution of the problem. The agents of the system are able to interact and evolve in parallel. We extend initial ideas presented in Gonçalves, Guimarães, and Souza (2013) by including the following contributions: improvements in the algorithm operators; parallel implementation of the local search heuristics; evaluation of the parameters of the algorithm in the calibration phase; extended experiments with more realistic data; and comparison of the proposed algorithm with the official query planner in H2 and other DBMS, namely HSQLDB, Derby and PostgreSQL.

We highlight as the main contribution of this paper, the development of a technique not yet explored in the join ordering optimization field. Such method solves the join ordering problem in a parallel way, and as will be reviewed in Section 2, most of the proposed algorithms in the literature process the related problem sequentially. Still regarding the proposed algorithm, a new crossover method can be highlighted. The experiment design is validated with a real Database Management System, and consequently, a real cost model. Finally, we emphasize a more realistic evaluation methodology of the algorithms.

The paper is organized as follows. Section 2 discusses in more detail the query optimization problem and some methodologies applied to solve it. The proposed optimizer is detailed in Section 3. The evaluation methodology is introduced in Section 4. The computational experiments and the conclusions/future work are presented in Sections 5 and 6, respectively.

2. Query optimization problem

The optimization problem of this work consists in defining the best order of execution of the join operations among the relations specified in the query to be processed. According to Ioannidis (1996), the solution space can be divided into two modules: the algebraic space and the space of structures and methods. The algebraic space is the focus of the discussion in this section, because refers to the execution order of the relational operations considered. The space of structures and methods, on the other hand, is related to the available methods for relational operators in the RDBMS (more information about these methods can be found in Garcia-Molina et al. (2008) and Elmasri & Navathe (2010)).

The final result of the optimizer is a plan with all the necessary instructions to the query execution. Besides the operations order, a query can be represented by many different shapes. Assume that, for example, a join operator for 4 relations (*multi-way join*) is available and then a query with 4 relations could be expressed by only one join operation. However, in practice, the join operation is binary (*two-way join*) because the combinations for multi-way joins grow very rapidly (Elmasri & Navathe, 2010) and there are mature and proven efficient implementations for binary join in the literature. A representation commonly used by RDBMS is called left-deep-tree (see Fig. 2). This representation has only relations at the leaves and the internal nodes are relational operations. Due to this representation, the join operation is treated as binary (join two tables only). Even with these restrictions, the number of possible solutions remains high – for a query with $N+1$ relations the number of solutions is given by $\binom{2N}{N}N!$. Further information about the join ordering problem can be found in Ibaraki and Kameda (1984), Swami and Gupta (1988), Ioannidis (1996) and Steinbrunn, Moerkotte, and Kemper (1997).

It is worth noting that the cost of a solution is not given necessarily by the actual cost of executing the query, but instead, can be given by a cost function F that estimates the real cost of the solution. The cost estimation can use many metrics, for instance: I/O, CPU and Network. The optimizer relies on the RDBMS for the task of estimating the cost of the solution. Therefore, in the optimization process, this function is abstracted and treated as a black box, just receiving a candidate solution and returning its (estimated) cost. Information about the cost model of relational operations are provided by Garcia-Molina et al. (2008) and Elmasri and Navathe (2010).

Listing 1 presents a practical example with a query that returns all marks from computer science students.

Two join operations can be extracted from the previous query: $J_1 = \{student \bowtie marks\}$ and $J_2 = \{student \bowtie dept\}$. Besides, two valid solutions can be identified in this simple example: $S_1 = \{J_1, J_2\}$ and $S_2 = \{J_2, J_1\}$, one with a lower cost than the other. In practice, the problem can have a huge combination of possible solutions, preventing the use of exact methods or exhaustive approaches. Nonetheless, non-exhaustive algorithms fit well in these situations.

Given the complexity of the problem, several studies were presented for non-distributed environments along the years since the early days of relational databases in the 1970s (Codd, 1970). The seminal work is presented by Selinger, Astrahan, Chamberlin, Lorie, and Price (1979), advancing an exhaustive method based on dynamic programming (*DP*) with a time complexity $O(N!)$, where N stands for the number of relations in the query. Ibaraki and Kameda (1984) presented two algorithms, named *A* and *B*, with a time complexity $O(N^3)$ and $O(N^2 \log N)$, respectively. An extension of Algorithm *B* named *KBZ* algorithm with time complexity $O(N^2)$ is presented by Krishnamurthy, Boral, and Zaniolo (1986). A simulated annealing (*SA*) version for the current problem is presented in Ioannidis and Wong (1987). Many methods are compared in Swami and Gupta (1988): Perturbation Walk, Quasi-random Sampling, Iterated Improvement (*II*), Sequence Heuristic and SA. Additionally, in Swami and Gupta (1988), the authors created a new evaluation methodology to check the heuristics,

```
SELECT name, disc, mark FROM student,
    marks, dept WHERE id_student=id_marks
    AND dpt_student=id_dept AND id_dept='
    DECOM'
```

Listing 1. Example – query SQL.