8-15-1997

A Genetic Programming Approach for Distributed Queries

Karen S.K Cheung
skare@msmail.is.cphk.hk

Nabil Kamel
iskame@cityu.edu.hk

A Genetic Programming Approach for Distributed Queries

Karen S.K. Cheung
E-mail: iskare@msmail.is.cphk.hk
Tel: (852) 2788-8476

Nabil Kamel
E-mail: iskame@cityu.edu.hk
Tel: (852) 2788-8697

Abstract With the emergence of relatively inexpensive and advanced communication technology, Distributed Database Management Systems (DDBMS) have become an integral part of many computer applications. Efficient query processing is one of the most important issues in distributed database systems. In a distributed environment, it is common that queries extract data from different sites. It is important to limit the amount of data transfer across different sites. Semijoin is a way to reduce the cost of expensive joins between various sites. A key issue in query optimization based on semijoin reduction is to find a good sequence of semijoins that reduce the relations referenced in a given query before the joins are performed. This paper proposes a new approach, based on Genetic Programming (GP), to improve the process of database query in Distributed Database Systems. A longer version of this paper is available.

1 Introduction

The objectives of query optimization is to ensure that either the total cost or the total response time for a query are minimized. When a query is presented to the system, it is essential to use the best possible way to search for the answer and respond to the user. As the difference in the computation cost between a good and a bad search strategy may be huge, the use of an efficient method for searching the required data is of utmost importance. Among the various methods that have been proposed for distributed query optimization, those based on semijoin to reduce the referenced relations have received considerable attention. A key issue in query optimization based on semijoin reduction is to find a good sequence of semijoins that reduce the relations referenced in a given query before the joins are performed. This paper proposes a new approach, based on Genetic Programming (GP), to improve the process of database query in Distributed Database Systems. Genetic Programming, which contains a probabilistic element, mimics Darwinian evolution to tackle search problems. The paper is organized as follows. Section 2 describes the search problem in distributed database. Section 3 gives a brief review on the prior work.

Section 4 introduces Genetic Programming (GP) and discusses how GP can be applied to perform the search in distributed query optimization. Computer simulation and a brief conclusion are included in Section 5.

2 The Search Problem

Let us use an example to illustrate. There are two relations located at Site 1 and Site 2 respectively, as shown in Figure 1. A user at Site 1 submits a query to prepare a class list of each course. It is apparent that a join of the two relations is required. Before performing an expensive join operation, *semijoin* () is used to reduce the size of a relation that needs to be transmitted.

STUDENT	Site 1
Student_ID	Student_Name
95111234	Peter Wong
95122345	Nancy Ma
95133456	May Tam
95144567	Christina Ng
95155678	Desmond Kwok
95166789	Leslie Chan
95177890	Candy Lee
95188901	Maria Lam
95199012	Benny Tsui
95100123	Ronald Li

REGISTRATION	Site 2
Student_ID	Course_ID
95111234	IS3001
95144567	IS3002
95144567	IS3006
95188901	IS3003
95199012	IS3001

Figure 1. STUDENT and REGISTRATION relations

The following are the steps of performing semijoin operation:

1. Project REGISTRATION on Student_ID and get a temporary result, A.

A	Student_ID
	95111234
	95144567
	95188901
	95199012

2. Transmit A to Site 1, perform a join and get the following result, B.

B	Student_ID	Student_Name
	95111234	Peter Wong
	95144567	Christina Ng
	95188901	Maria Lam
	95199012	Benny Tsui

The semijoin is a reduction operator; STUDENT REGISTRATION can be read as the reduction of STUDENT by REGISTRATION. Note that the semijoin operation is not symmetric, i.e., the result will differ if we reverse the order. In addition, different sequences of semijoins have different cost. It is not difficult for us to use semijoin to join relations between two sites. However, suppose we have a query which involves joining more than, say, eight sites, the situation will then become more complicated because the sequence of semijoins will greatly affect the communication costs.

3 Prior Work

Many algorithms have been proposed to find a good semijoin sequence for different type of tree queries. One of earliest research approaches in distributed query processing was developed by Wong [5]. This algorithm was further enhanced and implemented in the SDD-1 system. The SDD-1 algorithm addressed simple queries and used a hill-climbing approach to find as many semijoins as possible. However, due to the use of hill-climbing approach, the SDD-1 algorithm can only obtain a local optimum solution. Apers et al. introduced two query optimization algorithms, PARALLEL and SERIAL, that derived optimal solution for the class of simple queries. These algorithms were further improved in [1], which was called GENERAL algorithm to address general queries. Unlike simple queries, general queries are queries which involve relations containing multiple joining attributes. In GENERAL algorithm, a separate semijoin program is constructed for each relation. Though GENERAL is considered to be efficient, it can derive only quasi-optimal solution. Regarding chain queries, Chiu in [2] proposed to use a dynamic programming approach to find an optimal semijoin sequence. The dynamic programming approach can guarantee to get an optimal solution. However, since this method needs to compare all the semijoin sequence programs, it results in a high computation effort. Yu et al. in [7] used a dynamic programming approach to consider tree queries. Dynamic programming was also used in [4] for which joins and semijoins are included. Yoo and Lafortune [6] presented a new method, A* algorithm, to navigate the space of the semijoin sequences to return an optimal solution. A* performs efficiently in average less than five percent of the search space is searched before an optimal solution is found. However, five percent of a search space in complex queries still represents a substantial computational effort.

4 Application of Genetic Programming (GP) to Optimize the Search

According to the evolution theory of Darwin, biological structures that are more successful in grappling with their environment survive and reproduce at a higher rate. This principle is used in Evolutionary Algorithms, for which Genetic Algorithms and Genetic Programming are two of its major classes. Genetic Algorithm (GA) operates on populations of strings of binary digits, called chromosomes, with the strings encoded to represent individual chromosomes. Three processes, namely, *selection*, *crossover*, and *mutation* are applied to the string populations to create new ones. Reproduction takes place on a single chromosome and allows fitter chromosomes to reproduce more. Crossover, the sexual recombination, creates different chromosomes in the offspring by combining materials from the chromosomes of the two parents. Mutation causes the

chromosomes of biological offspring to be different from those of their biological parents. Genetic Programming (GP), proposed by John Koza, is an extension of Genetic Algorithm [3]. Although both GP and GA are mimicking Darwinian evolution for tackling search problems, they differ in the problem and solution representations in actual implementation while GA encodes the problem with flat string for further manipulation, GP adopts a hierarchical tree-structured representation.

The structures undergoing adaptation in GP are a population of individual computer programs from the search space. These computer programs are generally hierarchically organized and of dynamically varying size and shape. In our case, an individual consists of a specific sequence of semijoin operations. The individuals in the population are any possible sequences of semijoin operations. In our work, we maintain the search space, i.e., population size to 20, which are chosen from all possible different sequences of semijoin operations.

There are five preparation steps in order to apply GP, in our case, to find a semijoin sequence [3].

1. **Determining the set of terminals**

The first step is to identify a set of terminals that is used in the individual computer programs in the population. The terminals are the inputs, i.e., information that will be processed, for a problem. In our case, the terminals are the relations locating at different sites.

2. **Determining the set of functions**

The second step is to identify a set of functions that will be applied to the set of terminals, i.e., the inputs. The functions, in our case, are the semijoin operations.

3. **Determining the fitness measure**

The third step is to evaluate how good each solution of the problem at hand is. Fitness can be measured in different ways. The most common types of fitness used are the error measure and payoff value [3]. We consider the communication costs to be the fitness measure in our work.

4. **Determining the parameters and variables for monitoring the run**

The fourth step involves selecting the values of parameters to control the runs. GP is controlled by 19 parameters. *Population size* and *maximum number of generations* to be run are the two major parameters that should be specified, while the 17 minor parameters, which are used to control the process, can be neglected in most applications, including in this paper [3]. The population size is chosen according to the complexity of a problem. Generally, the larger the population, the better. In our work, the population size is set according to the problem size. Thus problems with more sites will offer a larger number

of semijoin sequences, and the number of generations is limited by the amount of time available for the problem.

5. **Determining the method of designating a result and the criterion for terminating a run**

The final step is to specify the criterion for designating a result and for terminating a run. The method of result designation assigns the best individual that ever appeared in any generation of the population (i.e., the best-so-far individual) as the result of a run of genetic programming. The best-so-far individual is then reported as the result of the entire run when the run eventually terminates according to the termination criterion. The run of genetic programming terminates when the termination criterion is satisfied. Usually, we terminate a run either when the prespecified number of generations has been reached, or when the success predicate has been satisfied. In our work, the success predicate is satisfied if the best solution obtained meets a certain prespecified threshold.

6. **Computer Simulation and Conclusion**

Computer simulation is conducted to demonstrate the feasibility of applying genetic programming to find a semijoin sequence that solves the query at lowest cost. To address this problem, we have implemented a prototype of our genetic programming approach in Common Lisp and perform a number of simulations. We will then compare the quality of the solutions proposed by the GP approach with the other existing approaches.

References

1. Apers, P.M.G., Hevner, A.R. and Yao, S.B. Optimization Algorithms for Distributed Queries. *IEEE Transactions on Software Engineering*. January 1983, Vol. 9, No. 1, pp. 57-68.

2. Chiu, D. M., Bernstein, P. A. and Ho, Y.C. Optimizing Chain Queries in a Dist. Database System. *SIAM Journal on Computing*, February 1984, Vol. 13, No. 1, pp. 116-134.

3. Koza, J.R. *Genetic Programming: On the Programming of Computer by Means of Natural Selection*. (MA: The MIT Press, 1992).

4. Lafortune, S. and Wong, E. A State Transition Model for Distributed Query Processing. *ACM Transactions on Database Systems*, September 1986, Vol. 11, No. 3, pp. 294-322.

5. Wong, E. Retrieving dispersed data from SDD-1: A system for distributed databases. *Proceedings of 2nd Berkeley Workshop Distributed Data Management and Computing Networks*, May 1977, pp. 217-235.

6. Yoo, H and Lafortune, S. An Intelligent Search Method for Query Optimization by Semijoin. *IEEE Transactions on Knowledge and Data Engineering*, June 1989, Vol. 1, No. 2, pp. 226-237.

7. Yu, C.T., Ozsoyoglu, Z.M. and Lam, K. Optimization of Distributed Tree Queries. *Journal of Computer and System Sciences*, 1984, Vol. 29, pp. 409-445.