Query Execution Algorithm in Web Environment with Limited Availability of Statistics

Juliusz Jezierski and Tadeusz Morzy

Poznan University of Technology
Piotrowo 3a, 60-965 Poznan, Poland
{jjezierski, tmorzy}@cs.put.poznan.pl

Abstract. Traditional static cost-based query optimization approach uses data statistics to evaluate costs of potential query execution plans for a given query. Unfortunately, this approach cannot be directly applied to Web environment due to limited availability of statistics and unpredictable delays in access to data sources. To cope with lack or limited availability of statistics we propose a novel competitive query execution strategy. The basic idea is to initiate simultaneously several equivalent query execution plans and measure dynamically their progress. Processing of the most promising plan is continued, whereas processing of remaining plans is stopped. We also present in the paper results of performance evaluation of the proposed strategy.

1 Introduction

There is increasing interest in query optimization and execution strategies for Web environment that can cope with two specific properties of this environment: lack or limited availability of data statistics and unpredictable delays in access to data sources. Typically, in Web environment query processing parameters may change significantly over time or they may be simply not available to query engines. Web sites that disseminate data in Web environment in the form of files, dynamically generated documents and data streams usually do not allow access to internal data statistics. The second specific property of Web environment is unexpected delay phenomenon in access to external data sources. Such delays may cause significant increase of system response time. They appear due to variable load of network devices resulting from a varying activity of users, and also, due to breakdowns. As a result, traditional static optimization and execution techniques cannot be directly applied to Web environment.

In the paper, we present the novel competition strategy of query execution in Web environment that solves or reduces limitations of previous solutions (e.g. [1,2,3,4]). Our approach consists in simultaneous execution of a set of alternative query execution plans for a given query. The system monitors execution of these plans, and the most attractive plans are promoted, while execution of the most expensive plans is canceled. Final query result is delivered to the user by the

plan that has won the competition according to rules defined by the strategy implementation.

2 Competition Strategy of Query Execution

In traditional database systems query specified by user is transferred to the query optimizer, which chooses optimal query execution plan (QEP) during query compilation. The query optimization process depends on: (1) a cost function used to evaluate the cost of a query, (2) the search space of all possible QEPs for a given query, and (3) a search strategy used to penetrate the search space of QEPs. Final QEP generated by the query optimizer is static and it does not change during its execution.

Our query execution strategy is based on the idea that the query optimization process should be continuous and interactive, which means that the search space of QEPs should be also analyzed during query execution. Query optimizer improves the initial QEP by taking into account data statistics gathered during query execution. Formally, the competition strategy of query execution can be defined as a triple: $CSQE = \{PGR, CC, FR\}$, where: PGR - rules of plan generation, CC - competition criteria, FR - feedback rules.

PGR denotes a set of rules used to generate QEPs participating in the competition. The important issue is the proper selection of initial QEPs. On the one hand, in order to reduce the overhead related to simultaneous processing of many QEPs, it is necessary to restrict the number of initiated plans. On the other hand, if the number of initiated plans is too small, then, the adaptation to changing conditions of runtime environment is automatically restricted. CC denotes competition criterion used to evaluate attractiveness of different QEPs (i.e. response time, evaluation cost). Proper definition of CC allows to limit overhead related to simultaneous processing of many plans by pruning ineffective QEPs. FR denotes a set of rules that control the competition process (e.g., start new plans). FR allows to adapt query execution process accordingly to changes of runtime environment parameters, e.g., delays in access to data sources.

The competition strategy has an "open" character and it can be implemented in many different ways. To illustrate the approach, we implemented greedy algorithm (abbreviated as GA), which implements our strategy. We assume no availability of statistics. All necessary data are dynamically established or estimated during query execution.

3 Experimental Evaluation

To demonstrate the practical relevance and universality of our strategy, we compare it with simple "brute force" strategy (abbreviated as BFS), which generate all possible query execution plans for a given query, and to comparison we have taken into account the average value of their results. We considered two basic performance evaluation criteria: system response time and utilized CPU time. Two main goals of experiments were: analysis of the impact of transfer rates

and initial delays on performance evaluation criteria. Arbitrary, three transfer rates were assumed for experiments: 20Kbytes/s (slow), 200Kbytes/s (normal) and 2Mbytes/s (fast). The algorithm was implemented in Java 1.4 and experiments were computed on PC Intel 1000Mhz, 512MB RAM under control of MS Windows 2000. We analyzed the cycle SQL query Q1 given below:

```
select * from A, B, C where A.b=B.a and B.c=C.b (Q1)
```

The aim of the first series of experiments was comparison of our strategy with BFS in case where costs of potential query execution plans significantly differ from each other. Thus, we generated data with large range of values of join selectivity coefficients: $\text{sel}(A \bowtie B) = 2 \cdot 10^{-3}$, $\text{sel}(A \bowtie C) = 2 \cdot 10^{-4}$ and $\text{sel}(B \bowtie C) = 2 \cdot 10^{-5}$. The volumes of the sources we assumed as follows: A–800KB($5 \cdot 10^3$ tuples), B–1500KB(10^4 tuples), C–2000KB($2 \cdot 10^4$ tuples).

Fig. 1. Elapsed time of Q1 execution versus initial delay of source C

Fig. 2. Utilized CPU time of Q1 execution versus initial delay of source C

Fig. 3. QEP1 of Q1

Fig. 4. QEP2 of Q1

Fig. 5. QEP3 of Q1

Figure 1 presents the system response time for the query Q1 versus initial delay of the source C and different values of the transfer rate. The cost ranking of QEP for Q1 is the following (from cheapest to the most expensive): 1. QEP1, 2. QEP2, 3. QEP3 (Fig. 3, 4, and 5). Most attractive plans require access to source C in the first step. For 2Mbytes/s transfer rate, we observe that the algorithm switches from QEP1 to QEP2 for 8 seconds delay. This switch appears when delay in access to source C is so long that execution of the subplan 1 of QEP3 is finished before the algorithm collects statistically reliable samples from executions of subplan 1 of QEP2 and subplan 1 of QEP3. From the figure follows, that the GA outperforms BFS strategy before switch from QEP1 to QEP3 occurred, and longer response time after the switch. For 200Kbytes/s transfer rate, response time provided by the algorithm is always better than

response time provided by BFS. For 20Kbytes/s transfer rate, we do not observe any switch. QEP1 always wins the competition. Moreover, for the whole range of delays for the source C, the GA outperforms BFS. However, for readability of Fig.1, we omit in the figure the results for 20Kbytes/s transfer rate.

Figure 2 presents utilized CPU time (i.e. overhead) versus initial delay of source C and different values of the transfer rate. The overhead depends on delay in access to most attractive sources. Increasing delay in data transfer from the source C delays a moment of competition termination, and, thus, extends also execution time of QEPs, belonging to a competition group, and CPU consumption. The largest overhead is observed for 2Mbytes/s transfer rate, while the smallest one is observed for 20Kbytes/s transfer rate. This phenomenon can be explained as follows: for a given delay of source C, in case of higher transfer rate, a large part of unattractive subplans (i.e. subplan 1 of QEP3) will be executed until the competition process stops their processing. In case of lower transfer rate, unattractive subplans consume less CPU since the algorithm cancels their processing "earlier". Notice that for 2Mbytes/s and 200Kbytes/s transfer rates, if delay of the source C is rather small, i.e. does not exceed several seconds, GA is cheaper than the BFS. For 20kbytes/s transfer rate, GA is several times cheaper than the BFS.

We also performed a series of experiments, which tested different transfer rates for other data sources. If attractive data sources transfer data with higher rate than other sources, then the competition overhead decreases. We also tested the performance of the GA with the 5-way join query (Q2). In this case, the GA produced query result few times faster than the BFS. It can be explained as follows: the query Q2 is more complex than Q1, and, therefore, a set of QEPs for Q2 is much larger than that of Q1. Therefore, an average cost of these plans taken into account in the comparison, is relatively large, whereas the GA generated nearly optimal QEP.

4 Summary

In this paper, we proposed novel strategy of dynamic query optimization and execution in Web environment, which cope with limited availability of data statistics and unexpected delay in access to data sources. We evaluated our strategy by a set of experiments for different transfer rates and different delay scenario, and proved its feasibility. As the experiments show, our strategy is especially appropriate for small and medium transfer rates (20Kbytes/s and 200Kbytes/s). The strategy is efficient also for large transfer rate (2Mbytes/s) and relatively small delays (several seconds) in access to attractive sources. The algorithm prefers bushy QEPs, which, when compared to linear QEPs produced by traditional static cost-based optimization algorithms provide usually better response times.

References

1. Urhan, T., Franklin, M.J., Amsaleg, L.: Cost based query scrambling for initial delays. In: Proc. ACM SIGMOD Conf., June 2-4, 1998, Seattle, USA, 130–141
2. Kabra, N., DeWitt, D.J.: Efficient mid-query re-optimization of sub-optimal query execution plans. In: Proc. ACM SIGMOD Conf., June 2-4, 1998, Seattle, USA, ACM Press (1998) 106–117
3. Avnur, R., Hellerstein, J.M.: Eddies: Continuously adaptive query processing. In: Proc. ACM SIGMOD Conf., May 16-18, 2000, Dallas, USA, 261–272
4. Viglas, S., Naughton, J.F., Burger, J.: Maximizing the output rate of multi-way join queries over streaming information sources. In: Proc. VLDB Conf., September 9-12, 2003, Berlin, Germany, Morgan Kaufmann (2003) 285–296