Rule Management for Semantic Query Optimisation in a Distributed Parallel Environment

Barry G.T. Lowden
Department of Computer Science, The University of Essex,
Wivenhoe Park, Colchester CO4 3SQ, Essex,
United Kingdom
and
Jerome Robinson
Department of Computer Science, The University of Essex,
Wivenhoe Park, Colchester CO4 3SQ, Essex,
United Kingdom

ABSTRACT

The importance of query optimisation has led to active research in this area over a number of years and the development of optimisation components as a standard feature of large Database Management Systems. More recent work is concerned with semantic query optimisation (SQO) which is guided by derived knowledge (rules), about the data itself, using techniques associated with data mining. Fast, efficient rule generation and maintenance is crucial to achieving high optimiser performance. At the same time it is recognized that most available computing power, in a networked client-server system, resides in the client workstations which are often under-utilized and that this resource is expanding rapidly with each successive new generation of workstation. In this paper we show how this spare capacity may be used to improve optimiser performance by distributing the rule management processes in a parallel virtual machine environment.

Keywords: Query Optimisation, PVM.

1. INTRODUCTION

Conventional query optimisation techniques tend to rely on a combination of heuristics and cost estimation procedures. The user query is normally expressed in some formal intermediate language based on either the relational algebra or calculus and the resulting query expression transformed, using a set of heuristic rules, into one which is more efficient to execute. This is achieved by ordering the sequence of operations to be performed in such a way as to minimize the amount of data processed in order to retrieve the result set. This may involve restructuring an algebraic tree or partitioning the query into a number of single variable sub-queries, which can be executed in a cost-effective sequence. In each case, however, the optimisation procedure is heavily dependent on accurate statistics being available relating to the content and structure of the database and more importantly the optimised query is simply a re-ordered version of the original.

Semantic query optimisation differs markedly from the conventional approach in that the original query is transformed into an alternative which syntactically may be totally different, from the original, yet when executed retrieves the same answer set. This is achieved by using semantic knowledge about the database to guide the transformation process. This knowledge is in the form of semantic rules that hold for the database state being queried. As an example consider an employee relation for which the following rules apply:

(i) department = ‘computing’ ⇒ benefit = ‘car’
(ii) benefit = ‘car’ ∧ salary > 30K ⇒ status = ‘executive’

and the relation is indexed on status.

A query of the form ‘retrieve all staff in computing who earn more than 35K’ may thus be transformed, using the above rules, into ‘retrieve all executives who work in computing and earn more than 35K’. If executives represent only 5% of the workforce then it may be seen that this semantically equivalent query will execute in around 5% of the time taken by the original since only ‘executive’ tuples need be checked for the two original conditions.
In general, given a query Q with constraints (C₁, …, Cₖ), query transformation may be achieved by repeated use of the following reformulation operators:

(i) Addition – given a rule x $\rightarrow$ y, if a subset of constraints in Q implies x, then we can add constraint y to Q.

(ii) Deletion – given a rule x $\rightarrow$ y, if a subset of constraints in Q implies x, and y implies Cᵢ, then we can delete Cᵢ from Q.

(iii) Refutation – given a rule x $\rightarrow$ y, if a subset of constraints in Q implies x, and y implies $\neg$Cᵢ, then we can assert that Q will return NIL.

Early work in semantic query optimisation was limited to user supplied knowledge such as integrity constraints and data dependencies [3]. Rules of this form are applicable to any state of the database but are limited in their range and usefulness. Later work has focussed on the automatic generation of association rules derived from the data itself [14, 6] using techniques closely related to data mining [15].

Whilst this latter approach opens up a rich source of additional semantic knowledge, a major outstanding weakness has been that the optimisation cost is a (potentially exponential) function of the number of rules employed and so can prove significant with respect to the overall execution time. Ensuring that the rule base is restricted to a small up-to-date set of highly effective rules is essential, therefore, to maximizing optimiser performance. This requires a fast rule management process including derivation and selection, which responds quickly to changes in both data and query patterns.

In this paper we describe this process and show how performance time may be significantly reduced by distribution across a typically under-utilized networked client server system. Section 2 provides background to the construction of semantically equivalent queries and in section 3 the rule maintenance algorithm is explained together with its implementation using a parallel virtual machine. Experimental results are then presented in section 4 with, finally, the conclusions in section 5.

2. ALTERNATIVE QUERY GENERATION

Given a user query Q, expressed in a language such as SQL, and a set of association rules R, of the form A $\rightarrow$ B, there are two main techniques for generating alternative queries using the reformulation operators defined above. Both strategies involve the repeated addition and deletion of query conditions as the reformulation algorithm processes through the rule set and are based on the well known methods of depth first and best first search.

**Depth First**

This strategy yields an exhaustive set of all possible alternative queries and is illustrated in Figure 1. The original query Q is transformed into queries Q₁ … Qₙ by successive application of all rule combinations using the reformulation operators. The projected cost of each new alternative is computed using a standard cost estimation function [4] and the cheapest selected as optimum. The cost estimator may also include a conventional optimisation component.

![Figure 1. Depth First Query Generation](image)

The problem with this search method is that as the number of matching rules increases the corresponding number of alternative queries generated rises exponentially. Thus n matching rules will yield $(2^n - 1)$ distinct alternatives and so this strategy becomes prohibitively expensive for all but the smallest rule sets. Even with the most efficient cost estimation procedures, it may be seen that the time taken to find the optimal alternative can easily exceed that of executing the original query. A more practical approach is therefore needed and the one normally adopted is that of best first query generation.

**Best First**

The essence of this strategy is to allocate a given amount of resource (time) in which to find a cheaper alternative to the original query. This allocation will usually be some function of the
estimated time to execute the original query \( f(Q) \). The generation process is illustrated in Figure 2.

\[ Q \Rightarrow Q_1 \Rightarrow Q_2 \Rightarrow Q_{2k} \Rightarrow Q_{2n} \Rightarrow Q_n \]

**Figure 2. Best First Query Generation**

Q is reformulated with respect to each matching rule in the rule set yielding alternative queries, \( Q_1 \) ... \( Q_n \). The projected cost of each alternative is evaluated and the cheapest (\( Q_2 \) in the illustration) is further reformulated and then eliminated from the search. The process continues with repeated selection, reformulation and elimination until the resource allocation runs out. At this point the cheapest alternative is returned for execution.

The strategy may not necessarily identify the optimum alternative and is heavily dependent on the order in which the rules are applied since this will determine which subset of the full range of alternatives is generated. Experiments associated with [IEEE] have shown that, where the rule order is random and the resource allocation is 10% of the original query costs, then best first search leads to a 35% average saving in query execution time as opposed to a potential saving, on average, of 43% which could be achieved by always selecting the optimal alternative. Significant improvements can be achieved however, using best first search, by limiting the rule set to a small number of high quality rules ranked according to their effectiveness. To accomplish this, our rule management system incorporates rule derivation, selection and maintenance components.

Rule Management

As already explained the query-driven rule selection and maintenance process, used in our research, is based on an inductive learning method which is described more fully in [7]. The resource intensive component of the procedure is called Continuous Data Analysis (CDA) which involves scanning 2-column subsets of a relation to determine relationships between the associated attributes.
Suppose we have a 2-column array, as illustrated in Figure 4, with columns denoted A and B. The CDA algorithm first sorts column A to ascending order and then attempts to identify rules of the form \( X \rightarrow Y \) where the antecedent \( X \) and consequent \( Y \) are value ranges drawn from columns A and B respectively. The algorithm also determines the end of range values for the consequent column and the empty ranges between values in the antecedent column. In the example given, these are (315-970) and (2-6, 9-9, 12-12) respectively. These values may be used for rule subsumption, described below, to discard redundant rules.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>758</td>
</tr>
<tr>
<td>≥ 7</td>
<td>315</td>
</tr>
<tr>
<td>≥ 8</td>
<td>756</td>
</tr>
<tr>
<td>10</td>
<td>585</td>
</tr>
<tr>
<td>10</td>
<td>970</td>
</tr>
<tr>
<td>11</td>
<td>543</td>
</tr>
<tr>
<td>13</td>
<td>605</td>
</tr>
<tr>
<td>14</td>
<td>522</td>
</tr>
</tbody>
</table>

Figure 3. Using CDA to discover rules.

The rule discovery process uses two pointers denoted the lower and upper pointers respectively. Initially both pointers point to the first row of the array.

The lower pointer is moved downwards, one row at a time, until it reaches the last instance of a value different from the previous value (the same value may be repeated for several rows). At the same time the corresponding value of the other column is checked and the lowest and highest values recorded. The ranges for both antecedent and consequent are thus easily computed yielding a potential rule, which may subsequently be added to the rule set. For example the first rule to be identified from the data shown, would be (1-7) \( \rightarrow \) (315-758).

As the process continues, the decision to add or discard rules depends on one of the following:

(a) Rules should have the range of their antecedents as wide as possible and consequents as narrow as possible to enable easier rule matching. One way of ensuring this is through rule subsumption [10,11]. If the antecedent range contains (i.e. completely covers) the antecedent range of the previous rule but has the same consequent range, the rule is added and the previous one deleted (subsumed). The new rule is more useful because more conditions in a query can be matched through the wider antecedent range.

(b) If the consequent range corresponds to the maximum and minimum values of that column, the rule is deleted since it offers no additional information.

The above procedure is in line with the general idea adopted by the more “traditional” KDD/DM approaches of pruning uninteresting rules while the rules are being located [9].

Once the decision is reached, the lower pointer is moved downwards again, checking for potential rules and adding or deleting rules as appropriate. The process continues with the upper pointer moving downwards progressively to the first instance of the next different value and the lower pointer moving to the last instance of the next value different from that now pointed to by the upper pointer. The algorithm terminates when the upper pointer reaches the last row.

4. IMPLEMENTATION RESULTS

The CDA algorithm is a comprehensive method for determining effective non-redundant rule sets, however resource considerations make it unsuitable for running on a single machine, particularly when used to identify rules in large data-sets. PVM, therefore, with its software alternative for distributed computing using existing machines, is well suited for the running of programs that implement CDA. The inexpensive PVM architecture and the simple yet functional CDA algorithm point naturally to a combined system.

We now present some rule maintenance results based on a 105-tuple, 5-attribute relation, containing randomly generated 3-digit values. The master program distributes each unsorted 2-column array to the slave machines for sequencing and rule set modification, using the CDA algorithm. Experiments were repeated at hourly intervals and ran for approximately 40 minutes.

Figures 5 and 6 show the results of a typical experiment and are an accurate reflection of those of the complete series.
The network timing graph, Figure 5, gives an indication of network load, since a low send rate reflects high loading. Load levels will, in turn, affect PVM performance.

In the set of experiments undertaken, tasks are spawned once and re-used until they are no longer required. Workload is distributed equally to each of these tasks early on during runtime. Since there is no need to spawn and delete tasks repeatedly for each round of experiments involving different message size, significant overheads are thus avoided.

Figure 6 shows the performance of two PVM programs which both call instances of the same slave program. PVMM receives results in a non-deterministic way, i.e. as soon as they are available in the buffer and regardless of their origin, whilst PVMM1 receives results on a round robin basis. In the experiments described, both follow a similar pattern, however we may conclude that if the virtual machine consists of machines of different speeds (as opposed to the situation described here, which uses similar machines of the same speeds) the disparity would be more obvious. PVMM1 should then offer better performance since the master process is not held waiting for slower hosts to return their results.

Another issue concerns the difficulty in predicting, in advance, the optimal number of hosts that will achieve the maximum improvement in performance. This is because resource availability must be balanced against network overheads and this will depend on the specific implementation. However, for very large data sets, the transmission overheads become less significant with respect to resources needed for rule maintenance and ideally the 2-column arrays should each be processed on separate slaves.

Alternatively, if this is not possible, then a reasonable estimate can be made with subsequent adjustments to achieve optimality.

It should be noted that both PVM programs display “peaks” which correspond roughly to the “peaks” displayed by the standalone since the latter also forms part of the virtual machine. Thus, factors affecting the standalone, e.g. external users, will also affect the virtual machine for the test period concerned. However, since work is distributed to other hosts the overall performance is still greatly improved.

Over the complete range of experiments undertaken, the PVM programs achieved, on average, a performance improvement in excess of 100% over that which was achieved by the standalone. This meant that an optimum rule set, up-dated as a result of a query, was returned in less than half the usual time making it substantially more effective in optimising subsequent queries.

5. CONCLUSIONS

In this paper we have described how the performance of a semantic optimiser may be improved by fast rule maintenance using a parallel virtual machine. The approach makes efficient use of spare capacity often available in a local network of workstations without affecting the throughput of other users. Whilst the technique has focussed on rule management there are potentially many other applications which could benefit from a similar implementation. As part of their current research
programme, the authors are now developing a parallel version of their query transformation algorithm.

Acknowledgement
The authors are indebted to Shiunn Liew who assisted with the experiments and result analysis.

REFERENCES


