A Strategy for Decomposing Complex Queries in a Heterogeneous DDB

S. M. Deen, R. R. Amin, M. C. Taylor

Preci Project, Department of Computing Science,
University of Aberdeen, Aberdeen, Scotland.

Abstract

In a generalised distributed database system with decentralised controls and heterogeneous and pre-existing nodes, queries can be very complex, particularly if they provide a data integration facility. We describe here an algorithm for the optimal decomposition of such queries into subqueries, taking into consideration the availability of nodal operations (some nodes may not be able to perform all operations) and other factors. This algorithm is being implemented in the PRECI* system.

In a distributed database system, an efficient query processing strategy is essential for ameliorated performance. In general there will be many possible strategies for processing a particular query, and ideally each of these should be evaluated in order to determine the best strategy. Unfortunately, however, the problem of selecting optimal strategies for complex queries is NP-complete, so it is not feasible to evaluate every strategy for such queries. Many query decomposition algorithms have therefore been designed to produce optimal or near-optimal strategies only under a set of highly restrictive assumptions that apply to a particular implementation (1, 2). The only algorithm developed for a DDB which allows heterogeneous pre-existing databases as nodes is that of the MULTIBASE project (3). They evaluate strategies in terms of data movement and local processing, disregarding response time. All final processing is done at the result node, so that there is little parallel processing. In PRECI* (4, 6) we have aimed to achieve a higher degree of parallelism while still taking account of data movement and local processing.

Recognising the NP completeness of the general problem, we propose to tackle it in two stages: optimal decomposition of a query into subqueries and the subsequent allocation of the subqueries optimally to nodes, taking into account the presence of replicated data and network characteristics. We believe that for complex queries in heterogeneous DDBs, this two-staged approach is most fruitful although it may not yield the optimal strategy. In this paper, we present only the decomposition strategy which involves determining the order of operations to be performed, applying transformations to the original query expression in order to reduce the total cost (in terms of data movement and local processing) or to reduce response time by increasing parallelism without increasing the total cost. Operations are then grouped into subqueries. For the node allocation stage, we decide where each subquery should be executed, evaluating each possible strategy according to the total cost and response time. There may be many possible strategies since, in PRECI*, operations on external data (data sent from another node) may be performed at any node which supports an appropriate interface, unlike in MULTIBASE where only the result node is used for such operations. We assume that a given node may not be able to perform all PAL operations, and this is taken into consideration in the query decomposition strategy.

PRECI* is a generalised distributed database management system supporting heterogeneous, possibly pre-existing, databases as nodes (4, 5). It also allows data replication under global supervision. Any database may join PRECI* as a node, provided it supplies a minimal relational interface, and any network could be used to link the nodes. Queries to the DDB are expressed in the PRECI algebraic language (PAL), and address a global external schema which is defined, optionally via a global conceptual schema, in terms of the nodal schemas, using PAL. A query can be represented as a parse-tree. Its mapping to the collection of nodal schemas is then done by query modification, by which each relation in the parse tree is replaced by its definition in terms of the nodal schemas. Because the nodes can be pre-existing databases, there are likely to be incompatibilities between them which must be resolved by data integration techniques (5, 7, 8). PAL contains a number of constructs specifically for data integration (more details later). When mappings between nodal and external schemas are complex, even a fairly simple query over a global external schema can become quite complex when mapped to the collection of nodal schemas.

Once the query has been expressed over the collection of nodal schemas, decomposition proceeds. Given a query expression there are two decisions to be made:

(1) What is the best decomposition of this expression into subexpressions?

(2) Can the expression be improved by transforming it to an equivalent expression?

Decomposition of an expression can be done by

```
repeat

identify a branch of the parse tree that
can be answered by a single node;
detach this branch and replace it by a
single vertex available to all nodes;
}
while (more than one subexpression is left)
```

(a branch of a query tree can be answered by a single node if each vertex is available to that node. A relation is available to those nodes at which it is stored, while an operator is available to those nodes which support that operator as part of their relational interface. Any other vertex is available to all nodes).

In determining the best set of subexpressions, we take the view that usually an expression should be broken only where necessary, with as few subqueries as possible. The fewer the subqueries, the fewer the number of intermediate results to be sent between nodes. Further, a subquery involving a large number of operations will often produce a result substantially smaller than the sum of the sizes of its constituent relations. So this strategy should produce low communication costs. For some types of expression it will also reduce local processing costs by doing processing on locally stored data, rather than on external data (data sent from another node) for which indices are not available.

However, there are situations in which an expression, which is the union of two subexpressions, is best divided even if it could be answered by a single node. This is because the result of a union operation is large in comparison to its operands, and performing a union on external data is not expensive.

Thus the breakpoints of an expression (i.e. the vertices at which it should be split into subexpressions) are the following:

(1) any vertex which is the root of a subtree whose vertices are not all available to a single node;

(2) any vertex which holds a union operator.

From a query expression, and the list of breakpoints which describe its decomposition, we then seek transformations which can improve the query expression. There are two classes of transformations to be considered:

(1) distribute a unary operation over a binary operation

(2) change the order of two adjacent unary operations.

Obviously these transformations can only be considered if they produce an equivalent query expression.

When considering transformations in class (1), we apply the following rules:

Rule 1

Distribute a unary operation over a binary operation if the binary operation is a breakpoint of the expression and the unary operation tends to reduce the size of its operand.

Rule 2

Distribute a unary operation over a binary operation if the binary operation is a breakpoint of the expression; the unary operation does not significantly increase the size of its operand and is best done on locally stored data; and the operand is locally stored (i.e. no descendant of the binary operation in the expression tree is a breakpoint).

Transformations in class (1) allow the unary operation to be applied before data transmission. When Rule 1 applies, this means that the transformation has reduced the volume of data transmitted. Selection and projection are examples of unary operations which fall into this category. When Rule 2 applies, processing time can be reduced by performing the unary operation on locally stored data, where indices are available, and sometimes response time can be improved by increasing parallelism.

When considering transformations in class (2), we apply Rules 3 and 4:

Rule 3

> Change the order of two adjacent unary operations if the first operation (i.e. the first to be evaluated) is a breakpoint and the second operation reduces the size of its operand.

Rule 4

> Change the order of two adjacent unary operations if the first is an expensive operation and the second reduces the size of its operand.

Applying Rule 3 allows an operation to be performed which reduces the volume of data to be transmitted. Applying Rule 4 gives the expensive operation a smaller operand on which to work, thereby reducing processing costs. For example, if a node is unable to perform some unary operation on its data, it will have to send the data to another node for processing. In such cases a transformation under Rule 3, which causes a selection or projection to be done first, will be profitable. Similarly, a transformation under Rule 4 can cause a selection or projection to be done before a complex operation.

To illustrate the application of these principles to query decomposition in PRECI*, we must first describe some of the constructs of PAL. PAL is based on the relational algebra but permits nested selection and includes some special constructs which are particularly useful for data integration. The use of these constructs is described in (5).

The "alteration" command has two forms. It can either extend a relation by an extra attribute, or it can define an attribute to replace one of a relation's existing attributes. The crux of the syntax is:

$$R : EXT(C = \langle\text{attribute definition}\rangle) \qquad (1)$$
$$S : REP(C\ BY\ b\ [= \langle\text{attribute definition}\rangle]) \qquad (2)$$

The symbol ":" may be read as "where". In (1) the relation R is extended by a new attribute C, subject to an optional predicate. In (2) the attribute C in relation S is replaced by attribute b. The new attribute will take the same values as C, unless the optional clause is included to define b in the same way as in (1).

The transpose operation also has two forms:

$$TRC(R : (c1, c2, \ldots, cn) \rightarrow c, b) \qquad (3)$$
$$TCR(S : c \rightarrow (c1, c2, \ldots, cn), b) \qquad (4)$$

The operation TRC (Transpose Rows to Column) transforms a (n + 1)ary relation R (a, cl, c2, ..., cn) into a ternary relation T (a, b, c) by changing cl to cn of the same domain into part of a new column c, and by adding a new attribute b for sequencing. Attribute a can be composite. Conversely TCR (Transpose Column to Rows) transforms a ternary relation into an (n + 1)ary relation by changing column c into a row described by attributes cl, c2, ..., cn in order of the values of b.

These integration commands can involve a substantial amount of processing so they are best done before data transmission, i.e. when indices are available. The decomposition algorithm aims to achieve this whenever possible by applying appropriate transformations to the query expression.

We can now apply the rules described earlier to PAL expressions. Selection and projection are unary operations which reduce the size of their operands, so by Rule 1, they should be distributed over any binary operation which is a breakpoint, provided the transformation produces an equivalent expression.

Alteration is a unary operation which meets the requirements of Rule 2 when its operand is locally stored, so it should be distributed over union. Sometimes alteration can increase the size of its operand, but in practice it will often be followed by a selection or projection which can also be distributed over the union to decrease the size. Transpose also meets the requirements of Rule 2 so it too can be distributed over union.

When an alteration is adjacent to a selection or projection, it is preferable to do the selection or projection first - the requirements of Rule 3 and of Rule 4 are met in this case. This will not be possible however when a selection is defined in terms of attributes created by the alteration, or if a projection removes attributes needed for the alteration.

These transformations produce an optimised query expression together with the breakpoints which indicate the points at which the expression should be split into subexpressions. An optimiser can then take over to carry out the second stage, namely, subquery allocations to nodes based on estimates of the communications cost, processing cost and response time of each strategy. We have not yet studied this second stage of optimisation.

The query decomposition technique is currently being implemented as part of a research prototype of PRECI* at Aberdeen. A fuller description of the technique is given in (6).

This work is supported by the UK Science Research Council and EEC Cost 11 bis grants. We wish to thank all our PRECI collaborators, particularly David Bell of Ulster Polytechnic and Jane Grimson of Trinity College, Dublin for comments and suggestions.

References

1. Hevner A. & Yao S. B. "Query Processing in a Distributed Database", Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, California, August 1978.

2. Epstein R., Stonebraker M. & Wong E. "Distributed Query Processing in a Relational Database System", Proceedings of ACM SIGMOD Conference, May 1978.

3. Dayal U. "Processing Queries over Generalisation Hierarchies in a Multidatabase System", Proceedings of the Ninth International Conference on Very Large Databases, Florence, Italy, 1983.

4. Deen S. M., Amin R. R., Ofori-Dwumfuo G. O. & Taylor M. C. "The Architecture of a Generalised Distributed Database System - PRECI*", to be published in the Computer Journal.

5. Deen S. M., Amin R. R. & Taylor, M. C. "Data Integration in Distributed Databases", submitted for publication.

6. Deen S. M., Amin R. R. & Taylor, M. C. "Query Decomposition in PRECI*", Proceedings of the Third Seminar on Distributed Data Sharing Systems, Parma, Italy, 1984 (to be published by North Holland; F. Schreiber and W. Litwin eds).

7. Motro A. & Buneman P. "Constructing Superviews", Proceedings of ACM SIGMOD Conference, Michigan, 1981.

8. Dayal U. & Hwang H. Y. "View Definition and Generalisation for Database Integration in MULTIBASE: a System for Heterogeneous Distributed Databases", Proceedings of the Sixth Berkeley Workshop on Distributed Data Management and Computer Networks, 1982.