Optimization of Query Postprocessing in Heterogeneous Distributed Multidatabase Systems

S. Mohsen Sedighi Janusz R. Getta
Department of Computer Science
University of Wollongong
Northfields Ave.,
Wollongong, NSW 2522
Australia
E-mail: [jrg, moh]@cs.uow.edu.au

June 19, 1996

Abstract

Query processing in heterogeneous distributed multidatabase systems requires decomposition of the queries into subqueries independently computed in the component databases. In the query postprocessing stage the partial results are integrated into the final answer.

This paper deals with the efficiency of the integration of partial results obtained from component database systems. We propose several optimization techniques based on transformations of data integration expression. It is shown that hidden dependencies between the arguments of a data integration expression can be used for reduction of the size of partial answers. It is proven that in specific cases a part of the final answer may be computed even when not all of the partial results are available.

Index Terms - Post query optimization, multidatabase systems, distributed query processing, query optimization, relational algebra.

Contents

1 Introduction **3**

2 Basic concepts **5**

3 Optimization techniques **7**

4 Query postprocessing **10**
4.1 Algorithm 10

5 Summary and further research **11**

A Appendix **14**

List of Figures

1 A labelled syntax tree of an expression $(r_1^* \cap r_2) \bowtie (r_3 - r_4^*)$. 6
2 A labelled syntax tree of an expression $(r_1^* \cup r_2) \bowtie (r_3 - (r_4 - r_5^*))$. 7
3 Labelling of a syntax tree. 9
4 Labelling of a syntax tree that leads to identification of a hidden dependence. . . 10
5 A skeleton of syntax tree with the common subtrees. 10

List of Tables

1 Labelling for binary operations. 6
2 Labelling for unary operations. 7
3 Hidden dependencies for join operation $(k = x \cap y)$. 9
4 Hidden dependencies for intersection operation. 14
5 Hidden dependencies for union operation. 14
6 Hidden dependencies for difference operation. 14

1 Introduction

Efficient query processing in distributed and heterogeneous multidatabase systems[SheL90] is an important factor in the future development of global information systems. Intensive research has been carried out in the area of query processing in distributed homogeneous database systems[BGWR81, ApHY83, CerP85, ChuH82, YuLi90], unfortunately, there are a number of reasons why these results cannot be directly applied to query processing in distributed and heterogeneous multidatabase systems[RePR89, SchC92, BreT85]. The major problem is that a *high level of autonomy* is preserved by the individual database systems after their incorporation into a multidatabase system[GarK88, Corn88]. The strong functional independence of individual database sites reduces the opportunity for collaboration of several database systems on processing of a single query. The high level of autonomy is the main reason why the individual database sites process the queries submitted from the other sites accordingly to a simple "receive-process-respond" schema. *Software* and *semantic heterogeneities*[SheL90] also each require different approaches to query processing in multidatabase systems. Software heterogeneity means that different database sites are managed by the different database management systems, which may be based on different database models. Semantic heterogeneity means that the same or similar database structures have different meaning in different databases. Software heterogeneities affect query processing when translation of queries from one formalism to another becomes necessary. It is also noted that semantic heterogeneities may affect the value of the use of replicated data by causing difficulty in the recognition of replicas of exactly the same semantic meaning which have been stored in different databases.

This paper proposes a new approach to the optimization of query processing in distributed and heterogeneous database systems. We consider a model of a multidatabase system that consists of several independent and heterogeneous database systems, henceforth known as *component databases*. Each of the component databases is located at a separate site on the network. The network user obtains a unified view of all components through a common database model. A common view is implemented by additional software, henceforth known as *middleware*, on top of the local DBMSs. Middleware is responsible for the translation and decomposition of queries, the coordination of query processing, and communication with other sites and translation of data into a common format. An important assumption is that middleware does not affect the autonomy of the component database systems. Communication with other databases is implemented through a message passing system in which the individual messages consist of either queries or the results of query processing.

A query submitted to the global view of a multidatabase system is initially decomposed into subqueries in such a way that processing of an individual subquery requires access to only one component database[Daya85]. Next each subquery is translated into the query language of a corresponding component database system before it is submitted for execution.

The strategy employed for subquery submission has an important impact on the performance of global query processing. Two opposite strategies of query submission can be identified. The first is the *sequential submission strategy* which is based on the assumption that partial results which have been obtained from the processing of one subquery may be used to speed up the computation of another subquery at another site thus it may be used to reduce amount of data transmitted between the component databases. For example, consider a multidatabase system providing access to the component databases through a common view in relational database model. The system provides a view of the relational tables r and s which are implemented in separate component databases d_1 and d_2. Note that implementation of r and s in the component databases does not need to be relational and that middleware provides all the necessary translations. Let x be a set of the attributes common to r and s. A query $r \bowtie s$ submitted to a common view of a multidatabase system is decomposed into two subqueries $q_{d_1}(r)$ and $q_{d_2}(s)$ expressed in the query languages of the respective database sites d_1 and d_2. Let us assume that accordingly to a sequential submission strategy the system first submits $q_{d_1}(r)$ to d_1. The results

obtained from d_1 are translated into a relational table r. In homogeneous distributed systems it is reasonable to send $\pi_x(r)$ to site d_2 to compute $s \ltimes \pi_x(r)$ in order to reduce amount of data transmitted from d_2. Unfortunately, we cannot apply the same method to a heterogeneous multidatabase system because each one of the component database systems accepts queries but not data. A solution to this problem exists. If $\pi_x(r)$ is not too large, then it can be translated into a preposition formula ϕ_r. Each tuple from $\pi_x(r)$ is translated into a conjunction of atomic terms *attribute=value*, while a formula ϕ_r is obtained as disjunction of the tuple translations. If table $\pi_x(r)$ is large then the system partitions it into the disjoint tables $\pi_x(r_1)$, ..., $\pi_x(r_k)$ such that $r = r_1 \cup \ldots \cup r_k$. Each one of the tables may be individually translated into ϕ_{r_1}, ..., ϕ_{r_k}. The second subquery is modified into $q_{d2}(\sigma_{\phi_r}(s))$ before its submission. We shall call such an approach a *heterogeneous semijoin*. Its objective is to reduce amount of data transmitted from site d_2. Obviously, a sequential query submission strategy does not provide the best results in every possible case. For instance, when the query processing times of $q_{d_1}(r)$ and $q_{d_2}(\sigma_{\phi_r}(s))$ dominate over the transmission time then sequential submission of subqueries is not the best method. In such a case it is better to simultaneously submit $q_{d_1}(r)$ and $q_{d_2}(s)$ for "parallel" execution in d_1 and d_2. A strategy where all subqueries are simultaneously submitted to the respective component databases is known as *simultaneous submission strategy* [YuLi90]. It is based on the assumption that simultaneous computation of subqueries in the component database saves more time than the hypothetical savings in transmission time achieved when sequential submission is applied. The strategy of simultaneous subquery submission seems to be somewhat more flexible in heterogeneous distributed multidatabase systems because, in addition to heterogeneity and autonomy of component databases, not all of the required database operations exist in all the component databases. In practice the best solution is a hybrid approach where some of the subqueries are submitted and computed sequentially yet others are submitted simultaneously. Determination of the best submission strategy is dependent on factors such as the computational complexity of the individual subqueries, the size of the respective results, and the efficiency of query processing in the component databases and the network parameters. Some of these problems have been already addressed. For instance, the works [DuKS90, ZhuL94, HaSN96] deal with the problem of subquery costs estimation within component databases. The optimization of query processing in multidatabase systems has been proposed in [LuSh92, LuOG93, LimS93].

In this paper we focus our attention on optimization in a phase of query postprocessing where a simultaneous strategy is applied for submission of the subqueries within a heterogeneous distributed multidatabase system. Query postprocessing commences when the results of the subquery computations return to a query submission site. The objective of query postprocessing is the integration of the partial results obtained into the final answer.

Although the subqueries are submitted simultaneously, their results do not return to a submission site at exactly the same moment of time. The delay is due to the high level of autonomy of the component databases. Individual components decide when to respond to a query. Low transmission speeds may also cause delays as does poor hardware of a system that runs a component database. Delays are also caused by the high computational complexity of the subquery, and may also occur when large amounts of data are transmitted.

A naive approach to the integration of partial results into the final answer is to wait until all partial results are available. The specific objective of this work is to show that it is possible to commence integration of the partial results even when not all of them are available and to show how to perform such a gradual integration. Moreover, we show how to utilize the time the system spends waiting for the partial results on the preparation of those answers already available with a view to more effective integration with other results. Preparation of the partial results is possible on both logical and physical levels. For instance, on the physical level prehashing of a relational table may speed up the computation of joint operations. Indexing and clustering of the partial results are the other options. Preparation of the partial results on the logical level resembles classical syntax-based query optimization [Ullm89]. A query submitted to a common view of a multidatabase system is transformed into an equivalent form so that it

can be computed faster when the results of all subqueries are available.

We shall concentrate on the preparation of the partial results on the logical level. Although it is a syntax-based optimization technique its transformations differ from the usual ones. The following example provides an insight into the transformations required in our case. Let us assume that, on a certain level of processing a query is represented as a sequence of database operations on the partial results obtained from the component databases. For instance, let it be a relational algebra expression. The first objective of optimization is to reorder the sequence of operations such that the partial results which are already available become the arguments of the common operations.

As a simple example consider a relational algebra expression r* - (s* - t) and assume that the arguments with an asterisk superscript, i.e. r*, t* are available for processing while the system is waiting for a partial result s. An expression r* - (s* - t) is equivalent to an expression (r* - s*) $\cup$ (r* $\cap$ t). Therefore, it is possible to compute a part of the final answer (r* - s*) without the third argument.

The next objective of syntax transformations is to move the union operations towards the top of a query syntax tree. If one of the union subexpressions can be computed using the available partial results then a part of the final answer can be presented to the user before the other partial results are available (see example above). Moreover, conversion of a query into a union of subqueries allows a reduction in the size of partial results. For example, in expression (r* - s*) $\cup$ (r* $\cap$ t) all tuples in (r* - s*) can be eliminated from an argument r* that occurs in the second subexpression. Such transformation provides an expression (r* - s*) $\cup$ ((r* $\cap$ s*) $\cap$ t). Moreover, note that computation of a subexpression (r* $\cap$ s*) does not need any additional accesses to r* and s*. It can be done "by the way" while (r* - s*) is computed.

The objective of the third kind of transformation is to detect the hidden dependencies between the partial results which are not bound by the common operations and to use the dependencies to reduce the size of the partial results. As an example consider an expression r* - π_x(s* $\bowtie$ t). It is possible to reduce the size of s*by replacing it with the result of s* $\ltimes$ π_x(r). Such a transformation is justified by a fact that all tuples from s* table that have different x-values from other tuples from r* table will not affect the result of difference and can be removed (independently whether they are joinable with any tuples in t or not).

The remainder of this paper is structured as follows. Next section contains the definitions of the basic concepts. Section 3 discusses the optimization techniques. The query postprocessing algorithm is presented in Section 4. The paper is concluded in Section 5.

2 Basic concepts

Consider a model of multidatabase system that provide its users with a relational view of the component databases, a user expresses a query in a high level query language of the relational model. The system translates a query into a relational algebra expression in such a method that computation of each one of its arguments requires access to a maximum of one of the component databases. Then, the system generates the subqueries in the query languages of the corresponding component databases and simultaneously submits all of them for execution. When the results of query processing return from the component databases, the system translates each one of them into a separate relational table. The problems to be resolved are stated below. Given a relational algebra expression e(r_1, ..., r_k) and some of its arguments, the problems are:

(i) what operations can be performed using the available arguments to obtain a part of the final answer (whenever it is possible),

(ii) how to reduce size of the arguments, and

(iii) how to prepare them for further processing.

We assume that the system has enough time to perform such operations. In the future we shall call e(r_1, ..., r_k) the *data integration expression.*

Let π, σ, $\bowtie$, $\ltimes$, $\cap$, $\cup$, - denote the standard operations of relational algebra. In addition to the standard system we will define the following operation, namely that of *generalized difference.*

A generalized difference of relational tables r and s is denoted by r — s and its result is the set of all tuples from a table r that cannot be joined with any tuples from a table s. Generalized difference does not extend the expressive power of relational algebra and it is used only as a shorthand notation because r — s = r - (r $\ltimes$ s).

A syntax *tree* of a data integration expression is a binary tree where the internal nodes are labelled with the respective relational algebra operations and the leaves are labelled with the arguments. Each of the arguments of the data integration expression which are available for computations is marked with an asterisk superscript. A syntax tree represents an order as to how the partial results of subqueries should be integrated into the final answer.

All edges on the paths from the subscript leaf nodes to a root node are labelled to represent the flow of data from one operation to another. A label consists of a qualifier and argument name for which a flow of data is considered. There are five qualifiers: A, A+, E, E+, N+. Let α , β be the operation nodes in a syntax tree such that result of α becomes an argument of β. Moreover, let relational table r be an argument of α. An edge from α to β obtains a label A(r) if operation α returns all (A) and no other tuples from its argument r. An edge from α to β obtains a label A(r)+ if a result of operation α returns all tuples from r plus some other tuples. An edge from α to β is labelled by E(r) if in the result of α there exist some (E) tuples from r and there is no other tuples. An edge from α to β is labelled by E(r)+ if operation α returns some tuples from r plus some other tuples. Finally an edge from α to β obtains a label N(r)+ if operation α returns no (N) tuples from r plus some other tuples. As an example consider a data integration expression ($r_1^* \cap r_2$) $\bowtie$ (r_3 - r_4^*) whose labelled syntax tree is given in Figure 1.

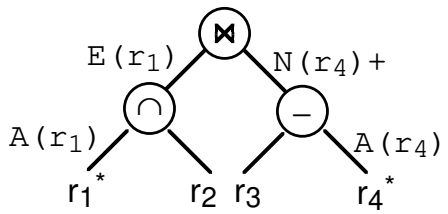

Figure 1: A labelled syntax tree of an expression ($r_1^* \cap r_2$) $\bowtie$ (r_3 - r_4^*).

An edge between the nodes ($\cap$) and ($\bowtie$) is labelled by E(r_1). If a set of all tuples from r_1 (A(r_1)) is intersected with some other set then in the result there exist some tuples from r_1 (E(r_1)) and no other tuples. An edge between the nodes (-) and ($\bowtie$) is labelled by N(r_4)+. If a set of all tuples from r_4 (A(r_4)) is subtracted from a set of tuples r_3 then the result contains no tuples from r_4 (N(r_4)+) and some other tuples from r_3. Labelling of the edges on the higher levels of syntax tree is described by Table 1 and Table 2. For instance, the leftmost column in

label\operation	*join*	*intersection*	*l-difference*	*r-difference*	*union*
A(r)	E(r)	E(r)	E(r)	N(r)+	A(r)+
A(r)+	E(r)+	E(r)+	E(r)+	N(r)+	A(r)+
N(r)+	N(r)+	N(r)+	N(r)+	E(r)+	E(r)+
E(r)	E(r)	E(r)	E(r)	E(r)+	E(r)+
E(r)+	E(r)+	E(r)+	E(r)+	E(r)+	E(r)+

Table 1: Labelling for binary operations.

Table 1 determines the labelling in all cases when a node represents join operation and one of its

label\operation	*selection*	*projection*
A(r)	E(r)	A_x(r)
A(r)+	E(r)+	A_x(r)+
N(r)+	N(r)+	N_x(r)+
E(r)	E(r)	E_x(r)
E(r)+	E(r)+	E_x(r)+

Table 2: Labelling for unary operations.

argument edges is labelled either by A(r), or A(r)+ or E(r) or E(r)+ or N(r)+. Then, the result edge of a join node should be labelled respectively by either E(r) or E(r)+ or N(r)+ or E(r) or E(r)+. Because the difference operation is not commutative we have to separately consider the cases when an argument edge represents the left or right argument. Figure 2 provides a labelled syntax tree for an expression ($r_1^* \cup r_2$) $\bowtie$ (r_3 - (r_4 - r_5^*)). An edge between the nodes ($\bowtie$) and (-) is labelled by E(r_5)+ because an argument edge of a node (-) is labelled by N(r_5)+ and in Table 1 intersection of N(r)+ row and right-difference column contains a label E(r)+. If a node

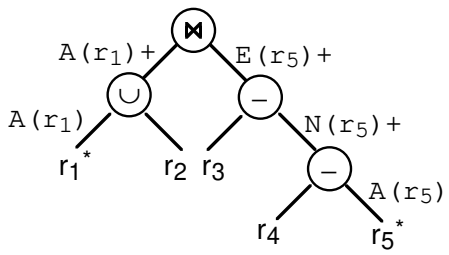

Figure 2: A labelled syntax tree of an expression ($r_1^* \cup r_2$) $\bowtie$ (r_3 - (r_4 - r_5^*)).

represents projection operation then a qualifier of edge label is indexed by a set of projected attributes. For instance E_x(r)+ means that result of operation contains some of x-values of the tuples from r plus some other x-values. Note that the binary operations represented in Table 1 do not change an index of qualifier.

3 Optimization techniques

This section reviews the optimization techniques proposed to improve performance of query postprocessing in multidatabase systems. We assume that data integration procedures are represented by the various relational algebra expressions.

This first group of optimizations includes all classical transformations of relational algebra expressions that speed up the query processing in conventional database systems. Their objective is to compute the unary operations as early as possible and to detect common subexpressions. These transformations commute projection and selection operations with the binary operations and combine sequences of unary operations.

The objective of the next group of optimizations is to transform a data integration expression into an equivalent form such that the arguments available for further computations are bound by the common operations and the total number of operations does not increase. This kind of transformations relies on application of the associative and distributive laws of relational algebra. The first objective is to reduce a number of operations in a data integration expression by a simple transformation that changes the order of operations. For instance $r_1^* \bowtie r_2 \bowtie ... \bowtie r_k^*$ is transformed into $r_{1k}^* \bowtie r_2 \bowtie ... \bowtie r_{k-1}$ where $r_{1k}^* = r_1^* \bowtie r_k^*$.

The next objective is to distribute binary operations[1]over union operations to obtain an expression which is a union of subexpressions such that each one of them contains no union operation and one of them can be completely computed using the available arguments. Distribution of the binary operations is always possible except the case when a union subexpression is on the right hand side of difference operation e.g. r* - (s* ∪ t). In this case we apply a transformation rule that replaces union with a difference, e.g. (r* - s*) - t. Such transformations are reasonable if the leftmost subexpression can be immediately computed. In general, distribution of binary operations over union increases a total number of operations in a data integration expression. Therefore we should consider it beneficial only if one of the union subexpressions can be immediately computed and a part of the final result may be delivered to the user even though not all of the arguments of the data integration expression are available. For example transformation of the expression (r* ∪ s) ⋈ t* into (r* ⋈ t*) ∪ (s ⋈ t*) is recommended because (r* ⋈ t*) provides part of the final answer.

The third kind of transformation reshuffles the available arguments of the data integration expression by replacement of the difference operations with the intersection and complement of the argument and by the application of the distributive laws of set algebra. This transformation is limited to certain subexpressions of the data integration expression. As an example consider the expression (t - r*) - (s* ∪ v). After the elimination of any difference it is equivalent to (t ∩ **x**(r*)) ∩ **x**(s* ∪ v) where **xx**denotes the set complement. Finally, using De Morgan's laws for sets and eliminating the complement operations we obtain (t - (r* ∪ s*)) - v. The transformation allows the immediate computation of (r* ∪ s*) and thus eliminates one operation.

It is important to recall that relational algebra is not finitely axiomatizable [ImiL84]. For this reason it is not possible to provide a complete and finite set of the transformation rules that correctly bound the available arguments' common operations and eliminate the operations in every possible case.

The objective of the next group of transformations is to reduce the size of the arguments of the data integration expression through the identification of *hidden dependencies*.

We say that there exists a hidden dependence between two not adjacent arguments of a data integration expression when it is possible to construct an expression that involves both arguments, such that its result is included in one of them and it can replace the argument without affecting the computation of the data integration expression in any possible case.

For instance consider a data integration expression (π_x(r* - v) ∩ t) ∩ (σ_ϕ(s*) ∪ w). There exists a hidden dependence between r and s because s can be replaced with the result of s ⋉ π_x(r) and such replacement does not affect the result of computation of the data expression for arbitrary values of t, v, and w. Intuitive justification of this conclusion is as follows. If s contains a tuple ξ such that there does not exist a tuple in r that has the same x-values as ξ then it will never be a member of the final result and thus it can be removed from s. This is true because the difference and intersection operations in an π_x(r* - v) ∩ t do not add any new tuples to π_x(r).

The hidden dependencies can be detected by labelling the syntax tree of the data integration expression. Given a syntax tree with Ψ operation in its root node and the arguments r* and s* are in the left and right subtrees (see Figure 3). Then in order to detect a hidden dependence between r* and s* we have to label the paths from the arguments to the root node. Then, depending on an operation Ψ and the labels L_r and L_s of the topmost edges, a hidden dependence can be determined using the tables 3, 4, 5, 6 (Appendix A). For instance labelling of a syntax tree of expression (π_x(r* - v) ∩ t) ∩ (σ_ϕ(s*) ∪ w) is given in Figure 4. Intersection of a column marked by E(s)+ and a row marked by E(s) in Table 4 provides a hidden dependence between r* and s*. Whenever an entry in the tables 3, 4, 5, 6 is equal to NIL then a hidden dependence does not exist. An optimization technique based on marking the tuples of available arguments is important when a data integration expression has one of two or more identical subexpressions

[1]Unary operations are distributed over union in the first step of optimization.

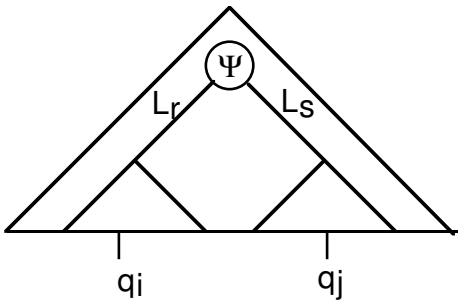

Figure 3: Labelling of a syntax tree.

$\bowtie$	$A(s)$	$A(s)+$	$N_y(s)+$	$E_y(s)$	$E_y(s)+$
A(r)	compute r$\bowtie$s	s = s $\ltimes$ r	s = s $\ltimes$ π_k(r)	r = r $\ltimes$ π_k(s) s = s $\ltimes$ π_k(r)	s = s $\ltimes$ π_k(r)
A(r)+	r = r $\ltimes$ s	NIL	NIL	r = r $\ltimes$ π_k(s)	NIL
N_x(r)+	r = r $\ltimes$ π_k(s)	NIL	NIL	r = r $\ltimes$ π_k(s)	NIL
E_x(r)	r = r $\ltimes$ π_k(s) s = s $\ltimes$ π_k(r)	s = s $\ltimes$ π_k(r)	s = s $\ltimes$ π_k(r)	r = r $\ltimes$ π_k(s) s = s $\ltimes$ π_k(r)	s = s $\ltimes$ π_k(r)
E_x(r)+	r = r $\ltimes$ π_k(s)	NIL	NIL	r = r $\ltimes$ π_k(s)	NIL

Table 3: Hidden dependencies for join operation (k = x $\cap$ y).

such that there exists a hidden dependence in one of them. For instance consider a skeleton of data integration expression given in Figure 5. A subtree T is common for the operations α and β but only in β case there exists a hidden dependence between r and s. If we assume that dependence between r and s allows the replacement of r with f(r, s) then the right hand side subtree T becomes different from the left hand side subtree T. Unfortunately such optimization is not correct because it is no longer possible to compute T once. The solution to this problem is in the marking of the tuples in r with (+) and (-) markers and clustering r according to the markers possessed by the individual tuples. The tuples that belong to f(r, s) are marked with (+) while the other tuples, i.e. r - f(r, s) are marked with (-). Moreover, the computation of the subexpression T should be done in such a way that it returns two results, one for evaluation of a and the other one for evaluation of β. The result of T which is later on used as an argument of a contains all tuples independent of their markers. On the other hand, every tuple in the result of T which becomes an argument of β and such that it consists of a part or entire tuple marked with (-) should be discarded before computation of β. Such a tuple does not affect the result of β.

Marking of tuples allows for the computation of common subexpressions once and a reduction of the size of the argument. It requires specific implementation of relational algebra operations that preserves the marking and in the selected cases produces either of two results.

As an example consider a data integration expression (r* $\cup$ t) - (s* $\bowtie$ w). Labelling of the paths from the leaf nodes r* and s* to the root node (-) provides the labels A(r)+ and E(s). A hidden dependence between r and s does not exist. According to other optimization techniques the expression should be transformed into a union of subexpressions (r* - (s* $\bowtie$ w)) $\cup$ (t - (s* $\bowtie$ w)). Labelling in the first subexpression indicates that s can be replaced with the result of s $\ltimes$ r. Unfortunately, the reduction of s is not correct because it destroys common subexpressions. Therefore, s is marked and clustered into s' = s $\ltimes$ r and s" = s - s'. When w is available then the entire result of s* $\bowtie$ w* is stored in order to compute the second subexpression while only s'* $\bowtie$ w is kept for computation of the first subexpression.

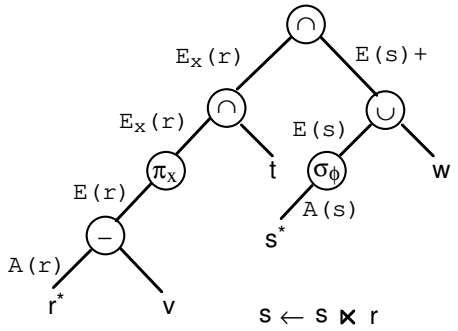

Figure 4: Labelling of a syntax tree that leads to identification of a hidden dependence.

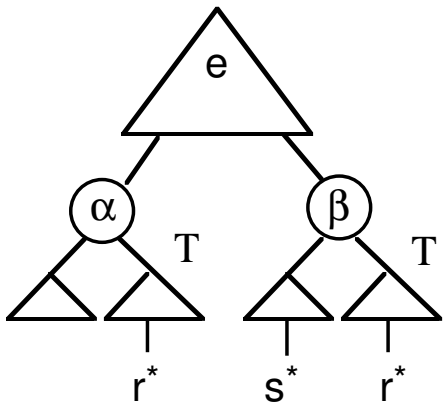

Figure 5: A skeleton of syntax tree with the common subtrees.

4 Query postprocessing

This section provides an algorithm for query postprocessing in heterogeneous, distributed multidatabase systems. It is assumed that in the initial stage of query processing a multidatabase system submits the subqueries simultaneously. Moreover, we assume that it is possible to estimate how much time is needed to compute the subqueries within the respective component databases. Integration of the partial results is performed accordingly to the following algorithm.

4.1 Algorithm

Input:

A syntax tree of data integration expression e(r_1, ..., r_k) and a sequence of pairs <t_1, r_1>, ..., <t_k, r_k> ordered in the ascending order of the values t_1,... , t_k. Each pair <t_i, r_i> for i =1, ..., k represents estimation of time elapsed from submission of a subquery q_i till arrival of its partial result r_i.

Output:

The result of e(r_1, ..., r_k).

Method

Let R be a set of partial results available for data integration. Initially $R = \emptyset$.

When a partial result r_i arrives after time t_i it is inserted into R. The system periodically inspects the set of partial results. If R is not empty, i.e. it contains the partial results r_i, r_{i+1} , ..., r_j then the following actions take place.

(i) All operation that have all of their arguments available are immediately computed and the partial results r_i, r_{i+1}, ... $r_j$are removed from R. The syntax tree is modified after the computations. The system inspects the set R and if it is not empty step (i) is repeated for the new arrivals. At any moment of time the system is able to determine amount of time t_a which is left to arrival of the next partial answer r_{j+1}.

(ii) The system identifies an appropriate syntax-based optimization techniques to reshuffle the arguments, to compute a part of the final answer, and to eliminate some of the operations. All optimizations that require less time than t_a are performed. Then, the system inspects R for new arrivals. If it is not empty the step (i) is repeated. Otherwise the system continues optimization.

(iii) The system identifies the hidden dependencies and all possible reductions of the arguments. All optimization that can be done before arrival of the next partial result are performed. If R is still empty the system continues optimizations, otherwise it returns to step (i).

(iv) The system identifies all cases when reduction of the arguments conflicts with computation of the common subexpressions. In such cases the tuples from respective arguments are marked. If a set R is still empty the system waits for new partial arguments as no other optimizations can be done. Otherwise the system returns to step (i).

5 Summary and further research

This work addresses the problem of efficiency in query postprocessing in heterogeneous distributed multidatabase systems. Query postprocessing integrates the partial results of the subqueries computed in the component database into the final answer. The efficiency of integration strongly depends on the strategy selected for the submission of subqueries to the component database systems. We identify two generic submission strategies: sequential and simultaneous. In this paper query postprocessing is considered only for simultaneous submission strategy.

On the logical level, optimization of query postprocessing is usually concerned with syntactical transformation of data integration expressions in order to reduce response times. Our approach takes advantage of the time that the system is waiting until all partial results performed are available. Three optimization techniques are proposed. The first one is based on the transformation of data integration expressions so that it is possible to compute a part of the final answer from incomplete partial results and it is thus possible to eliminate some of the operations. Another reduces size of the available arguments by identification of the hidden dependencies. The third one performs marking of the partial results so that further computations can be performed with greater speed.

An interesting question is the effectiveness of this sort of optimization. As in the other approaches to syntax-based query optimization, the effectiveness of the entire process depends on how complex the data integration expression really is and how large the arguments are. Therefore, it is difficult to provide formulas that precisely determine the costs and benefits of optimization. Improvements are not significant for simple data integration expressions with small sized arguments. It is important to note that in our case the optimizations do not need any extra processing time because all of them are performed while the system is waiting for the partial results. Another important feature is that in some of the cases it is possible to compute a part of the final answer even when the partial results are incomplete. Preprocessing of the partial results by marking tuples enables faster computation of binary operations. Detection of the hidden dependencies allows size reduction of arguments.

Although we apply several optimization techniques in our query postprocessing algorithm, there are a number of other aspects that require further investigation. Though we have assumed that the exact response time was known for each subquery, there are cases when the response

time can be estimated with only limited precision. Therefore, any more sophisticated query postprocessing algorithm should be able to react in an intelligent way when some of the partial results arrive earlier or later than expected. Yet, another problem is the identification of the best subquery submission strategy when some information about network parameters, performance of the component database systems and computational complexity of subqueries is available.

References

[ApHY83] Apers P. M. G., Hevner A. R., Yao S. B. Optimization Algorithms for Distributed Queries, *IEEE Transactions on Software Engineering*, Vol. 9, No. 1, January 1983, pp. 57-68.

[BGWR81] Bernstein P. A., Goodman N., Wong E., Reeve C. L., Rothnie J. B. Query Processing in a System for Distributed Databases (SDD-1), *ACM Transactions on Database Systems*, Vol. 6, No. 4, December 1981, pp. 602-625.

[BreT85] Breitbart Y. J., Tieman L. R., ADDS-Heterogeneous Distributed Database System, in *Distributed Data Sharing Systems*, Schreiber F. A., Litwin W. (Eds), Elsevier Science Publishers, North-Holland, 1985.

[CerP85] Ceri S., Pelagatti G., *Distributed Databases: Principles & Systems*, McGraw-Hill, New York, 1985.

[ChuH82] Chu W. W., Hurley P., Optimal Query Processing for Distributed Database Systems, *IEEE Transactions on Computers*, Vol. 31, No. 9, September 1982, pp. 135-150.

[Corn88] Cornelis B., Site Autonomy in a Distributed Database Environment, IEEE Spring COMPCON 1988, *Thirty-Third IEEE Computer Society International Conference*, 1988, San Francisco, CA, USA, pp. 440-443.

[Daya85] Dayal U., Query Processing in a Multidatabase System, in *Query Processing in Database Systems*, Kim W., Reiner D. S., Batory D. S. (Eds.), Spring-Verlag, Berlin, 1985.

[DuKS90] Du W., Krishnamurthy R., Shan M.C. Query Optimization in Heterogeneous DBMS, *Proceedings of the 18th VLDB Conference* pp. 277-299, 1992

[GarK88] Garcia-Molina H., Kogan B., Node Autonomy in Distributed Systems, *Proceedings International Symposium on Databases in Parallel and Distributed Systems*, 1988, pp. 158-166, In: Multidatabase Systems: An Advanced Solution for Global Information Sharing, Hurson A. R., Bright M. W., Pakzad S. H. (Eds.), IEEE Press, Washington, 1994.

[HaSN96] Harangsri B., Shepherd J., Ngu A. Query Classification in Multidatabase Systems, Proceedings of the 7th Australasian Database Conference, ADC'96, pp.147- 156, 1996

[ImiL84] Imielinski T., Lipski W. The Relational Model of Data and Cylindric Algebras, *Journal of Computer and System Science*, vol. 28, no. 1, 1984, pp. 88-102

[LimS93] Lim E-P., Srivastava J., Query Optimization and Processing in Federated Database Systems, *2nd International Conference on Information and Knowledge Management, CIKM*'93 Washington DC, 1993.

[LuOG93] Lu H., Ooi B-C., Goh C-H., Multidatabase Query Optimization: Issues and Solutions, *Proceedings RIDE-IMS'93, Research Issues in Data Engineering: Interoperability in Multidatabase Systems*, April 1993, Vienna, Austria, pp. 137-143.

[LuSh92] Lu H., Shan M-C., On Global Query Optimization in Multidatabase Systems, *Second International Workshop on Research Issues on Data Engineering: Transaction and Query Processing*, Tempe, Az, USA, February. 1992, pp. 217.

[RePR89] Reddy M. P., Prasad B. E., Reddy P. G., Query Processing in Heterogeneous Distributed Database Management Systems, in *Integration of Information Systems: Bridging Heterogeneous Databases*, Gupta A. (Ed.), IEEE Press, New York, 1989, pp. 264-278.

[SchC92] Scheuermann P., Chong E. I., Query Processing in Multidatabase Systems, *International Conference on Systems, Man and Cybernetics*, New York, 1992, pp. 1250-1254.

[SheL90] Sheth A. P., Larson J. A., Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases, *ACM Computing Surveys*, vol. 22, no. 3, September 1990, pp. 183-236.

[Ullm89] Ullman J.D. Principles of Database and Knowledge-Base Systems, vol. 2, pp. 662-668, Computer Science Press, 1989

[YuLi90] Yu C., Liu C., Experiences with Distributed Query Processing, *Sixth International Conference on Data Engineering*, Los Angeles, CA, USA, February 1990, pp. 192-199.

[ZhuL94] Zhu Q., Larson P.A., A Query Sampling Method for Estimating Local Cost Parameters in a Multidatabase System, Data Engineering, pp. 144-153, 1994

A Appendix

Hidden dependencies for intersection, union and difference operations. Note that symbol $\ltimes$ denotes semijoin and "—" denotes generalized difference.

$\cap$	$A(s)$	$A(s)+$	$N_x(s)+$	$E_x(s)$	$E_x(s)+$
A(r)	compute r $\cap$ s	s = s $\ltimes$ r	s = s $\ltimes$ r	r = r $\ltimes$ $\pi_x(s)$ s = s $\ltimes$ r	s = s $\ltimes$ r
A(r)+	r = r $\ltimes$ s	NIL	NIL	r = r $\ltimes$ $\pi_x(s)$	NIL
N_x(r)+	r = r $\ltimes$ s	NIL	NIL	r = r $\ltimes$ $\pi_x(s)$	NIL
E_x(r)	r = r $\ltimes$ s s = s $\ltimes$ $\pi_x(r)$	s = s $\ltimes$ $\pi_x(r)$	s = s $\ltimes$ $\pi_x(r)$	r = r $\ltimes$ $\pi_x(s)$ s = s $\ltimes$ $\pi_x(r)$	s = s $\ltimes$ $\pi_x(r)$
E_x(r)+	r = r $\ltimes$ $\pi_x(s)$	NIL	NIL	r = r $\ltimes$ $\pi_x(s)$	NIL

Table 4: Hidden dependencies for intersection operation.

$\cup$	$A(s)$	$A(s)+$	$N_x(s)+$	$E_x(s)$	$E_x(s)+$
A(r)	compute r $\cup$ s	either r = r - s or s = s - r	s = s - r	s = s - r	s = s - r
A(r)+	either r = r - s or s = s - r	either r = r - s or s = s - r	s = s - r	s = s - r	s = s - r
N_x(r)+	r = r - s	r = r - s	NIL	NIL	NIL
E_x(r)	r = r — s	r = r — s	NIL	NIL	NIL
E_x(r)+	r = r — s	r = r — s	NIL	NIL	NIL

Table 5: Hidden dependencies for union operation.

–	$A(s)$	$A(s)+$	$N_x(s)+$	$E_x(s)$	$E_x(s)+$
A(r)	compute r = r - s	r = r - s s = s $\ltimes$ r	s = s $\ltimes$ r	s = s $\ltimes$ r	s = s $\ltimes$ r
A(r)+	r = r - s	r = r - s	NIL	NIL	NIL
N_x(r)+	either r = r - s or s = s - r	NIL	NIL	NIL	NIL
E_x(r)	s = s $\ltimes$ r r = r — s	s = s $\ltimes$ $\pi_x(r)$ r = r — s	s = s $\ltimes$ $\pi_x(r)$	s = s $\ltimes$ $\pi_x(r)$	s = s $\ltimes$ $\pi_x(r)$
E_x(r)+	r = r — s	r = r — s	NIL	NIL	NIL

Table 6: Hidden dependencies for difference operation.