September 2000

Query scrambling in distributed multidatabase systems

J. R. Getta
University of Wollongong, jrg@uow.edu.au

Query scrambling in distributed multidatabase systems

Abstract

This work addresses the problem of efficient query processing in multidatabase systems distributed over wide-area networks. The solution unifies the query scrambling and reduction approaches to dynamic optimization of query processing plans at the data integration stage. The paper presents a new data integration algorithm based on query scrambling and the extended reduction technique. The algorithm both reschedules data integration operations and reductions of arguments and permits the concurrent computations of operations and reductions.

Publication Details

This paper originally appeared as: Getta, JR, Query scrambling in distributed multidatabase systems, Proceedings. 11th International Workshop on Database and Expert Systems Applications, 4-8 September 2000, 647-652. Copyright IEEE 2000.

Query Scrambling in Distributed Multidatabase Systems

Janusz R. Getta

School of Information Technology and Computer Science
University of Wollongong
Wollongong, NSW 2522, Australia
E-mail: *jrg@cs.uow.edu.au*

Abstract

This work addresses the problem of efficient query processing in multidatabase systems distributed over wide-area networks. The solution unifies the query scrambling and reduction approaches to dynamic optimization of query processing plans at data integration stage. The paper presents a new data integration algorithm based on query scrambling and extended reduction technique. The algorithm both reschedules data integration operations and reductions of arguments and permits for the concurrent computations of operations and reductions.

1. Introduction

Query processing in multidatabase systems distributed over wide-area networks exhibits a number of complex performance problems. Selection of the best global query processing plan at preprocessing stage and dynamic optimization of data integration operations at postprocessing stage have the most important impact on performance. A typical distributed multidatabase system consists of a central database system linked to a number of remote, autonomous and heterogeneous local database systems. A software layer installed at a central site makes distribution and heterogeneity of the local systems transparent to the end-users. In a typical multidatabase system a users obtain a relational view of fully homogeneous and centralised database system. The queries submitted by the users are globally optimized, decomposed into the subqueries, and translated into the dialects of query languages available at the local systems. Then, a query processing coordinator optimizes a global processing plan and submits the subqueries to the local sites accordingly to this plan. The results collected at the postprocessing stage are translated into a common data format and integrated into the final answer.

Query optimization at preprocessing stage includes global optimization and selection of the optimal subquery processing plan. A number of research works have already targeted selection of the optimal global processing plans [4, 5, 6, 12, 9, 15, 7].

Query optimization at postprocessing stage addresses the problem of effective integration of partial results into the final answer. In a multidatabase system where a global user's view is the relational one a data integration procedure is formally represented as an expressions of relational algebra and called as *data integration expression.* Optimization of data integration expressions is conceptually different from the classical syntax based and cost based optimization of relational algebra expressions. The main difference is that not all of the arguments are available at the beginning of integration stage. Due to congestion and network failures, different computational complexities of the subqueries and different workloads at the local sites the partial results arrive at a central in a highly irregular manner. Such behaviour may significantly delay computation of the final answer. Reduction of the delays is based on a common-sense approach to utilise idle time and to perform the computations that may reduce the future workload. This idea was for the first time independently proposed in [3] and [2].

The works [2, 13] presented and evaluated a query transformation technique called as *query scrambling.* The objective of query scrambling is to change an order of operations in data integration expression such that the system is able to perform the other useful integrations while it is waiting for the missing arguments. [3] proposed the general transformations of relational algebra expressions into the equivalent ones where the computations involving the missing arguments are delayed to the later stages of integration. [2] considered the same problem for join expressions and proposed a method that changes an order of join operations to

0-7695-0680-1/00 $10.00

avoid the delays. A cost based analysis of the same algorithm was provided in [13]. Generalization of the method to iterator-based execution database engines was proposed in [1]. The works [11, 10] introduced a concept of *reduction* as another class of operations that may be performed to utilise idle time. The objective of reduction is to eliminate from the arguments available at a central site all rows that have no impact on the final result of data integration.

The objective of this work is to unify the approaches of query scrambling and reductions to dynamic optimization of query execution plans at data integration stage. In particular, we remove the limitations of query scrambling to join expressions only and we remove the limitations of reductions to computation of one operation at a time. Our solution further simplifies identification of reductions and shows that transformations of join expressions are the special cases of reduction operations.

The rest of the paper is structured as follows. Section 2 describes a method for labelling data integration syntax trees and identification of reduction operations. The next section contains a new query scrambling algorithm based on a model of concurrent computation of data integration expression. Finally, section 4 compares the present solution with the others, lists a number future works, and concludes the paper.

2. Reductions

This section introduces a concept of reduction and proposes a method for identification of reductions in data integration expressions.

We consider a typical system of relational algebra operations consisting of $\bowtie_x$ (either equijoin over x attributes or Cartesian product when $x = \emptyset$), $\cup$ (union), $\cap$ (intersection), $-$ (difference), π_x (projection on the attributes in x), σ_ϕ (selection over condition ϕ)}.

Let $e(r_1, \ldots, r_n)$ be a data integration expression constructed over the arguments $r_1, \ldots, r_n$ and the operations listed above. The arguments represent the results of subqueries submitted to the local database systems. Let r_i, r_j be any two arguments that are not directly bound with a relational operation and let x be a non empty intersection of their relational schemas. Let α be an operation in the root of the smallest syntax subtree that includes both arguments r_i and r_j, see Figure 1.

We say that r_i can be *positively reduced* by r_j if elimination from r_i all rows that have no rows with matching x-values in r_j has no impact on the result of expression e. Positive reduction of r_i by r_j is denoted by $\varrho^+(r_i \leftarrow r_j)$ and it is computed as semijoin $r_i \ltimes_x r_j$.

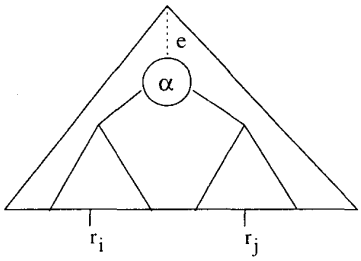

Figure 1. A sample syntax tree of data integration expression e

We say that r_i can be *negatively reduced* by r_j if elimination from r_i all rows that have at least one row with matching x-values in r_j has no impact on the result of expression e. Negative reduction of r_i by r_j is denoted by $\varrho^-(r_i \leftarrow r_j)$ and it is computed as $r_i - (r_i \ltimes_x r_j)$.

Simultaneous reduction of r_i by r_j and r_j by r_i is called as *mutual reduction* and it is denoted by $\varrho(r_i \leftrightarrow r_j)$.

To find whether an argument r_i can be reduced by an argument r_j we have to trace what subsets of x-values from r_i and r_j are included in the direct arguments of operation α located at the root of smallest common subtree of r_i and r_j. For instance, if all operations performed along the path from argument r_i to node α are either join, selection or projection then it is easy to prove that projection on x of an argument of α is a subset of $\pi_x(r_i)$. If α is a join operation and projection on x of one of its arguments is a subset of $\pi_x(r_i)$ and projection of the other is a subset of $\pi_x(r_j)$ then it is possible to remove from r_i and r_j all rows that do not match over attributes x.

To trace a flow of data from operation to operation we attach the labels to the edges of syntax tree accordingly to the following rules.

(i) If an edge represents an argument whose projection on x is always identical to $\pi_x(r)$ then such edge obtains a label x_r.

(ii) If an edge represent an argument whose projection on x may be a subset of $\pi_x(r)$ then such edge obtains a label x_r-.

(iii) If an edge represents an argument whose projection on x may have no values in $\pi_x(r)$ then such edge obtains a label $-x_r$.

(iv) If an edge represents an argument whose projection on x contains all values from $\pi_x(r)$ and some other values then such edge obtains a label x_r+.

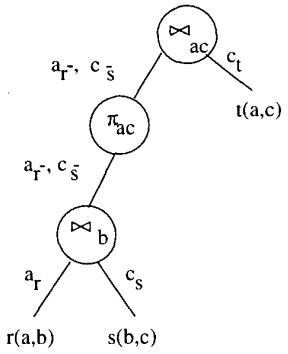

Figure 2. A labelled syntax tree

	$\cup$	$\cap$	$\bowtie_z$
x_r	x_r+	x_r-	$z = \emptyset \Rightarrow x_r$ else $\Rightarrow x_r-$
x_r+	x_r+	x_r*	x_r*
x_r-	x_r*	x_r-	x_r-
$-x_r$	x_r*	$-x_r$	$-x_r$
x_r*	x_r*	x_r*	x_r*

Table 1. Reduction table, part 1

(v) If an edge represents an argument whose projection on x is any other set of x-values then such edge obtains a label x_r*.

As an example consider c-values of an argument $s(b,c)$ in Figure 2. The left edge of top level join operation has a label c_s- because the result of $\pi_{ac}(s(bc) \bowtie_b r(a,b))$ is in the general case a subset of $\pi_c(s)$. The right edge of top join operation has a label c_t because it represents an argument $t(a,c)$. If top level join operation is performed over a subset of $\pi_c(s)$ and a set of $\pi_c(t)$ then it is possible to remove from s and t all rows that do not have matching c-values, i.e to perform a reduction $\varrho^+(s \leftrightarrow t)$.

In the general case identification of reductions between any two arguments r_i and r_j requires labelling of the paths from r_i and r_j up to the root node of their smallest common syntax subtree. The labelling is performed accordingly to the rules included in the tables 1 and 2.

For instance, if an edge representing an argument of join operation has a label x_r then an edge representing the result of join operation obtains a label x_r- because a potential join may eliminate some of x-values and it does not create any new x-values. A label is taken from the intersection of row x_r and column $\bowtie_z$ in Table 1.

Table 2 has the columns $-_{left}$ and $-_{right}$ because set difference is not commutative and we have to distinguish between its left and right argument.

	σ_ϕ	$\pi_z, x \cap z \neq \emptyset$	$-_{left}$	$-_{right}$
x_r	x_r-	$(x \cap z)_r$	x_r-	$x = z \Rightarrow -x_r$ else x_r*
x_r+	x_r*	$(x \cap z)_r*$	x_r*	$x = z \Rightarrow -x_r$ else x_r*
x_r-	x_r-	$(x \cap z)_r-$	x_r-	x_r*
$-x_r$	$-x_r$	$(x \cap z)_r*$	$-x_r$	x_r*
x_r*	x_r*	$(x \cap z)_r*$	x_r*	x_r*

Table 2. Reduction table, part 2

The reductions are determined by the following rules.

Rule 1 ($\alpha = \bowtie_z$ and $z \neq \emptyset$ and $z \neq x$)

(a) If both arguments of $\bowtie_z$ have labels x_r or x_r- and x_s or x_s- then $\varrho^+(r \leftarrow s)$ is a valid reduction.

(b) If one argument of $\bowtie_z$ has either label x_r or x_r- and the other has either label x_s* or x_s+ then $\varrho^+(s \leftarrow r)$ is a valid reduction.

Rule 2 ($\alpha = \cup$)

(a) If one argument of $\cup$ has either label x_r or x_r+ and the other has either label x_s or x_s+ then either $\varrho^-(r \leftrightarrow s)$ or $\varrho^-(s \leftarrow r)$ is a valid reduction.

(b) If one argument of $\cup$ has label x_r and the other has label x_s- then $\varrho^-(s \leftarrow r)$ is a valid reduction.

(c) If one argument of $\cup$ has label x_r+ and the other has label x_s* then $\varrho^-(s \leftarrow r)$ is a valid reduction.

Rule 3 ($\alpha = \cap$)

(a) If both arguments of $\cap$ have labels x_r or x_r- and x_s or x_s- then $\varrho^+(r \leftrightarrow s)$ is a valid reduction.

(b) If one argument of $\cap$ has either label x_r or x_r- and the other has either label x_s* or x_s+ then $\varrho^+(s \leftarrow r)$ is a valid reduction.

(c) If one argument of $\cap$ has label $-x_r$ and the other has either label x_r or x_r- or x_s* or x_s+ then $\varrho^-(s \leftarrow r)$ is a valid reduction.

Rule 4 ($\alpha = -$)

(a) If the left argument of $-$ has either label x_r or x_r- and the right argument is has either label x_s or x_s- or x_s+ or x_s* then is $\varrho^+(s \leftarrow r)$ is a valid reduction.

(b) If the left argument of $-$ has label $-x_r$ and the right argument has either label x_s or x_s- or x_r+ or x_s* then $\varrho^-(s \leftarrow r)$ is a valid reduction.

Rule 5
For any other combination of the labels and any other operation α no reduction is possible.

To enhance the intuitions associated with the labelling of syntax trees we justify a rule 4(a). Assume that the arguments of difference operation have the labels x_r- and x_s-. Then, it means that the first argument contains a set of x-values included in $\pi_x(r)$ and the second argument contains a set of x-values included in $\pi_x(s)$. Therefore it is valid to remove from s all rows whose x-values are not included in r, i.e. to compute reduction $\varrho^+(s \leftarrow r)$.

3 Computations

This section describes organisation of data integration subsystem and data integration algorithm based on the unified approach to query scrambling.

Integration of the partial results into the final answer is performed concurrently for all available arguments. It means that data integration operations are computed as soon as its arguments are available. Availability means that either transmission of an argument from a local site is completed or another computations that result with an argument are completed, or reduction that involved an argument is completed. If an argument is used to compute an operation then it can be simultaneously used to reduce the size of another argument. However, it cannot be reduced by another argument in the same moment of time.

The computations of operations and reductions are not pipelined. The rows produced by one operation are not immediately used by the next operation. The only exception from this rule are unary operators which are always pipelined to the outputs of the preceding operations.

Computation of either positive or negative reductions $\varrho(r \leftarrow s)$ over a common part of schema r and s always creates a join index [14]. The system is able detect such an index and use it to speed up the computations of the relevant join operations.

When r is used as an argument of operation α and as an argument of reductions $\varrho(t_1 \leftarrow r) \ldots, \varrho(t_n \leftarrow r)$ then it is read only once from disk storage.

The order of computations is determined by an order in which the results from the local sites arrive at a central site and by *computation graph* of data integration expression. Computation graph is a syntax tree

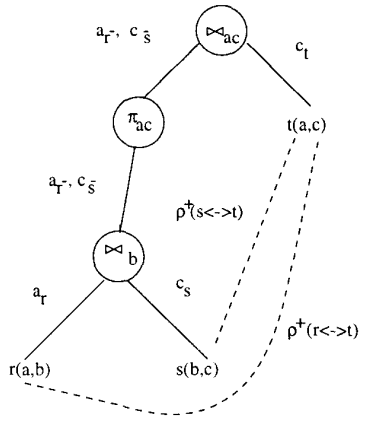

Figure 3. A sample computation graph

with the additional dashed edges representing the valid reductions, e.g. see computation graph in Figure 3.

Data integration is performed in the following steps.

Step 1: Construct a syntax tree of data integration expression, find all valid reductions, and construct a computation graph.

Step 2: When an argument r is available perform one of the following cases.

Case 1 If it is possible to compute an operation $\alpha(r, s)$ then find all arguments $t_1, \ldots, t_n$ then can be reduced by r. Then start the simultaneous computations of $\alpha(r, s)$ and the reductions $\varrho(t_1 \leftarrow r), \ldots, \varrho(t_n \leftarrow r)$. When the computations are finished make a result of $\alpha(r, s)$ and reduced $t_1, \ldots, t_n$ available.

Case 2 If it is possible to perform a valid mutual reduction $\varrho(r \leftrightarrow t)$ then find all arguments $t_1, \ldots, t_n$ that can be reduced by r. Then start the simultaneous computations of $\varrho(r \leftrightarrow t)$ and the reductions $\varrho(t_1 \leftarrow r), \ldots, \varrho(t_n \leftarrow r)$. When the computations are finished make $r, t, t_1, \ldots, t_n$ available.

Case 3 If it is possible to use r to reduce the arguments $t_1, \ldots, t_n$ then start the simultaneous computations of $\varrho(t_1 \leftarrow r), \ldots, \varrho(t_n \leftarrow t)$. When the computations are finished make $r, t_1, \ldots, t_n$ available.

Case 4 If r can be reduced by an argument t then start computation of $\varrho(r \leftarrow t)$. When the computation is finished make r, t available.

When operation $\alpha(r, s)$ is completed then both arguments r and s and all their reductions are re-

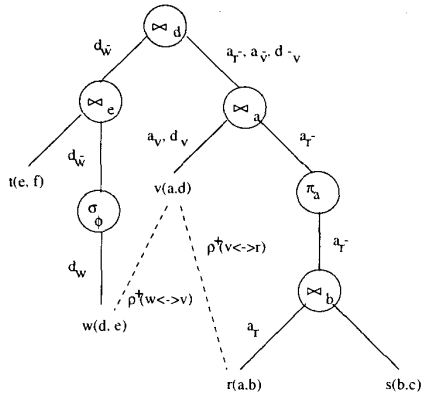

Figure 4. A computation graph of expression $(v(a,d) \bowtie_a (\pi_a(r(a,b) \bowtie_b s(b,c)))) \bowtie_d (t(e,f) \bowtie_e \sigma_\phi(w(d,e)))$

moved from a computation graph. A node labelled by α is replaced with a temporary result r_i.

When a reduction involving r and t is completed than the respective dashed edge is removed from a computation graph.

Any mutual reduction $\varrho(r \leftarrow t)$ is equivalent to a pair of reductions $\varrho(r \leftarrow t)$, and $\varrho(t \leftarrow r)$. If only one part of mutual reduction is computed then the respective dashed edge should change its label to the second reduction.

Step 3: Finish the computations when the final results is achieved. Otherwise, resume from Step 2.

As an example consider computation of a data integration expression $(v(a,d) \bowtie_a (\pi_a(r(a,b) \bowtie_b s(b,c)))) \bowtie_d (t(e,f) \bowtie_e \sigma_\phi(w(d,e)))$. Computation graph of the expression is given in Figure 4.

Let us assume that the arguments arrive in the following order: $r > v > s > w > t$. Execution trace of the algorithm is included in Table 3.

In order to get the final answer as fast as possible, the algorithm gives the highest priority to the operations of data integration expression. If it is impossible to do so then the next the algorithm attempts to compute a bilateral reduction. If an argument is available an operation of data integration expression is computed before any reductions.

Event	Action
r is available	no action
v is available	$v := \varrho^+(v \leftarrow r)$ and $r := \varrho^+(r \leftarrow v)$
s is available	$r_1 := \pi_a(r \bowtie s)$
	$r_2 := r_1 \bowtie v$
w is available	$w := \varrho^+(w \leftarrow v)$ and $r_3 := \sigma_\phi(w)$
t is available	$r_4 := t \bowtie r_3$
	$result := r_4 \bowtie r_2$

Table 3. A sample execution when $r > v > s > w > t$.

4. Comparison, further works, and conclusions

Unification of query scrambling with reductions and adoption of event based computation model shows a number of advantages over the approaches proposed earlier.

The original approach to query scrambling used a sequential model of computations in which the data integration operations were performed one at a time. Application of the same model to reductions will not benefit a lot because if an argument can be used to compute a data integration operation and reduction of another argument than it has to be read twice. Concurrent computation of more than one operation at a time solves this problem, e.g. simultaneous computation of $w := \varrho^+(w \leftarrow v)$ and $r_3 := \sigma_\phi(w)$ in the last example needs only one access to argument w.

The original approach to query scrambling considered the relational algebra expressions built of the join operations only. The reduction technique generalises this approach to any relational algebra expression. It is important to note that identification of the reductions by labelling of syntax trees leads to exactly the same results as the traditional heuristic scrambling method when it is applied join-only expressions.

In addition to elimination of the irrelevant rows, the reduction technique speeds up the further computations by construction of the join indices.

Concurrent computations solve the problem what to do when a reduction is initiated and unavailable argument arrives a moment later creating a dilemma whether the original order of operations should be restored or not. In the traditional approach, when an order is changed the system never returns to the previous one and leaves an argument waiting for its turn. In our approach any reduction is also considered as beneficial and it is never interrupted by arrival of an argument. However, when an argument is available, the system initiates another process to compute an op-

eration and/or reduction involving the argument. The only exception to this rule is when the new process attempts to use another argument which is currently reduced by the running computations.

Some problems remain to be solved. Our algorithm does not initiate the computations when only a part of an argument is available. Granulation of the arguments at the level of individual rows may solve this problem.

Our algorithm blindly assumes that whenever an argument is available the highest priority is assigned to an operation of data integration expression. It is not be the best solution in every possible case. For instance, it is well known that identification of the best order of join computations has an important impact on the performance. In our case a reduction computed before a data integration operation may eliminate a lot of rows and reduce time spent on the operation. To solve this problem estimation of the argument size and cost based analysis is needed.

The reduction technique is based on identification of "data coincidences" among the arguments of data integration expression. An interesting theoretical question is whether a set of rules (section 2 rules 1-4) governing this process is complete.

Transformation of a data integration expressions is yet another approach to query scrambling. If an argument is not available then it is usually possible to rewrite an expression such that the operations are performed in a different order. For example, an expression $(r(a,b) \bowtie_b (s(b,c) - t(b,c))$ is equivalent to $(r(a,b) \bowtie_b s(b,c)) \; t(b,c)$ where operation removes from the result of $r(a,b) \bowtie_b s(b,c)$ all rows that have the same (b,c)-values is in $t(b,c)$. Advantage of this method is that the partial results may be presented to a user before all computations are completed. If a user is interested in the presence and/or absence of certain data in the final result then there no need to wait for the third argument. An important important obstacle of this method is a fact that relational algebra is not finitely axiomatizable and the equivalence problem for certain expressions is not decidable [8].

Finally, an interesting problem is generalization of query scrambling to a level where interpretation of any computer program that needs input from a number of distributed sources is controlled by availability of input data.

References

[1] L. Amsaleg, M. Franklin, and T. Urhan. Dynamic query operator scheduling for wide-area remote access. *Journal of Distributed and Parallel Databases*, 6(8), July 1998.

[2] L. Amsaleg, M. J. Franklin, A. Tomasic, and T. Urhan. Scrambling query plans to cope with unexpected delays. In *Proceedings of the 4th Intl. Conf. on Parallel and Distributed Information Systems*, December 1996.

[3] J. Chudziak and J. Getta. On efficient query evaluation in multidatabase systems. In *Proceedings of the 4th Intl. Conf. on Parallel and Distributed Information Systems*, pages 46–54, June 1995.

[4] W. Du, M. Shan, and U. Dayal. Reducing multidatabase query response time by tree balancing. In *Proceedings of the 1995 ACM SIGMOD Intl. Conf. on Management of Data*, pages 293–303, 1995.

[5] C. J. Eghazy, K. P. Triantis, and B. Bhaskar. A query processing algorithm for a system of heterogenous distributed databases. *Distributed and Parallel Databases*, 4(1), June 1996.

[6] C. Evrendilek, A. Dogac, S. Nural, and F. Ozcan. Multidatabase query optimization. *Distributed and Parallel Databases*, 5(1), June 1997.

[7] J. Getta and M. Sedighi. Optimizing global query processing plans in heterogeneous and distributed multidatabase systems. In *Tenth International Workshop on Database and Expert Systems Applications (DEXA'99)*, pages 12–16, September 1999.

[8] T. Imielinski and W. Lipski. The relational model of data and cylindric algebras. *Distributed and Parallel Databases*, 28(1):80–102, 1984.

[9] S. Salza, G. Barone, and T. Morzy. A distributed algorithm for global query optimization in multidatabase systems. In *International Conference on Advances in Database and Information Systems*, pages 95–106, 1998.

[10] S. M. Sedighi. *Optimization of Query processing in Heterogeneous Distributed Multidatabase Systems.* PhD thesis, School of Information Technology and Computer Science, University of Wollongong, 1998.

[11] S. M. Sedighi and J. R. Getta. Optimization of query postprocessing in heterogeneous distributed multidatabase systems. In *Proceedings of the ISCA International Conference on Parallel and Distributed Computing Systems*, pages 626–632, 1996.

[12] D. K. Subramanian and K. Subramanian. Query optimization in multidatabase systems. *Distributed and Parallel Databases*, 6(2):183–210, March 1998.

[13] T. Urhan, M. J. Franklin, and L. Amsaleg. Cost-based query scrambling for initial delays. In *Proceedings of the 1998 ACM SIGMOD Intl. Conf. on Management of Data*, pages 130–141, June 1998.

[14] P. Valduriez. Join indices. *ACM Transactions on Database Systems*, 12(2), June 1987.

[15] Q. Zhu and P. A. Larson. Solving local cost estimation problem for global query optimization in multidatabase systems. *Distributed and Parallel Databases*, 6(4):373–420, Oct 1998.