Minimization of Resource Consumption for Multidatabase Query Optimization

Chiang Lee *Chih-Horng Ke* *Jer-Bin Chang*
Institute of Information Engineering
National Cheng-Kung University
Tainan, Taiwan(R.O.C)
leec@dblab.iie.ncku.edu.tw

Yaw-Huei Chen
Department of Mathematics and Science Education
National Chia-Yi Teachers College
Chia-Yi, Taiwan(R.O.C)
ychen@sun11.ncytc.edu.tw

Abstract

Due to the autonomy of individual databases, a multidatabase system (MDBS) has no control over the task execution within each database system. This complicates the issue of query optimization in a MDBS. Past researchers tackled this problem mainly by regenerating the cost model of each participating database systems. However, a completely autonomous participating database system does not own the information (data) as well as the mechanism (software) for converting data of one database to another to resolve the data type conflict problems that can occur in any MDBS environment. In addition, unpredictable factors that can drastically deviate the accuracy of a regenerated cost model (in the past research) come into play at unknown time. This confines the processing of an intersite operation (such as a join over two relations of different databases) to the MDBS only. The participating database systems will not be able to share the workload of the MDBS under this circumstance. Hence, how to minimize the consumption of system resources of the MDBS is an urgent problem. In this paper, we propose three scheduling algorithms that are used in the MDBS to reduce the processing cost of a multidatabase query. A major difference between our strategies and the past methods is that ours do not require the knowledge of the cost models of the participating databases.

1 Introduction

In a multidatabase system (MDBS), autonomous and heterogeneous database systems are integrated to serve for users who want to retrieve data from multiple data sources. As a user's query usually involves action of multiple participating database systems (PDBSs), processing of such a query is much more time-consuming than processing a local query in a single database system. The issue of query optimization in this environment is apparently a key issue to resolve in order to achieve efficiency for the MDBS. As each PDBS is autonomous, the MDBS has no control over the internal optimization decisions within each PDBS. This makes query optimization at the global level an extremely difficult task that deserves an extensive study.

In spite of the importance of the work, it has so far received little attention. As the query optimizer of each PDBS is a "black-box" to the MDBS, the main focus of the past research was on how to regenerate in the MDBS the cost model of the query optimizer of each PDBS. Du, Krishinamurthy, and Shan in [3] proposed a calibrating database which is synthetically created so as to reasonably deduce the coefficients of cost formulas of a cost model. Works in [13, 16, 5, 6] are other examples of having a similar approach. The effectiveness of all these schemes, however, heavily relies on the precision of the regenerated cost models of the PDBSs. Our study reveals, however, that many factors affect the execution cost and the processing time of a query in a PDBS, which makes query optimization using this (regenerating cost models) approach unrealistic if autonomy of the PDBSs cannot be compromised. For instance, one of the factors is that some commercial DBMSs automatically cache a large portion of a recently accessed relation in memory for some time and others only cache a very small portion or not cache at all for such data. For those DBMSs that do, a subsequent query, if it happens to involve the cached relation(s), is processed much faster than the previous query. The two dramatically different processing times for the same operation on the same relation could make the regenerated cost model of a PDBS (which is usually based on those times) significantly deviate from its real execution time. Global query optimizer using these inaccurate cost models could lead to making incorrect decisions. An even worse situation is that there is no way to predict how long such a relation will stay in the cache (since it depends on many other factors), which lets the regenerated cost model in the MDBS simply be untrustable.

Another factor which worsens the situation is that in many occasions multiple joins are performed in a PDBS. However, there is no way to predict the execution order of these join operations. Different execution order of the joins

could lead to a dramatically different execution time. None of the previously proposed methods covers this problem.

Recently, Du, Shan and Dayal [4] proposed a method for reducing query response time by using tree balancing. This method is based on an assumption that all PDBSs have the same processing rate for database operations. As all PDBSs are equal in terms of processing speed, it becomes unnecessary to estimate the cost models of the PDBSs in their method. Therefore, the issue of regenerating cost models in that paper is ignored. In reality, however, having to have cost models for optimization is still a major problem for their method to be valuable.

We have proposed methods for multidatabase query optimization at different levels that do not need the knowledge of cost models [8, 9]. In [8], we proposed a hyperrelational algebra for the MDBS. This algebra is worked as a query transformation and optimization tool at the algebraic level. In [9], we discussed multidatabase query optimization at the schema level, a level of optimization that has never be paid attention to. In this paper, we will propose an optimization strategy at a lower level than our previous work but still without the need of the cost models of the PDBSs. We observe that there exist various types of heterogeneity between databases [15, 8, 9]. In order to perform operations such as join between relations of two PDBSs, conversion of data must first be performed on these relations to resolve the representation discrepancy. The mapping functions (or mapping tables) of heterogeneous data and the mapping mechanisms (software) all reside in the MDBS. One may argue that the mapping tables for resolving data discrepancy can be sent to a PDBS to let the PDBS perform the tasks that are otherwise performed in the MDBS. However, we stress that not only the mapping tables but also the mechanisms (software) are needed to execute those conversions. Without the mechanisms, a PDBS is still unable to convert data of different types. All past research as mentioned earlier ignored this issue in the design of an intersite join optimization strategy.

Along this line of thought, we can identity two basic query execution strategies: (1) PDBSs perform the operations, and the MDBS converts the data, and (2) MDBS performs the operations and also data conversion. Let us use an example to illustrate the two strategies. Assume that $R \bowtie_1 S \bowtie_2 T$ is a multidatabase query, where R, S, and T belong to relations of distinct databases.

1. *PDBSs perform the operations :*
   Assume that the databases containing R, S, and T are PDBS$_R$, PDBS$_S$, and PDBS$_T$, respectively. A strategy of this category contains the following steps :
   (a) PDBS$_R$ sends R to MDBS for data conversion.
   (b) MDBS converts R to a format understandable to PDBS$_S$.
   (c) MDBS sends R to PDBS$_S$.
   (d) PDBS$_S$ performs $R \bowtie_1 S$.
   (e) PDBS$_S$ sends the result, letting it be RS, to MDBS for another conversion.
   (f) MDBS converts RS to a format understandable to PDBS$_T$.
   (g) MDBS sends RS to PDBS$_T$.
   (h) PDBS$_T$ performs $RS \bowtie_2 T$.
   (i) PDBS$_T$ sends the result, letting it be RST, to MDBS for conversion.
   (j) MDBS converts the result and sends it back to the user.

   Certainly, we can also send S to PDBS$_R$ to perform the join, or even perform $\bowtie_2$ before $\bowtie_1$. But, as long as joins are performed in PDBS$_S$, the required communications between PDBSs and the MDBS as well as data conversion to be performed in the MDBS are basically the same.
2. *MDBS performs the operations :*
   The same query will be executed in the following steps:
   (a) PDBS$_R$, PDBS$_S$, and PDBS$_T$ send R, S, and T, respectively, to MDBS simultaneously.
   (b) MDBS performs conversion on these relations.
   (c) MDBS performs $\bowtie_1$ and $\bowtie_2$.
   (d) MDBS sends the result to the user.

Obviously, the first strategy involves too much communication between each PDBS and the MDBS, causing serious overhead on query processing. The contention could be aggravated in today's applications that are accessed through the World Wide Web (WWW), a key feature desired by the users and implemented by most DBMS vendors into their current/next generation DBMSs. On the other hand, strategy 2 does not require as much communication. In addition, many joins performed in MDBS can be reduced to simply a merge operation if the relations (subquery results) sent out from PDBSs are sorted. Hence, strategy 2 can be a preferred choice. However, the MDBS's workload is heavy in this strategy. How to reduce query cost becomes crucial to the performance of a MDBS. In this paper, we propose a few optimization methods to alleviate the heavy workload problem with the MDBS. As each PDBS is like a processor in a parallel as well as in a distributed database, it is natural to ask whether our problem is the same as query optimization in parallel database systems, or in homogeneous distributed database systems? Also as all received relations are processed solely in the MDBS (as in strategy 2 above), how is this part different from a query optimization in a centralized database system is another question that needs explanations. We answer these questions in the following.

- *Different from query optimization in parallel database systems*
  In a parallel database system, each task (such as sort and join) can be assigned freely to any processors. In the MDBS environment under discussion, local tasks are processed locally and intersite operations are all processed in the MDBS. Hence, the environment under study is much more restrictive on the collaboration of PDBSs for task assignment.

- *Different from query optimization in distributed database systems*
  Similar to a parallel database system, a local system of a homogeneous distributed database system can process data sent from any other database, except that communication cost needs to be taken into account. No restrictions (due to heterogeneity) on task assignment to processors need to be considered. This is very different from the issue discussed in our environment.
- *Different from query optimization in centralized databases*
  A key difference is that in a centralized database hash-based join methods are normally considered more efficient than sort-merge join methods. In a multidatabase environment under study, however, since sorting the subquery results is performed in each PDBS rather than in the MDBS, it is desired to have them all sorted (on an attribute on which the next join is performed) before they are sent to the MDBS to alleviate the burden of and facilitate the global joins in the MDBS. Hence, it is expected that a new optimization algorithm which is able to utilize as many merge joins on sorted subquery results as possible. We will discuss this further in a later section.

From the above discussion, we see that the past query optimization methods cannot be directly applied to the new environment. New methods must be designed.

In this paper, we propose three query processing strategies for truly autonomous multidatabase environment. The first two strategies are based on traditional optimization concepts and skills and mainly designed for the purpose of comparing with the third strategy.

The third strategy is based on the sort-merge join concept. The traditional sort-merge join algorithm contains two major steps : *sorting* relations and *merging* them to get the join result. In our case, however, the sort part of an intersite join is accomplished in the PDBSs, that is, all subquery results output from the PDBSs are sorted, and the sort is on the attribute over which the next join (in the MDBS) is performed. Then, the sorted relations are sent to the MDBS to perform the merge part of the intersite join. As the sort part of a sort-merge join algorithm dominates the join cost, performing them in parallel in the PDBSs naturally speeds up the join process. The merge part is performed in the MDBS because a multidatabase operation usually involves resolution of data discrepancy [15, 11]. Our focus of this algorithm is on the merge part performed in the MDBS. The goal is to find maximum number of pairs of relations that can be merge joined in the MDBS. As merge is a simple process with a linear complexity, MDBS's workload is minimized by employing these merge joins. The advantages of our methods are (1) unlike the previous approaches, our strategies can proceed without the need of regenerating local cost models, (2) the MDBS workload is minimized, and (3) much simpler and easier to implement in comparison with those complex cost model regeneration methods proposed before.

The rest of the paper is organized as follows. In Section 2, we discuss a categorization of MDBS environments and the environment to which this paper is dedicated. Also, we give the assumptions made in this paper. In Section 3, we present two optimization strategies which do not involve PDBSs. Then in Section 4, we present another strategy that properly utilizes PDBSs' processing power for query processing. The performance study and comparisons of these methods are discussed in Section 5. Finally, conclusions and our future work are depicted in Section 6.

2 Environments and Assumptions

2.1 Environments

The DBMSs that participate in a multidatabase environment can be classified to three categories [3]:

- *Proprietary DBMSs*: The participating DBMSs provide all the relevant information on cost functions and database statistics.
- *Conforming DBMSs*: The participating DBMSs provide database statistics but are incapable of divulging the cost functions.
- *Non-Conforming DBMSs*: The participating DBMSs are incapable of divulging either the database statistics or the cost functions.

Past effort mainly focused on the first two types of multidatabase environments [3, 13, 16, 4, 5, 6]. Du *et al.* in [3] argued that the synthetic database calibration technique for deducing the logical cost model in a Conforming DBMSs environment is extensible to the Non-Conforming DBMSs environment. We suppose, however, that even though their calibrating technique is extensible to the Non-Conforming environment, the MDBS is still impossible to utilize the PDBSs to process an intersite join because data type conflicts have to be resolved by the MDBS.

Our effort in this paper is dedicated to the *Non-Conforming DBMSs* environment. The reasons are, first, it is the least studied environment up to now. All past query optimization methods are not designed for this environment. Second, it is the most realistic environment considering the fact that almost all DBMSs today either have had or will be supplied with an accessing interface to the world wide web. There are also vendors utilizing *data warehousing* and *digital library* techniques to provide users with various data sources. All these sources are basically heterogeneous, same as in the Non-Conforming DBMSs environment. Both of the other two environments discussed in [3] require a major modification/enhancement to the core modules of the existing DBMSs.

2.2 Optimization Goal and Assumptions

Normally there are two optimization goals in a database system. One is the *response time* and the other the *query execution cost*. In our environment, we minimize the consumption of MDBS system resources so that it will not be easily overloaded during query processing. Therefore, *our goal is to minimize the query execution cost in the MDBS* so that the MDBS can serve a maximum number of concurrent users. Minimizing query response time is not the goal because query response time is reduced at the cost of using more system resources. This is unsuitable in our environment.

A few assumptions are made in this research. The first assumption is that the autonomy of a PDBS should not be violated or compromised for any reasons. The MDBS can only interact with a PDBS and utilize a PDBS's processing power under this restriction. Through this assumption, it is ensured that a running database system is able to participate a multidatabase without the need of modifying any part of its system code for the query optimization purpose.

Our second assumption is that all local operations (i.e., involving relations of a single database) of a query must be performed locally. A multidatabase (global) query after compilation is decomposed into subqueries, each being processible in a PDBS. The processed result is then transmitted to the MDBS. Certainly, it is possible that the result size of a local operation (such as a join) becomes larger than its input size such that it seems to be better to send its input relations, instead of its result, to the MDBS and let the MDBS perform the (join) operation so as to reduce communication overhead. We do not consider this approach because our major concern here is that the MDBS could easily be overloaded if all locally processible tasks are sent to the MDBS. Since our optimization goal is to minimize the query execution cost in the MDBS, processing local operations locally becomes a natural choice.

3 Two Scheduling Strategies Not Involving PDBS

Figure 1 is a "reduced" join graph of a multidatabase query in a MDBS. By reduced graph, we mean that operations and relations of a local query (subquery) are represented by only one node, which is in fact the result of a local query. Hence, each node is a result relation from a PDBS. In this particular example, the number of PDBSs involved in this mutlidatabase query is 10. Each edge between two nodes stands for a join on the two corresponding relations. Each node is labeled by the name of the relation and each edge labeled by the join attribute. Formally, let us refer to such a graph as $G(V, E)$, where V and E are the sets of nodes and edges, respectively. Let the number of nodes (relations) and edges (joins) be $|V|$ and $|E|$, respectively.

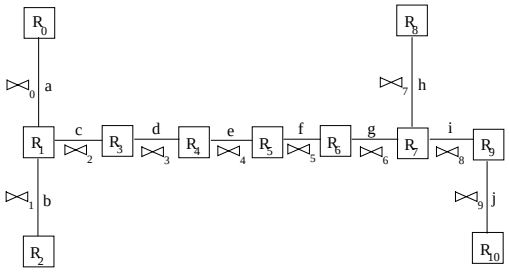

Figure 1. A join graph.

3.1 The First-Come-First-Serve (FCFS) strategy

A straightforward method of joining the incoming relations in the MDBS is to use a first-come-first-serve (FCFS) strategy, meaning whichever relations that are received first by the MDBS are joined first in the MDBS. For instance, two joins $R_1 \bowtie R_2$ and $R_3 \bowtie R_4$ are involved in a query. If R_1 and R_2 are received in the MDBS earlier than R_3 and R_4, MDBS will perform $R_1 \bowtie R_2$ before $R_3 \bowtie R_4$. This strategy represents a case that we do not have any knowledge at all about the possible execution order of joins. MDBS will perform a join on whichever relations that have arrived the MDBS. Intuitively, this is a good strategy because we normally have no control over the traffic of a network. A predetermined sequence of joins may not give a good performance if a relation that is scheduled to be joined early is blocked by the traffic of a network. We studied the performance of this strategy and the results will be presented in the next section. In the following, we give a formal presentation of the FCFS strategy. The complexity of this strategy is O($|E|$), where $|E|$ is the number of edges of a query graph $G(V, E)$, because each edge (i.e., join) need only be visited once in the algorithm.

FCFS strategy :
Input : a join graph $G(V, E)$;
Output : a join sequence Q;
Repeat until $|V|$ = 1 ;
Begin
Choose $R_i \bowtie R_j$ such that $\overline{R_i R_j} \in E$ and R_i and R_j have been arrived MDBS (if more than one joins can be performed (their operands are ready), an arbitrary one is selected);
Add $R_i \bowtie R_j$ into Q ;
Update graph G by merging nodes R_i and R_j into $R_{min(i,j)}$;
End

3.2 The Greedy Scheduling (GRS) strategy

The GRS strategy is an iterative algorithm that selects in each iteration the join that has a lowest join selectivity factor

at the current stage. This algorithm is named so because it produces the least amount of data at every join stage. The details of the algorithm is listed in the following.

GRS strategy :
Input : a join graph $G(V, E)$;
Output : a join sequence Q;
Repeat until $|V|$ = 1;
Begin
Choose $R_i \bowtie R_j$ such that $\overline{R_i R_j} \in E$ and it has the lowest join selectivity factor at the current stage; (if more than one such joins are available, an arbitrary one is selected;)
Add $R_i \bowtie R_j$ into Q;
Update graph G by merging nodes R_i and R_j into $R_{min(i,j)}$;
End

A number of criteria could be used to select a join [2] in a greedy multijoin algorithm. Three frequently seen criteria include:

1. **min(JS)** : select a join with the minimum join selectivity factor (producing *relatively* the least amount of data, or $|R_i \bowtie R_j|/(|R_i| * |R_j|)$ being minimum).
2. **min(**$|R_i| + |R_j|$**)** : select a join whose total amount of input data is minimum.
3. **min(**$|R_i| * |R_j| * JS$**)** : select a join that produces the least amount of data, i.e., $|R_i \bowtie R_j|$ being minimum.

These criteria aim at minimizing the intermediate relation sizes so as to achieve a lower total execution cost. Each of them has its strength and weakness. The third criterion may look attractive at the first sight. However, the smallest join result (i.e., $|R_i \bowtie R_j|$) may simply be incurred by the join of two small relations. The relative gain $|R_i \bowtie R_j|/(|R_i| * |R_j|)$ may be much greater than that of the first criterion. A performance study of these criteria has been conducted in [12] and the result shows that none of them outperforms the others under all circumstances. In this paper, we simply adopt the first criterion, min(JS), in the GRS strategy.

The complexity of the GRS strategy is O($|E|^2$), where $|E|$ is the number of edges, because it needs $|E|$ iterations and each iteration checks on at most $|E|$ edges.

4 PDBS Sharing Workload With the MDBS

4.1 How to utilize PDBS processing power?

The hash, the sort-merge, and the nested-loop join algorithms are the most frequently used algorithms in traditional database systems. Especially, the hybrid hash join algorithm has been considered the fastest join algorithm among all. Now the question is that in the Non-Conforming DBMSs environment can these algorithms be similarly implemented and still as efficient as in a centralized database environment? We briefly discuss this issue in the following.

Assume that R_A(A1, A2, A3) and R_B(B1, B2, B3) are relations of different databases and a multidatabase query is to join these two relations on A1 and B1 (i.e., R_A.A1 = R_B.B1).

To implement a sort-merge algorithm in our environment, only one query is needed to send to each PDBS. The query requires that the resulting relation be sorted in order on the join attribute. In the above example, a query sent to database A will be like this.

Select *
From R_A
Order By R_A.A1

The same query but asking for a sorted order on attribute B1 will be sent to database B. The results obtained from these two databases can then be easily merged by the MDBS to obtain the join result. As merge is a simple and linear process, the workload added onto the MDBS is extremely light.

In [10], we also discussed the implementation of the other two algorithms, the hash join and the nested-loop join algorithms. The discussion is too lengthy to be presented here. We only summarize the result of our investigation in the following table. It is apparent that the sort-merge join is the most suitable algorithm for joins in a mutlidatabase environment, and therefore it is adopted as the global join algorithm in this research.

Algorithms	Consumed PDBS processing power	Consumed MDBS processing power
Hash join	Overutilized	Medium
Sort-merge join	Medium	Low
Nested-loop join	Low	Very High

4.2 The Maximum Merge Scheduling(MMS) strategy

The MDBS's task is to determine the maximum number of pairs of sorted relations that can be merged (i.e., join) and the instruct the corresponding PDBSs to sort their results. If we take the join graph in Figure 1 as an example, the maximum number of pairs would be five, as those pairs of relations circled by dotted ovals shown in Figure 2. Certainly, there may be other choices on the pairs of relations, such as choosing R_1 and R_2 as a pair rather than the pair R_0 and R_1. Based on these matching pairs, the MDBS asks the PDBSs to sort their results on the given join attributes. For instance, the MDBS will ask the PDBS owning R_0 to sort on attribute a and same to the PDBS having R_1. In this way, there will be five joins simplified to a simple "merge" process in the MDBS, greatly reduce the burden of the MDBS. In our algorithm (to be presented shortly), we group those larger relations as a pair when we have multiple choices because by sorting larger relations, the PDBSs share more workload of the MDBS.

An existing algorithm, named the *maximum matching algorithm* [1], can be used to find the maximum number of matching pairs of a given graph. In this maximum matching algorithm, a graph G is called **1-regular** if every vertex of G has degree 1, where the **degree** of a vertex is the number of vertices adjacent to this vertex. Let adjacent edges be those having a same terminal vertex. A **matching** of a graph G is a 1-regular subgraph of G, that is, *a subgraph containing a collection of nonadjacent edges*. The **cardinality** of a matching represents the number of nonadjacent edges within the matching. A matching of the maximum cardinality of graph G is called a **maximum matching** of G. For an example, Figure 2 shows a maximum matching of the join graph. The pairs of nodes that are circled by dotted ovals are the matching pairs. As this maximum matching algorithm is not difficult to infer based on the above description, we do not list here the details of the algorithm. Readers may refer to [1] for details.

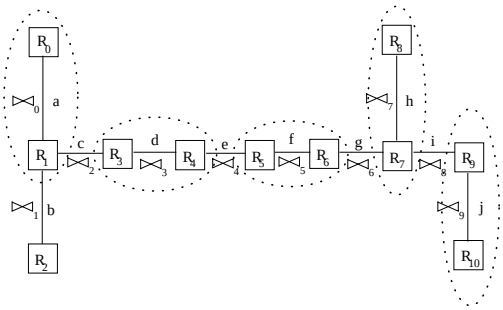

Figure 2. One of the maximum matchings found using the maximum matching algorithm.

This existing algorithm, however, does not work while there are more than one join on the same attribute. Figure 3 gives another example in which the relations are the same but the join attributes are different from those of Figure 1. Obviously the maximum matching for our purpose should be the two groups as shown in the graph, because it allows seven joins to be implemented by a sort-merge join rather than five by using the maximum matching algorithm. Our maximum merging strategy (MMS) is designed to work for such general query graphs.

We take a simple and yet effective approach to maximize the number of matching pairs in order to reduce the complexity of our algorithm. The basic idea is that we first use the existing maximum matching algorithm to find the maximum number of relation pairs, and let this graph be G. Then, we identify the relations connected through the same join attribute, such as a dotted oval in Figure 3. For each set, say C, of these relations, we check whether benefit (more matching pairs) is gained if the part of G corresponding to C is replaced by C. If it is beneficial, then they are replaced by C. If

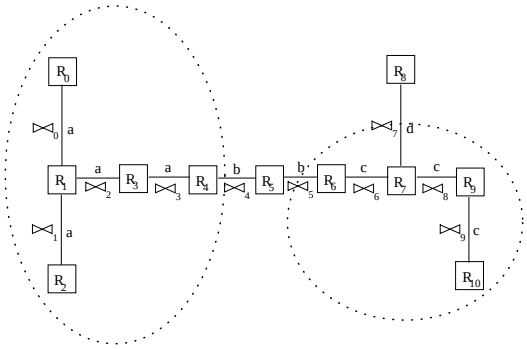

Figure 3. The same join graph with different set of join attributes.

the number of matching pairs does not change by replacing them with C, then we check whether the total intermediate result size after the replacement will decrease. If it decreases in size, then the replacement is still adopted because it helps to reduce the workload of the MDBS. The same process continues until all such sets are examined. In order not to miss a large set (i.e., with more nodes), this process should start from the largest set and proceed in a descending order of the set size (number of nodes in a set) because selecting one set of matching relations might affect the selection of a set that is considered later.

Let us illustrate our idea by using the example in Figure 3. Assume that the maximum matching pairs found by using the maximum matching algorithm is the one shown in Figure 2, but notice that their join attributes should be the same as those in Figure 3. There are three sets of relations connected through a same attribute. In descending order of the size, they are the set $A = \{R_0, R_1, R_2, R_3, R_4\}$, the set $C = \{R_6, R_7, R_9, R_{10}\}$, and the set $B = \{R_4, R_5, R_6\}$, in which set B is not circled because eventually it is not selected in our result. For convenience, let us denote the set of relations $\{R_i, R_j, \cdots, R_k\}$ that are connected through the same attribute as $\overline{\overline{R_i R_j \cdots R_k}}$. We start from set A, replace the matching pairs $\overline{\overline{R_0 R_1}}$ and $\overline{\overline{R_3 R_4}}$ by the set of matching relations $\overline{\overline{R_0 R_1 R_2 R_3 R_4}}$ and check whether the number of edges increases. Since in this case the number is increased from two to four, we adopt the new set. Next, we consider the set C. If the matching pairs $\overline{\overline{R_5 R_6}}$, $\overline{\overline{R_7 R_8}}$ and $\overline{\overline{R_9 R_{10}}}$ are replaced by the set $\overline{\overline{R_6 R_7 R_9 R_{10}}}$, the number of edges is still three. So whether they will be replaced by the new set is determined by the intermediate result size. That is, assuming that the result relation of $R_i \bowtie R_j \bowtie \ldots \bowtie R_k$ is denoted as $R_{ij\ldots k}$, and the size of relation R denoted as $|R|$. Then, the total size of the intermediate result relations (for only the part of involved relations $R_5, R_6, R_7, R_8, R_9, R_{10}$) before the replacement is $|R_{56}| + |R_{78}| + |R_{910}|$, and that

after the replacement is $|R_5|+|R_{67910}|+|R_8|$. If the size become smaller, then the replacement is adopted. Otherwise, it is rejected. If C is adopted, then B need not be checked because a matching of a graph must be 1-regular. Otherwise, the set B is examined similarly. We list our algorithm details in the following.

Maximum Merging Strategy:
Input: join graph $G(V, E)$;
Output: join strategy Q;

Let $|V|$ be the number of vertices of G, and $|E|$ be the numer of edges of G;
We denote the join result of $R_1 \bowtie R_2 \cdots R_k$ as $R_{12\cdots k}$, and the size of R as $|R|$;
Use the maximum matching algorithm to find for G a set of matching pairs S_G with the number of matching pairs = M;
Assume that there are I sets of connected vertices within each set the edges have the same label (i.e., join attribute) and the number of vertices in each set ≥ 3, where $0 \leq I \leq \lfloor |V|/3 \rfloor$;
Each of these sets of vertices is a subgraph of G, and we denote such a subgraph as $G_i(V_i, E_i)$, where $0 \leq i \leq I$;
Let $V_i = \{R_{i_1}, R_{i_2}, \cdots, R_{i_{|V_i|}}\}$;
Sort G_i's in descending order (i.e., G_1 is the largest set);
Let $Q = \phi$;
For $i = 1$ to I Do
Begin
$G'(V', E') = G(V, E) - G_i(V_i, E_i)$;
Find maximum matching pairs for G' using the maximum matching algorithm; let the number of matching pairs be M';
CASE
- $M < M' + |E_i|$
  Begin
  $G(V, E) = G'(V', E')$;
  $M = M'$;
  $Q = Q + \{\overline{\overline{R_{i_1} R_{i_2} \cdots R_{i_{|V_i|}}}}\}$; /* $R_{i_j} \in V_i$ */
  End
- $M = M' + |E_i|$
  Begin
  If $\Sigma_{\forall h,k}|R_{hk}| + \Sigma_{\forall f,g}|R_{fg}| > \Sigma_{\forall h}|R_h| + |R_{i_1 i_2 \cdots i_{|V_i|}}|$, where the indices h, k, f, g refer to $\overline{\overline{R_h R_k}}, \overline{\overline{R_f R_g}} \in S_G$ and $R_h \in V'$, and $R_k, R_f, R_g \in V_i$
  /* $\overline{\overline{R_h R_k}}$ is a matching pair of G that is broken by the selection of G_i; $\overline{\overline{R_f R_g}}$ is a matching pair of G residing in G_i. */
  then /* accept G_i */
  $G(V, E) = G'(V', E')$;
  $M = M'$;
  $Q = Q + \{\overline{\overline{R_{i_1} R_{i_2} \cdots R_{i_{|V_i|}}}}\}$;
  End
- $M > M' + |E_i|$
  Begin
  Do nothing;
  End

End-of-For
Find matching pairs of $G(V, E)$ using the maximum matching algorithm and let $S_G = \{\overline{\overline{R_{11}R_{12}}}, \overline{\overline{R_{21}R_{22}}}, \cdots, \overline{\overline{R_{m1}R_{m2}}}\}$ be the set of matching pairs;
/* $G(V, E)$ might have been updated so that a recalculation is still needed. */
$Q = Q + S_G$;
/* End of algorithm */

At the end of this algorithm, the relations that will be merge-joined are determined. The results of these merge joins, however, could be unsorted on the attribute to be joined next. In general, these later stages of joins could have either one or two of the input relations being unsorted on the join attribute. We process these joins based on two simple rules:

1. if one of the input relations is sorted on the join attribute, then perform an *order-preserving hash sort* on the unsorted relation and then do the merge join. We term this join a *semi-merge join* to distinguish it from a merge join.
2. if both of the relations are unsorted, then perform a hybrid hash join.

The reason why we employ an order-preserving hash sort in rule 1 is as follows. In order to minimize the number of disk accesses in the MDBS, when one of the input relations is already sorted, the second relation is only sorted at the block (bucket) level. This means that after the order-preserving hash sort, if B_0, B_1, $\cdots$, B_n are the produced blocks, then records $b_0 \in B_0$, $b_1 \in B_1$, $\cdots$, $b_n \in B_n$ are in order, but records within each block are not in order. Then, joining records of two corresponding blocks can be performed in memory. This is especially efficient for today's systems in which a large memory is usually provided. By sorting only at the block level, the sorting process is accomplished in one scan of the relation, which significantly reduces the number of disk accesses. Since one of the input relations needs to be scanned and order-preserving-hash-sorted, the cost of such a "semi-merge join" is higher than that of a merge join, i.e., *cost*(*merge join*) < *cost*(*semi-merge join*). A hash join on both unsorted relations, on the other hand, requires a scan and hash on both relations and then joins them. Hence, it is the most costly operation. Overall, the relationship of the costs of these three operations are *cost*(*merge join*) < *cost*(*semi-merge join*) < *cost*(*hash join*).

As all results sent from PDBSs are sorted, there can be maximally $\lceil N/2 \rceil$ joins reduced to a simple merge process (i.e., merge join) for a global query of N joins, even if all join attributes are distinct attributes. If there are multiple joins on the same attribute, then this number can be even higher. Since a maximum number of merge joins is desired, a *bushy query tree* is in general better than a *left* or *right-deep tree* [14, 2]. In the performance study, we generate a set of queries to compare the performance of using different processing strategies. For the MMS, we always build for each query a bushy query tree such that a maximum number of merge joins can be employed.

5 Performance Study

5.1 Query model

In order to compare the performance of the proposed strategies, we design a model that generates queries in a random manner. Each query in our model is a graph where a node is a relation and an edge is a join. Local operations are processed locally in the PDBSs. Hence, each node of a generated query graph represents a result relation output from a PDBS. The query can be either non-cyclic or cyclic (i.e., loops may be formed in a query). There are two key tasks in generating such graphs. One is to model the joins on pairs of relations and another to model the join attributes as given in the following. Through these steps, 1000 join queries are generated for doing the experiments.

The steps for modeling pairs of relations

Let the number of relations involved in a query be N_r.

1. Create a node as the initial node.
2. Denote the initial node as R_i.
3. R_i may connect to $1 \sim 4$ new nodes according to the probabilities: 1 node (1/2), 2 nodes (1/4), 3 nodes (1/8), and 4 nodes (1/8), where X in () is the probability. The nodes that R_i has been connected to are included in the generated number. That is, if p is the generated number (which may be 1, 2, 3, or 4 nodes) and if R_i has been connected to p' nodes before this step, then the actual number of new nodes added to connect to R_i is $p - p'$ if $p > p'$, or 0 if $p \leq p'$. However, the number of nodes of each query should not be more than N_r. When this step is done, mark R_i.
4. If the number of nodes of a query is equal to N_r, go to Step 5.
   If all nodes of this join graph are marked but the number of nodes is smaller than N_r, select a new node from this join graph as the initial node and go to Step 2.
   Otherwise, select an arbitrary unmarked node and denote it as R_i. Go to Step 3.
5. For each pair of nonadjacent nodes in this join graph, an edge is generated to connect them based on a probability of 1/10.

From Step 1 to Step 4 this model creates tree queries. Step 5 is to generate cyclic queries based on a certain probability.

The steps for modeling join attributes

We say that two edges are adjacent if they have a same terminal node.

1. Assign different join attributes to all edges. Let all edges be unmarked.
2. Randomly select an edge as an initial edge and denote it as E_i.
3. The join attribute of an unmarked edge, say E_j, that is adjacent to E_i may be changed to the same join attribute as E_i based on a probability of 1/4.
   If E_j's join attribute is changed to be the same as E_i's join attribute, then mark E_j. After all unmarked edges adjacent to E_i have been gone through this process, marks E_i.
4. If all edges of a join graph are marked, Protocol stops. Otherwise, arbitrarily select an unmarked edge and let it be E_i. Go to Step 3.

Parameters used in our study and their default values are summarized in the following table. These values are standard values and have been used in many other works [14, 7]. Also similar to other research, we assume that the memory size is large enough to contain at least a pair of corresponding blocks of the two relations to be joined so as to simplify our simulation task. The execution cost is measured in the number of disk accesses. CPU cost is ignored since it is normally much smaller than disk I/O cost.

Symbol	*Meaning (default value)*
N_r	The number of relations ($5 \sim 30$)
$\|R\|$	The number of tuples of relation R (10^6)
T_s	The size of each tuple (200 bytes)
P_s	The size of each page (4 Kbytes)
JS	The join selectivity factor ($2*10^{-7} \sim 2*10^{-6}$)

5.2 Results and Discussion

In our experiments, the size of each relation received by the MDBS is assumed to be 10^6. Hence, if JS is equal to 10^{-6}, the result of a join will have a same size as the source relations, i.e., $|R \bowtie S| = |R| = |S|$. In order to examine the effect of join selectivity factor on the performance of the strategies, we divide JS into three ranges.

1. $2*10^{-7} \sim 8*10^{-7}$: represents a **low** join selectivity factor. A join in this case will produce a result of only $20\% \sim 80\%$ of its input size.
2. $8*10^{-7} \sim 1.2*10^{-6}$: stands for a **medium** join selectivity factor. Result size of such a join falls between 80% to 120% of the input size.
3. $1.2*10^{-6} \sim 1.8*10^{-6}$: stands for a **high** join selectivity factor. Result size of such a join is increased by more than 20 percent (i.e., 120%) of the input size up to 180%.

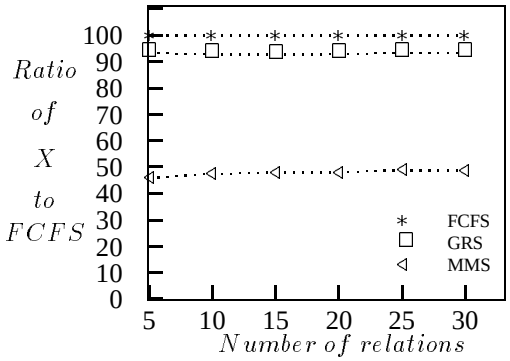

Figure 4. Comparison of strategies at low JS.

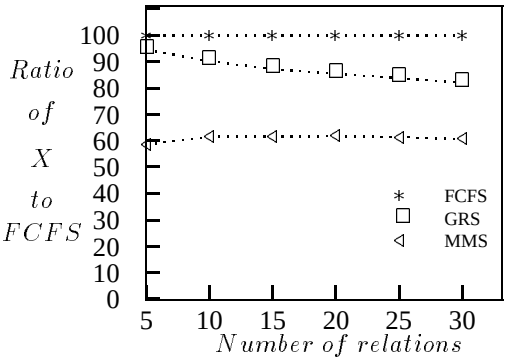

Figure 5. Comparison of strategies at medium JS.

5.2.1 Low join selectivity

We first examine the result when JS is low. Figure 4 shows the performance ratio of all strategies under a low JS. The performance ratio is obtained by dividing the average execution cost of the generated queries of a proposed strategy by that of the FCFS strategy. 'X' in the label of the vertical axis of the figure means one of the three strategies. Hence, the curve for FCFS is always 100%, and X's curve below FCFS's curve means that X is better than FCFS and above FCFS's curve means X is worse.

As we expected, FCFS is the worst because it does not utilize any knowledge about the join selectivity factors in scheduling the join operations. GRS is better than FCFS by only about 10%. MMS is the best and outperforms the other two consistently by about 50%. In other words, it consumes only about half of the computing power that is required by the other two strategies. This is mainly because of the reduction of a join to simply a merge process in that strategy. The consistent benefit margin of MMS over the others indicates that it is independent of N_r, the number of relations.

5.2.2 Medium join selectivity

Figure 5 presents the result of medium join selectivity. In this experiment, we found that GRS performs much better than FCFS at large N_r. This is because the join selectivity factor is larger in this experiment. The greedy strategy becomes more effective when the number of relations to be joined increases. The MMS shows a consistent 40% improvement over the FCFS strategy and also a significant enhancement over the GRS.

5.2.3 High join selectivity

Figure 6 gives the result of high JS. In this case, both GRS and MMS outperform FCFS by a great margin, especially when N_r is large. This is because the JS is high such that the output size keep increasing during the joins. Hence, if the number of joins increases, the benefit over FCFS increases too, enlarging the difference between FCFS and the other two strategies. Also we notice that the performance of GRS is getting closer to that of MMS at large N_r. This indicates that GRS is actually a good choice too under the condition that JS is high because this algorithm is simple, easy to implement, and do not require the PDBSs to sort their output relations. The only limitation is that it is confined to a high JS.

In summary, the MMS strategy is the best and outperforms the other two by a significant margin under all circumstances. GRS can be a second choice, in an environment where implementation complexity is a consideration and the join selectivity factor is known to be large.

6 Conclusions and Future work

Conflict resolution on values of relations from different databases is often required in performing intersite database operations because of the heterogeneity of data between autonomous databases. Previous researches ignored this heterogeneity in their query optimization algorithms. We argued that the MDBS has to perform intersite operations because only the MDBS has the required conflict resolution information and mechanisms. As the MDBS can easily become a bottleneck while serving for multidatabase users/applications, we proposed some algorithms which minimize the consumption of system resources such that the MDBS bottleneck problem can be alleviated. The advantages of our methods are (1) the knowledge of cost models of the PDBSs is not needed such that there is no need to regenerate all PDBSs' cost models in the MDBS (this is

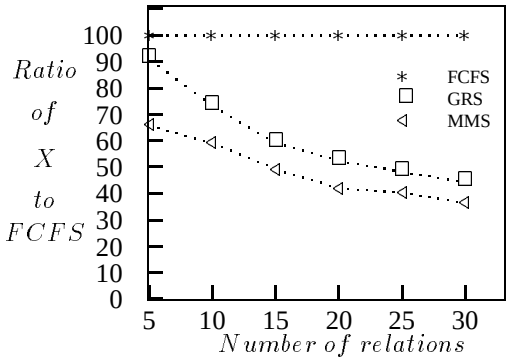

Figure 6. Comparison of strategies at high JS.

however required in all previous works), (2) our methods reduce the load of the MDBS and even distribute part of the task (sort) to the PDBSs, and (3) much simpler and easier to implement comparing to those complex cost model regeneration methods proposed before.

Our future work is to combine this work with our previous study on optimization at the schema level [9] and the algebra level [8] to compose a complete query compilation/optimization framework and implement these modules in a web-based multidata source sharing system.

References

[1] Gary Chartrand and Ortrud R. Oellermann, *Applied and Algorithmic Graph Theory*, McGraw-Hill, 1993, ISBN 0-07-112575-2, pp. 182.

[2] Ming-Syan Chen, Philip S. Yu, and Kun-Lung Wu, "Optimization of Parallel Execution for Multi-Join Queries", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, No. 3, June 1996, pp. 416-428.

[3] W. Du, R. Krishinamurthy, and M. Shan, "Query Optimization in a Heterogeneous DBMS", *Proceedings of the 18th VLDB Conference*, Vancouver, British Columbia, Canada, 1992, pp. 277-291.

[4] W. Du, M. Shan, and U. Dayal, "Reducing Multidatabase Query Response Time By Tree Balancing", *ACM International SIGMOD Conference*, California, 1995, pp. 293-303.

[5] C. J. Egyhazy, K. P. Triantis, and B. Bhasker,"A Query Processing Algorithm for a System of Heterogeneous Distributed Databases," *Distributed and Parallel Databases*, Vol. 4, No. 1, Jan. 1996, pp. 49-79.

[6] C. Evrendilek, A. Dogac, S. Nural, and F. Ozcan, "Multidatabase Query Optimization," *Distributed and Parallel Databases*, Vol. 5, No. 1, Jan. 1997, pp. 77-114.

[7] Kien A. Hua, Chiang Lee and Chau M. Hua, "Dynamic Load Balancing in Multicomputer Database System Using Partition Tuning," *IEEE Transactions on Knowledge and Data Engineering*, Vol.7, No.6, December 1995, pp.968-983.

[8] Chiang Lee and Ming-Chuan Wu,"A Hyperrelational Approach to Integration and Manipulation of Data in Multidatabase Systems," *International Journal of Cooperative Information Systems*, Vol. 5, No. 4, 1996, pp. 395-429.

[9] Chiang Lee and Chia-Jung Chen,"Query Optimization in Multidatabase systems considering Schema Conflicts," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, No. 6, Nov. 1997.

[10] Chiang Lee, Chih-Horng Ke, Jer-Bin Chang, and Yaw-Huei Chen,"Minimization of Resource Consumption for Multidatabase Query Optimization," *Technical Report # 87-03*, Department of Computer Science, National Cheng-Kung University, Taiwan, 1998.

[11] Ee-peng Lim, Jaideep Srivastava, Satya Prabhakar, and James Richardson, "Entity Identification in Database Integration", *Proceedings of 9th IEEE International Conference on Data Engineering*, Vienna, Austria, Apr. 1993, pp. 294-301.

[12] Hongjun Lu, Ming-Chien Shan, and Kian-Lee Tan, "Optimization of Multi-Way Join Queries", *Proceedings of the 17th Interational Conference on VLDB*, Barcelona, Sep. 1991, pp. 549-560.

[13] Hongjun Lu, Beng-Chin Ooi, Cheng-Hian Goh, "Multidatabase Query Optimization : Issues and Solutions", *Proceedings of 3th IEEE International Workshop on Research Issue in Data Engineering : Interoperability in Multidatabase Systems (RIDE-IMS'93)*, Vienna, Austria, Apr. 1993, pp. 137-143.

[14] Hongjun Lu, Beng-Chin Ooi, and Kian-Lee Tan, *Query Processing in Parallel Relational Database Systems*, IEEE Computer Society Press, 1994.

[15] Won Kim and Jungyun Seo, "Classifying Schematic and Data Heterogeneity in Multidatabase Systems", *IEEE Computer*, Dec. 1991, pp. 12-18.

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