# On Global Multidatabase Query Optimization

*Hongjun Lu* *Beng-Chin Ooi* *Cheng-Hian Goh*
Department of Information Systems and Computer Science
National University of Singapore
10 Kent Ridge Crescent, Singapore 0511
Internet: {luhj,ooibc,gohch}@iscs.nus.sg

*Extended Abstract*

## 1 Introduction

*Multidatabase management systems (MDBMS)* [6] enable data sharing among heterogeneous local databases (called *component databases*) and thus provide interoperability required by diverse applications. Among the research topics in multidatabase systems, few work has been reported on global query optimization [3, 4] compared to other topics such as schema integration. We suggest that there are several reasons why this is so. First, global query optimization in multidatabase systems is a complex problem. Second, the potential returns, in terms of both academic value of this research as well as possible performance gain from optimization, are not clear.

In this extended abstract, we claim that global query optimization is necessary for high performance MDBMS as in the case of conventional distributed database systems. However, contrary to what is commonly assumed, query optimization in multidatabase systems encompasses a number of additional issues which arise from autonomy and heterogeneity of component databases. To circumvent these difficulties, we extend the existing model for distributed query optimization to operate in the multidatabase context. Finally, we also outline the design of a multidatabase query optimizer.

## 2 Motivation

To date, many researchers have remained skeptical of the value of global query optimization in the context of multidatabase systems because autonomy of the component databases implies the MDBMS has little control over the actual query processing performed by component databases. We advocate that there is in fact much scope for research in this area.

Consider this query,

```
select *
from T1, T2, T3
where T1.c1 = T2.c2
and T2.c3 = T3.c4
and T1.cx = 5
and T2.cy = 7
and T3.cz = 6;
```

where T1, T2 and T3 are stored in three different database systems D1, D2, and D3 respectively. To process a multidatabase query such as this, subqueries will have to be formed and distributed to the respective component database systems. These subqueries are typically optimized and executed as user queries by the corresponding component DBMS. Note that the subqueries may have to be executed in certain order and the intermediate results will also need to be transferred among the component databases.

Intuitively, this query can be decomposed into the following set of subqueries.

```
Qu1: select * into Q1
     from D1.T1
     where D1.T1.cx = 5;
Qu2: select * into Q2
     from D2.T2
     where D2.T2.cy = 7;
Qu3: select * into Q3
     from D3.T3
     where D3.T3.cz = 6;
Qu4: select * from Q1, Q2, Q3
     where Q1.c1 = Q2.c2
     and Q2.c3 = Q3.c4;
```

The first three subqueries are single database queries. The last subquery, $Qu4$, can be executed as a single database query by collecting the intermediate results Q1, Q2 and Q3 at some site. For example, we can transfer Q2 and Q3 to database D2 and execute $Qu4$ as a single database query at D2 since all Q1, Q2 and Q3 are available. Note that the first three queries can be processed at the three different DBMS in parallel but the last one has to wait for the results from the first three subqueries.

There could also be another processing strategy: (i) broadcast Q2 to the database D1 and D3, and (ii) decompose $Qu4$ into three join subqueries which can be executed at D1 and D2:

```
Qu4.1: select * into Q41
       from D1.Q1, D2.Q2
       where Q1.c1 = Q2.c2;
Qu4.2: select * into Q42
       from D3.Q1, D2.Q2
       where Q2.c3 = Q3.c4;
Qu4.3: select * from Q41, Q42
       where Q41.c2 = Q42.c2
       and Q41.c2 = Q42.c2
       and Q41.c3 = Q42.c3;
```

In fact, more alternatives are available. For example, $Qu1$ can be evaluated first after which the intermediate result Q1 can be sent to D2. Subquery D2 now becomes

```
Qu2': select * into Q12
      from D2.Q1, D2.T1
```

where Q1.c1 = T2.c1
and T2.cy = 7;

The result of $Qu2'$ can now be sent to D3 and the execution of the subquery

$Qu3'$: select * from Q12, T3
where Q12.c3 = T3.c4
and T3.cz = 6;

yields the final result.

The above example illustrates that there could be a number of processing plans for even a simple multidatabase query. Moreover, these plans may have different number of subqueries and hence different number of invocations of component DBMS. The parallelism and the size of data to be transferred among the participating DBMS may also differ. These differences will certainly result in different performances. In the above example, subquery $Qu1$, $Qu2$ and $Qu3$ can be executed concurrently at three different database sites. Subquery $Qu4.1$ and $Qu4.2$ can also be executed concurrently. Concurrently executing a number of subqueries can help reduce the response time. On the other hand, sequential execution of the subqueries may lead to better total processing time. For example, subquery $Qu2'$ combines the selection on T2 (T2.cy > 7) and the join with Q1 into one query. The selection can therefore be performed during the join. The cost of scanning over T2, and hence the total processing cost, is reduced. Minimizing the total processing time may be important in those instances in which a user does not mind waiting a little longer in order that he can be charged less for accessing the various databases.

While the above example may appear simple, it illustrates clearly that we do need a global optimizer to consider (amongst other things) the alternative execution plans and to select an optimal or near-optimal plan for execution. In summary: although the component DBMS are autonomous and most query processing and optimization is being performed at the component database site, the need for global query optimization remains especially if good performances is expected of the multidatabase system.

# 3 Beyond Distributed Query Optimization

While some researchers may concur with the need for global query optimization in a multidatabase system, they have the impression that the query optimization techniques which have been developed for distributed database systems [9] are good enough and can be readily used in MDBMS. This conclusion is however, not quite correct. In fact, the distinctive features of multidatabase systems give rise to a number of unique issues which do not arise in query optimization for homogeneous distributed database systems.

## 3.1 Site Autonomy

*Site autonomy* [1] in multidatabases refers to the situation whereby each component DBMS retains complete control over local data and processing. This has a number of implications for query optimization in a multidatabase system.

First, *communication autonomy* in multidatabase systems means that component sites independently determines what information it will share with the global system, what global request it will service, when it will participate in the multidatabase, and also when it will stop participating. This adds to the complexity of query processing and optimization since any component database system may terminate its services without any advance notice.

Second, *design autonomy* implies that component DBAs are free to optimize local access paths and query processing methods to satisfy local user requirements without any obligation to inform the MDBMS of these changes. Consequently, statistical information which is needed for effective global query optimization are not readily available and may not remain accurate as component systems evolve over time.

Third, *execution autonomy* results in the situation whereby the global system interfaces with the component DBMS at the user level, and hence is not able to influence how query processing is being carred out in the component database. This means that there are now no opportunity for low-level cooperation across systems and hence primitive query processing techniques proposed for distributed databases may no longer be applicable. For example, the semijoin operation has been proposed in order that data transmission between sites can be reduced. When two relations R and S at two different sites, A and B, are joined using the semijoin method, join column values of R is filtered out and sent to site B to fetch matching tuples in relation S. The matching tuples retrieved are sent back to site A and merged with the corresponding R-tuples to complete the join. In DDBMS, this method is quite effective because of the facilities provided by the underlying system environment. In fact, the R-tuples (or just the TIDs of those tuples) retrieved in site A can be staged in memory to wait for the incoming S-tuples and the local processing time (disk access of R-tuple) can be reduced (such as implemented in System R*, the fetch-as-needed join method). In MDBMS, it is hard to implement semijoins efficiently in the same way. The semijoin becomes three subqueries: the first query is to select the join column values of R at site A and send the results to site B; the second query is to join these values with relation S at site B and send the results to site A; and the third query is to join the results with R at site A again to generate the final join results. Although the semi-join may filter out the unnecessary tuple and reduce the data transferred, the local processing time may increase dramatically because relation R has to be scanned twice.

## 3.2 System Heterogeneity

Unlike distributed database systems, component sites in a multidatabase system are not homogeneous. *System heterogeneity* may occur at several levels: component databases may reside on computer systems with different architectures, they may be connected via different types of network supporting different communication protocols, or they may support different data models.

In query optimization for distributed database systems, it is assumed that component sites are equal in terms of their processing capability. This assumption no longer seems reasonable in the context of multidatabase systems since component sites may vary drastically in terms of their availability and processing costs. For instance, a component DBMS may not even be a first class DBMS and thus lack important DBMS features. Some systems also will not support intermediate processing due to the constraints placed on utilization of system resources. Obviously, these constraints need

to be conveyed to the multidatabase query processor since they have a significant impact on the generation of the query processing plan.

### 3.3 Semantic Heterogeneity

In a distributed database system, the same real world object may be represented in more than one component sites but these representations are always structurally compatible. On the other hand, semantic heterogeneity in a multidatabase system results in the same data being represented differently in different component databases. Until recently, this problem has been mostly addressed from a schema integration perspective. We suggest that semantic heterogeneity has an important impact on query processing and optimization.

Consider the *inter-database instance identification problem* which is first highlighted in [8]. In order to match the same entity instance stored in two relations at distinct component sites, some common identifier must exist between them. In many instances, this may not be the case. A brute force solution (which is impractical for large databases) is to store the synonyms of all identifiers in a table and use this for conflict resolution. A more promising approach, described in [8], is to make use of entity properties common to both databases in deciding whether or not two tuples actually refer to the same real world entity. From a query processing perspective, this implies that new query processing methods need to be introduced and it is likely that these will not be supported by the component DBMS. For instance, Chatterjee and Segev [2] have described a new join operator, called *Entity Join (E-join)*, which accomplishes the join of two relations by comparing *useful* attributes are not necessarily structurally identical.

Recently, it has been pointed out that comparing data originating from different databases is only meaningful when their *contexts* are being taken into consideration [7]. For example, it is not meaningful to compare share prices from New York Stock Exchange and the European markets since the currencies used in reporting their prices are different. Hence, data interchange must be preceded by *context mediation* during which data from a different context are converted to an equivalent representation in the current context. Obviously, context mediation can be an expensive operation. The cost incurred as a result of context mediations must be taken into consideration during the generation of the query plan since different orderings of subqueries can result in drastic variations in the cost of context mediations.

## 4 Major Issues and Possible Solutions

Drawing from experiences in distributed and centralized query optimization, we note that the generation of an optimal query plan depends largely on

1. the optimization strategy used by the optimizer, and
2. the accuracy of the statistics used in supporting decision making by the query optimizer, including estimates of subquery execution cost, and the size of the intermediate results.

Contrary to what is commonly assumed, existing algorithms and methods employed for distributed query optimization cannot be readily applied for multidatabase query optimization since autonomy and heterogeneity of component databases have given rise to a number of new issues. For instance, the optimization strategies developed for distributed database systems make extensive use of semijoins which are not as attractive in the multidatabase context. Moreover, these do not take into consideration the cost incurred as a result of context mediations, nor the effect of system heterogeneity in generating the query plan. Similarly, the non-availability of reliable estimates for subquery execution cost and intermediate data size places severe constraints on the effectiveness of the optimization strategies. It is therefore essential for the multidatabase query optimizer to gather vital statistics of component databases while incurring minimal overhead, and to develop query optimization strategies which are more tolerant of fuzzy input information.

### 4.1 The Multidatabase Catalog

In order to generate an execution plan for a given query, the multidatabase query optimizer requires information on

- the locations of the data which are referenced by the query;
- the names of the required data as understood by the component DBMS;
- the database profile for the respective component databases which are needed for estimating the cost of subqueries;
- system-related statistics of components sites, including
  - accessibility of individual component systems (some systems provide unlimited access to foreign users but others may only provide limited access to their resources);
  - workload characteristics measured in terms of CPU, I/O and communication line utilization; and
  - computing power of the host computing system in terms of its processing speed.

To start with, we assume that these information is kept in a *multidatabase catalog* which is accessible to the query optimizer.

It is worthwhile pointing out that the multidatabase catalog is not merely a repository for the global schema. For instance, in order to support new processing methods such as the E-join, additional semantic information (such as those properties of an entity which are deemed to be 'useful' in identifying an entity) has to be kept. Moreover, it is necessary to identify what statistics should be kept, who should update these statistics, and when the update should be performed. One solution to this is a lazy evaluation approach: statistical information of the component databases are collected and updated when they needed. Hence, when a global query is to be processed, the global query optimizer will sent a query to the component database system to get the most updated statistics. The information obtained is used both in the subsequent optimization process and to update the catalog. This approach is justified on the basis that component databases has no obligation to inform the multidatabase of any changes in its system. Updating the multidatabase catalog according to every change in component

database systems is usually expensive and may not be realistic. The cost of maintaining the multidatabase catalog should be charged to the multidatabase users who issue the global queries. Furthermore, requesting the statistics when query is to be processed ensures that the most updated statistics is being used. To reduce the optimization overhead for the global queries incurred by gathering statistics during optimization, two mechanisms can be incorporated. First, the query optimizer may occasionally issue queries to collect and update the statistic for those frequently accessed component databases during off-peak hours. Second, the query optimizer can also use the information currently available in the catalog without issuing sampling queries to reduce the overhead, whenever it is felt that the available statistics are sufficiently current.

## 4.2 The Cost Model

In analogy to distributed query optimization, there can be two objectives in multidatabase query optimization: minimization of *response time* or, *computation cost*. The cost of executing a global multidatabase query comprises many components, arising from

- generation of a query plan;
- invocation of component DBMS;
- processing of subqueries;
- transferring intermediate results among participating component DBMS;
- context mediations; and
- assembly of global query results.

Unlike the case in distributed query optimization, the cost of subqueries cannot be easily determined since the multidatabase query optimizer has no information on the profile of the database, the access paths, or the access methods which are supported by the component DBMS. Similarly, the size of intermediate results are generally unknown and hence communication and context mediation costs are also hard to determine.

We propose the adoption of a *fuzzy* approach in circumventing the above difficulties. Under this approach, subqueries to a component databases are classified according to their complexity into various categories which fall on a linear scale. The execution of queries in each category is sampled againest the component database, and the costs are recorded to form the cost estimates for queries in this category. Note that, with proper sampling technique, this cost estimation takes the system characteristic of the component DBMS into consideration. In a similar manner, the component DBMSs are also classified. The optimizer can make use of cost estimates for similar systems in making estimates for component DBMS whose cost information are not available in the catalog. We also make estimates of intermediate result size and context conversion cost for each such category. These estimates are used as inputs to the query optimizer and will be updated as the component databases evolve.

## 4.3 Subquery Execution Monitoring

To overcome the difficulty arising from the autonomy of component DBMS, the monitoring of subquery execution becomes very important. Although most query optimizers now use compilation approach, query processing and optimization could be interpretive. That is, rather than generating the whole query execution plan before query execution commences, the query optimizer determines the next step in the query execution sequence only after the previous step has been completed. This strategy has the advantage of providing the query optimizer with more accurate information about the data size of the intermediate results (which is one important parameter used in estimating the processing costs). For this reason, this strategy seems more attractive in the multidatabase environment.

We introduce a new function to the multidatabase query optimizer: subquery execution monitoring. Basically, the function requires the query optimizer to collect the completion time for subqueries sent to the component DBMS. A subquery has an estimated completion time that forms the basis for global query optimization. When the completion time of a subquery sent to certain component DBMS far exceeds this estimated completion time, some remedial procedure can be taken. For instance, the unexecuted parts of the plan can be adjusted to avoid sending unexecuted subqueries to this DBMS. Alternatively, the user who issues the query can be notified. This approach moreover allows the discrepancy between the estimated query execution time and the actual execution to be used in updating the data catalog and hence improved the quality of query execution plans generated subsequently.

# 5 Design of a Multidatabase Query Optimizer

The architecture of our multidatabase query optimizer is shown in Figure 1. A global query is first parsed and then decomposed (by the *query decomposer*) into query units which are represented in the form of a *query unit graph*. The *plan generator* constructs subqueries from the query unit graph and estimates their execution costs. The query plan with the minimum estimated cost will be sent to the *dispatcher* who will coordinate the execution of the query. It also collects the statistics about subquery execution and send to the *statistics manager*. The latter will update the multidatabase catalog if necessary.

## 5.1 The Query Decomposer

The function of the query decomposer is to decompose a multidatabase query into query units. A query unit corresponds to primitive operations needed to process a query, such as selection, projection, or join on available data at a single database site. The decomposition can be accomplished according to the following heuristics:

1. Selections and projections on single relations form query units by themselves.
2. Joins and other operations involving only relations stored at the same component database also form query units.

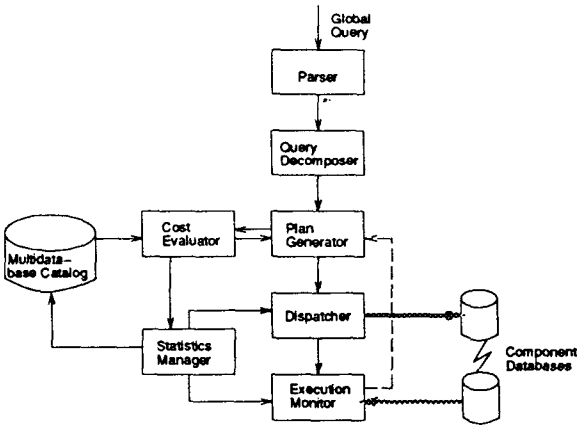


Figure 1: Architecture for a multidatabase query optimizer.

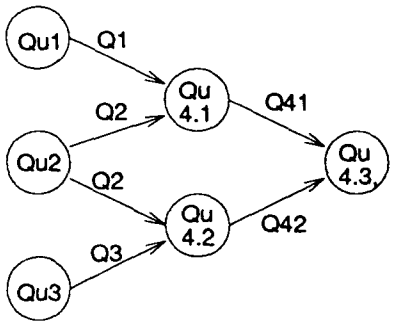


Figure 2: Query graph corresponding to example in Section 2.

3. When a relation is the union of relations at different component databases, query units are formed for each site.

4. For a join (or other operation) involving two different databases, the relations in the join condition are replaced with query units resulting from the query units that retrieve the relation (or part of it) from the original database.

The basic principle here is to decompose a query to the finest level in order to explore all possible execution plans. The example query in Section 2 can thus be decomposed into six query units represented by the query unit graph in Figure 2.

In the above discussion, we have used the relational operators such as projection, selection and join out of convenience. Their meaning however can be easily generalized to transcend the relational data model. For example, any operation that restrictively retrieve data items satisfying some criteria from a database can be viewed as a selection, no matter the data items are tuples in a relation or records in a set linked by pointers.

## 5.2 The Plan Generator

Given a query unit graph, the plan generator constructs the possible plans consisting of the subqueries and their execution sequence. The decomposed query units are grouped together to form subqueries. This grouping process is guided by the cost functions and heuristics. The heuristics are used for reducing the search space. Such heuristics include scheduling more work to the high speed databases with low utilization and high accessibility. A query execution plan specifies the subqueries, the component DBMS involved and the expected response time which is provided by the cost evaluator.

The plan generator interacts with the cost evaluator during the plan generation process. When an execution plan is generated, it is passed to the cost evaluator which provides an estimated cost. A plan which satisfies the optimization objective is finally chosen as the execution plan for the query and sent to the dispatcher who coordinates the execution of the plan among the participating component DBMS

## 5.3 The Cost Evaluator

The cost evaluator works closely with the plan generator. Its function is to estimate the cost of a query execution plan based on the cost model described in the last section and the information stored in the catalog.

## 5.4 The Subquery Execution Monitor

Subquery executions are monitored to ensure that the global query is executed as expected. When a component DBMS completes the execution of a subquery, the monitor collects information about the executed subquery and sent to the statistics manager. This information includes the subquery type, the component DBMS type, the expected response time and the actual response time. The related statistics is then modified accordingly. If the execution of subqueries are much more costly than expected, the execution monitor may communicate with the plan generator and the dispatcher and the unexecuted portions of the plan may be modified. If possible, subqueries to be executed at the slow site could be moved to other sites. The processing plan can also be changed so that the more responsive sites may be assigned more tasks.

# 6 Conclusion

In this extended abstract, we claim that the global query optimization problem in heterogeneous multidatabase systems is fundamentally different from global query optimization in homogeneous distributed database systems. The existing query processing and optimization technologies must be re-examined. Major issues in the design of a multidatabase query optimizer, together with its architectural design, is highlighted. Our design objective is to have a query optimizer which is (i) robust to the heterogeneous environment, and (ii) adaptive to the evolution of component databases. Due to space constraint, we have only sketched the ideas in this extended abstract. The details of the design of the multidatabase query optimizer is presented in the full paper [5]. Performance of the proposed optimization approach is also being studied currently.

## Acknowledgment

The first author would like to thank M.-C Shan of Hewlett-Packard Laboratories who initiated the work reported here and had provided stimulating suggestions.

## References

[1] M. Bright, A. Hurson, and S. Pakzad. A taxonomy and current issues in multidatabase systems. *IEEE Computer*, 25(3):50–60, 1992.

[2] A. Chatterjee and A. Segev. Data manipulation in heterogeneous databases. *SIGMOD RECORD*, 20(4):64–68, 1991.

[3] A. Chen. Outerjoin optimization in multidatabase systems. In *Proceedings of the 2nd International Symposium on Distributed and Parallel Database Systems*, pages 211–218, 1990.

[4] U. Dayal. Query processing in multidatabase system. In W. Kim, D. Reiner, and S. Batory, editors, *Query Processing In Database System*. Springer Verlag, 1985.

[5] H. Lu, B. Ooi, and C. Goh. On global multidatabase query optimization. Unpublished manuscript, May 1992.

[6] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22(3):183–236, 1990.

[7] M. Siegel and S. Madnick. A metadata approach to resolving semantic conflicts. In *Proc. of the 17th International Conference on Very Large Data Bases*, 1991.

[8] R. Wang and S. Madnick. The inter-database instance identification problem in integrating autonomous systems. In *Proc. of the International Conference on Data Engineering*, pages 46–55, 1989.

[9] C. Yu and C. Chang. Distributed query processing. *ACM Comput. Surv.*, 16(4):399–433, 1984.