BY CRAIG S. MULLINS

Distributed Query Optimization

Query optimization is a difficult task in a distributed client/server environment and data location becomes a major factor. Understanding the issues involved enables programmers to develop efficient distributed optimization choices.

Database queries have become increasingly complex in the age of the distributed DBMS (DDBMS). This poses a difficulty for the programmer but also for the DDBMS. Query optimization is a difficult enough task in a non-distributed environment. Anyone who has tried to study and understand a cost-based query optimizer for a relational DBMS (such as DB2 or Sybase SQL Server) can readily attest to this fact. When adding distributed data into the mix, query optimization becomes even more complicated.

In order to optimize queries accurately, sufficient information must be available o determine which data access techniques are most effective (for example, table and column cardinality, organization information, and index availability). In a distributed, client/server environment, data location becomes a major factor. This article will discuss how adding location considerations to the optimization process increases complexity.

COMPONENTS OF DISTRIBUTED QUERY OPTIMIZATION

There are three components of distributed query optimization:

- **Access Method** — In most RDBMS products, tables can be accessed in one of two ways: by completely scanning the entire table or by using an index. The best access method to use will always depend upon the circumstances. For example, if 90 percent of the rows in the table are going to be accessed, you would not want to use an index. Scanning all of the rows would actually reduce I/O and overall cost. Whereas, when scanning 10 percent of the total rows, an index will usually provide more efficient access. Of course, some products provide additional access methods, such as hashing. Table scans and indexed access, however, can be found in all of the "Big Six" RDBMS products (i.e., DB2, Sybase, Oracle, Informix, Ingres, and Microsoft).

- **Join Criteria** — If more than one table is accessed, the manner in which they are to be joined together must be determined. Usually the DBMS will provide several different methods of joining tables. For example, DB2 provides three different join methods: merge scan join, nested loop join, and hybrid join. The optimizer must consider factors such as the order in which to join the tables and the number of qualifying rows for each join when calculating an optimal access path. In a distributed environment, which site to begin with in joining the tables is also a consideration.

- **Transmission Costs** — If data from multiple sites must be joined to satisfy a single query, then the cost of transmitting the results from intermediate steps needs to be factored into the equation. At times, it may be more cost effective simply to ship entire tables across the network to enable processing to occur at a single site, thereby reducing overall transmission costs. This component of query optimization is an issue only in a distributed environment.

SYSTEMATIC VS. PROGRAMMATIC OPTIMIZATION

There are two manners in which query optimization can occur: systematically or programmatically. Systematic optimization occurs when the RDBMS contains optimization algorithms that can be used internally to optimize each query.

Although systematic optimization is desirable, the optimizer is not always robust enough to be able to determine how best to join tables at disparate sites. Indeed, quite often the RDBMS does not even permit a distributed request joining multiple tables in a single SQL statement.

In the absence of systematic optimization, the programmer can optimize each request by coding the actual algorithms for selecting and joining between sites into each application program. This is referred to as programmatic optimization. With systematic optimization the RDBMS does all of the work.

Factors to consider when coding optimization logic into your application programs include:

- the size of the tables;
- the location of the tables;
- the availability of indexes;
- the need for procedural logic to support complex requests that can't be coded using SQL alone;
- the availability of denormalized structures (fragments, replicas, snapshots); and
- consider using common, reusable routines for each distinct request, simplifying maintenance and modification.

AN OPTIMIZATION EXAMPLE

In order to understand distributed query optimization more fully, let's take a look at an example of a query accessing tables in multiple locations. Consider the ramifications of coding a program to simply retrieve a list of all teachers who have taught physics to seniors. Furthermore, assume that the COURSE table and the ENROLLMENT table exist at Site 1; the STUDENT table exists at Site 2.

If either all of the tables existed at a single site, or the DBMS supported distributed multi-site requests, the SQL shown in Figure 1 would satisfy the requirements. However, if the DMBS can not perform (or optimize) distributed multi-site requests, programmatic optimization must be performed. There are at least six different ways to go about optimizing this three-table join.

Option 1: Start with Site 1 and join COURSE and ENROLLMENT, selecting only physics courses. For each qualifying row, move it to Site 2 to be joined with STUDENT to see if any are seniors.

Option 2: Start with Site 1 and join COURSE and ENROLLMENT, selecting only physics courses, and move the entire result set to Site 2 to be joined with STUDENT, checking for senior students only.

Option 3: Start with Site 2 and select only seniors from STUDENT. For each of these examine the join of COURSE and ENROLLMENT at Site 1 for physics classes.

Option 4: Start with Site 2 and select only seniors from STUDENT at Site 2, and move the entire result set to Site 1 to be joined with COURSE and ENROLLMENT, checking for physics classes only.

Option 5: Move the COURSE and ENROLLMENT tables to Site 2 and proceed with a local three-table join.

Option 6: Move the STUDENT to Site 1 and proceed with a local three-table join.

Which of these six options will perform the best? Unfortunately, the only correct answer is "It depends." The optimal choice will depend upon:

- the size of the tables;
- the size of the result sets — that is, the number of qualifying rows and their length in bytes; and
- the efficiency of the network.

Try different combinations at your site to optimize distributed queries. But remember, network traffic is usually the cause of most performance problems in a distributed environment. So devoting most of your energy to options involving the least amount of network traffic is a wise approach. In addition, bad design can also be the cause of many distributed performance problems.

NOT QUITE SO SIMPLE

The previous example is necessarily simplistic in order to demonstrate the inherent complexity of optimizing distributed queries. By adding more sites and/or more tables to the mix, the difficulty of optimization will increase because the number of options available increases.

Additionally, the specific query used is also quite simple. Instead of a simple three table join, the query could be a combination of joins, subqueries, and unions over more than three tables. The same number of options is available for any combination of two tables in the query.

Indeed, there are probably more options than those covered in this article. Consider a scenario similar to the one posed above in which we have three tables being joined over two sites. Tables A and B exist at Site 1 and Table C exists at Site 2. It is quite possible that it would be more efficient to process A at Site 1 and ship the results to Site 2. At site 2, the results would be joined to Table C. Those results would then be shipped back to Site 1 to be joined to Table B. It is not probable that this scenario would produce a more optimal strategy than the six outlined above, but in certain situation, it is possible.

Furthermore, some types of processing require procedural logic (such as looping and conditional if-then processing) to be interspersed with multiple SQL queries to produce a result. In these cases, the procedural logic should be factored into the optimization equation for optimal results. However, the optimizers available in the major RDBMS products don't do a good job of this for non-distributed queries, so the hope of a distributed optimizer performing this type of optimization any time soon is not good.

Finally, there is a laundry list of other considerations that must be taken into account that I have skipped for the sake of brevity. For example:

- The security and authorization implication of who can access what information at which site need to be examined and implemented.

- In a multi-site environment, it is possible (indeed quite likely over time) that one of the sites will not be available for any number of reasons (software upgrade, power outage, hardware/software failure, etc.).

- Declarative referential integrity among multiple sites, in which the data relationships are specified in each table's DDL, are not available

Figure 1: SQL to Satisfy Single Site or Multi-Site Requests

```
SELECT     C.TEACHER
FROM       COURSE                        C,
           ENROLLMENT                    E,
           STUDENT                       S
WHERE      C.COURSE_NO=E.COURSE_NO
AND        E.STUDENT_NO=S.STUDENT_NO
AND        S.STUDENT_LEVEL="SENIOR"
AND        C.COURSE_TYPE="PHYSICS"
```

in any DDBMS to date. The specification of these relationships would greatly assist application development efforts, as well as distributed query optimization.

■ Distributed structures can be implemented to augment performance. A multi-site, multi-table index structure could be created that would contain information on the physical location of tables, as well as the physical location of the data items within that table. This structure, however helpful from a performance perspective, would be difficult to maintain and administer due to its reliance on multiple sites.

■ The optimization process will be highly dependent upon the implementation and usage of the network. The amount of network traffic can vary from day-to-day, and even hour-to-hour, thereby impacting the optimization choice. Whenever the network is modified in any way (tuned, new release, additional nodes added, etc.), the optimization choice should be re-addressed as a new, more optimal path may now be available. This can quickly become a drain on the resources of the system (and the personnel administering the system).

SYNOPSIS

Introducing data distribution into the query optimization process makes a complex issue that much more complex. Until the distributed DBMS products support the systematic optimization of distributed multi-table SQL requests, programmatic optimization will be a fact of distributed life. Understanding the issues involved will enable application programmers to develop efficient distributed optimization choices. ts

Until the distributed DBMS products support the systematic optimization of distributed multi-table SQL requests, programmatic optimization will be a fact of distributed life.

Craig S. Mullins is a senior technical advisor and team leader of the Technical Communications group at PLATINUM **technology, inc.** *Craig's book,* **DB2 Developers Guide***, contains more than 1,200 pages of tips and guidelines for DB2 and can be ordered directly from the publisher, SAMS Publishing, at 1-800-428-5331. Craig can be reached via the Internet (mullins@platinum.com), CompuServe (70410,237), America Online (CraMullins), or at PLATINUM* **technology, inc.** *(800-442-6861, fax: 708-691-0709).*