Distributed Query Optimization

BY CRAIG S. MULLINS

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 to determine which data access techniques are most effective (i.e., table and column cardinality, organization information and index availability). In a distributed, client/server environment, data location and network characteristics become major factors. 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 Relational DataBase Management System (RDBMS) products, tables can be accessed in one of two ways: by completely scanning the entire table or by using one or more indexes. 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 normally want to use an index. Scanning all of the rows would actually reduce I/O (input/output) and overall cost. Whereas, when scanning 10 percent of the total rows an index will usually provide more efficient access.
- 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 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-efficient simply to ship entire tables across the network to enable processing to occur at a single site, thereby reducing overall 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 access 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 accessing between sites into each application program. This is referred to as programmatic optimization. With systematic optimization, the RDBMS does all of the work, but with programmatic optimization, the programmer does most of the work.

Some factors to consider when coding optimization logic into your application programs:

- The size of the tables;
- The location of the tables;
- The availability of indexes;
- The availability of denormalized structures (fragments, replicas, snapshots); and
- The use of common, reusable routines for most distinct requests, 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 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 all of the tables existed at a single site, or the DBMS supported distributed multi-site request, the following SQL statement would satisfy the requirements:

```
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"
```

But, if the DBMS cannot 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 they 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. Sort by student no. before or after shipping based on relative horsepower at sites 1 and 2.

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 that 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, poor design can also be the cause of many distributed performance problems.

Conclusion

Introducing data distribution into the query optimization process makes a complex issue even 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.

Craig S. Mullins is the senior analyst relations specialist for PLATINUM *technology, inc.* His duties include interfacing with industry analyst groups, authoring white papers, and speaking at user groups and industry conferences. He is the author of the book, *DB2 Developer's Guide*, which contains in-depth technical information on DB2 Version 3.

Was this article of value to you? If so, please let us know by circling Reader Service No. 33.