Distributing: How To Take Advantage of the SQL Environment

By Dr. George Schussel

As president of Digital Consulting Inc., Dr. Schussel has consulting and educational responsibilities in private industry and government. He is chairman of the DATABASE WORLD conference May 28 - 30, 1991 in Washington D.C.

Distributed DBMS- and SQL-based client/server computing are both increasing as companies downsize and spread their data base processing over several smaller machines. This is causing the cost of OLTP to decrease, even as the total number of machine cycles increases; the reason is that the combination of small machine cycles and OS/2 or UNIX platforms allows OLTP processing on environments that cost one to five percent of a mainframe configuration.

The best way to take advantage of the better performance (per dollar) of smaller machines is to use distributed DBMS software. The market is made up of products in two categories: true distributed DBMS and client/servers. As vendors improve the software capabilities of their client/server systems, it is likely that the functional differences between these two will tend to disappear, but probably not before the mid-1990s.

Most SQL DBMS vendors have jumped into the client/server game and the functionality delivered by these systems is not that different from true distributed DBMS. The key difference is that a client/server approach has a DBMS and dictionary at only certain designated nodes where the data resides; the client program is required to navigate to the correct server node by physically knowing which particular server to access for the necessary data. The client/server DBMS then does not support location transparency. On the contrary, a distributed DBMS that has a local DBMS and local data dictionary at each point in the network relieves the application program from having to perform navigation. This location transparency requirement, however, makes building a true distributed DBMS much more complex than building a client/server DBMS.

Interestingly, IBM does the reverse. They talk about "client/server" but really mean true distributed computing. IBM is building a fully functional distributed architecture for its SQL products: DB2, SQL/DS, SQL/400, OS/2EE and, consistent with the complexity of true distributed DBMS, is taking several years to develop it.

Distributed data base software has to provide all of the functionality of multiuser, mainframe data base software but allow the data in the data base itself to exist on a number of different but physically connected computers. The kinds of functionality the distributed DBMS must supply include maintenance of data integrity by automatically locking records and rolling back transactions that are only partially complete. The DBMS must attack deadlocks, automatically recovering completed transactions in the event of system failure. There should be a capability to optimize data access for a wide variety of different application demands. Distributed DBMS should have specialized I/O handling and space management techniques to ensure fast and stable transaction throughput. Naturally, these products must also have full data base security and administration utilities.

Requirements for Distributed DBMS

Following are seven comprehensive requirements that represent the current thinking on what the DBMS must perform to be qualified as fully distributed. While it is true that there are no products available today that meet all seven requirements, most major vendors such as ASK/Ingres, Oracle and IBM are aggressively adding functionality to their products to meet them.

Location Transparency — Programs and queries may access a single logical view of the data base; this logical view may be physically distributed over a number of different sites and nodes. Queries can access distributed

objects for both reading and writing without knowing the location of those objects. A change in the physical location of objects without change in the logical view requires no change of application programs. There is support for a distributed JOIN. To meet this requirement, it is necessary for full, local DBMS and the data dictionary to reside on each node.

Performance Transparency — It is essential to have a cost-based software optimizer to create the navigation for the satisfaction of queries. This software optimizer should determine the best path to the data. Performance of the software optimizer should not depend upon the original source of the query. In other words, because the query originates from point A, it should not cost more to run than the same query originating from point B.

An interesting evaluation and rating procedure for software optimizers has been developed by David McGoveran of Database Associates. McGoveran's rating scheme (described in the Fall 1991 *InfoDB*) starts with the lowest level of functionality (class 1) where the optimizer is syntax sensitive and uses no intelligence in the search strategy (e.g., Oracle's Version 6). The schema progresses to the most sophisticated class of optimizers (class 6), which is syntax insensitive and uses cost data and heuristic algorithms to determine an optimal search strategy. This class also takes advantage of statistics about the data base to perform a global optimization in a fully distributed environment. (There are no Class 6 optimizers at this time.)

Copy Transparency — The DBMS should optionally support the capability of having multiple physical copies of the same logical data. Advantages of this include superior performance from having local rather than remote access to data, and non-stop operation in the event of one site going down. If a site is down, the software must be smart enough to re-route a query to another source where data exists. The system should support "fail over reconstruction." This means that when the down site becomes live again, the software automatically reconstructs the data at that site to make it current.

Transaction Transparency — The system supports transactions that update data at multiple sites. Those transactions behave exactly the same as others that are local. This means that transactions will commit or abort. In order to have distributed commit capabilities, a technical protocol known as a two-phase commit is required.

Fragmentation Transparency — The distributed DBMS allows a user to cut relations into pieces (horizontally or vertically) and place those pieces at multiple physical sites. The software has a capability to recombine those tables into units as necessary to answer queries.

Schema Change Transparency — Changes to data base object design need only be made once in the distributed data dictionary. The dictionary and DBMS automatically populate other physical catalogs.

Local DBMS Transparency — The distributed DBMS services are provided regardless of the brand of the local DBMS. This means that support for remote data access and gateways into heterogeneous DBMS products is necessary.

Advantages of Distributed Data Base Technology

Distributed DBMS technology provides the highest level of services for supporting distributed processing. Specific advantages from the use of this technology include:

- As your processing needs grow, you can upgrade the hardware environment incrementally and as needed without throwing away your previous investments. By spreading the processing over many smaller machines, you take advantage of the downsized cost advantage that smaller machines hold over larger ones.
- The fact that a distributed DBMS offers support for replicated data can contribute mightily to satisfying requirements for high availability and fault tolerance. This same architecture is helpful for hardware maintenance because of its modularity.
- Distributed DBMS technology offers high performance SQL-based processing because its architecture takes advantage of parallel processing on many computers across a network. As a result, you can use relational processing for online transactions that might otherwise have been impractical on a single mainframe.

Note that a client/server SQL type of DBMS (like Sybase or Gupta), while not meeting all seven of the requirements above, does offer the preceding advantages.

Potential Problems with DDB

In pursuing these advantages, however, the prospective user is wise to understand the potential pitfalls of this technology:

- Communication costs can be quite high; using a two-phase commit protocol generates a lot of communications traffic.
- There is need for gateway technology to handle SQL differences among the different DBMS vendors.
- The predictability of total costs for distributed queries is variable. In other words, it is hard to predict ahead of time how much it's going to cost you to get a job done.
- Supporting concurrency along with deadlock protection is a very difficult technology.
- Supporting full recovery with fail over reconstruction is very expensive.
- Performing a JOIN across different physical nodes is very expensive using today's technology and networks.
- Some advanced relational functions that are reasonable in a single computer are difficult and expensive across a distributed network (e.g., the enforcing of semantic integrity restraints).
- The job of the data base administrator is more difficult because, added to all currently existing requirements, are the integrity, optimizer, communication and data owner issues of the distributed world.
- Data security issues are not well understood or proven. It would appear that a distributed environment is more susceptible to security breaks than a data base contained in one box.

Conclusion

Downsizing and distributing the computing environment are very real and practical approaches to getting more functionality at a lower cost. The availability of SQL has greatly accelerated the movement toward distributed and client/server DBMS environments. This has been principally because SQL has emerged as the standard data base language. The fact that SQL is a relational language and, therefore, supports set processing is also very helpful in a distributed environment. Distributed and client/server SQL DBMSes form the keystone in the migration to the distributed network-based parallel computing architectures of the 1990s.

■ *Was this article of value to you? If so, please let us know by circling Reader Service No. 79.*