DISTRIBUTED AND CLIENT/SERVER DBMS: UNDERPINNING FOR DOWNSIZING

I. INTRODUCTION TO THE CHAPTER

One of the key trends of modern computing is the downsizing and distributing of applications. This is happening because companies want to take advantage of modern microprocessor technology and at the same time gain the benefits of new styles of software using graphical interfaces (GUI). Client/server and distributed database technologies are fundamental enabling technologies for downsizing.

Client/server approaches allow the distributing of applications over multiple computers, with the databases residing on server machines, while applications run on client computers, usually PC's. A local area network (LAN) provides the connection and transport protocol to connect the clients and servers.

Distributed databases offer capabilities similar to client/server databases, except more so. More so in the sense that a database management system (DBMS) resides on each node of the network and allows transparent access to data anywhere on the network, without requiring the user to physically navigate to the data. Many of the advanced functions described later in this chapter such as stored procedures and triggers and two-phase commit are available in both client/server and distributed DBMS.

In a client/server approach the application must be aware of the physical location of data, as least as respects to what server it's located on. Once an SQL inquiry or remote procedure call (RPC) is directed to the right server, its query optimizer for SQL will handle the internal navigation of the database.

Client/server DBMS and distributed DBMS have much in common as will be brought out later in this chapter. Either architecture is appropriate for distributing applications. Both are based on the SQL language invented in the 1970's by IBM and standardized by ANSI and ISO as the common data access language for relational databases.

II. INTRODUCTION TO DISTRIBUTED DATABASE COMPUTING

The market for modern distributed DBMS software started in 1987 with the announcement of INGRES-STAR, a distributed relational system from RTI (now the INGRES Division of ASK computers) of Alameda, California. Most of the original research on distributed database technology for relational systems took place at IBM Corporation in IBM's two principal California software laboratories, Almaden and Santa Theresa. The first widely discussed distributed relational experiment was a project called R-Star, developed within IBM's laboratories. It is because of IBM's early use of the word STAR in describing this technology that most vendor's distributed database systems names have incorporated "STAR" in one form or in another.

Today, the market for distributed DBMS is almost entirely based on the SQL language and extensions. (The principal exception is Computer Associates which inherited prior to relational systems with IDMS and DATACOM and has implemented distributed versions, both with and without SQL)

True distributed DBMS products can be considered to occupy the Mercedes Benz segment of the market place. These products support a local DBMS at every node in the network along with local data dictionary capability. The requirement for a full DBMS piece on each node is the essential differentiating factor from client/server systems. In the client/server approach the DBMS resides on one (or a few) nodes and is accessed from a requester piece of software that resides on the client machines.

The market for true distributed DBMS has grown slowly for two reasons: 1) users aren't sure of how to use the products and 2) the vendors are taking the better part of a decade to deliver full functionality. One important unanswered concern of companies who want use true fully distributed DBMS environments is the cost of the communications for functions that have historically been run internal to single computers.

A related technology to distributed DBMS is distributed access (DA). DA is about the building of gateways that allow one DBMS to access data stored in another.

Distributed access can properly be thought of as a subset of technologies that are being delivered by those vendors selling true distributed DBMS or client server DBMS technologies. The demand for distributed access, of course, is greatest for the most popular mainframe file and database environments such as IBM's IMS, DB2, VSAM, and DEC's Rdb. Local DBMS capability is not a requirement for distributed access. Most vendors provide a piece of software known as a requester to be run in the client side of the RDA environment. Some of the products in this market are not finished gateways but tool kits for users to build their own custom gateways. It is beyond the scope of this chapter to pursue the issue of DA further.

If true distributed DBMS products are the top of the market then client server DBMS engines are the Fords and Chevrolets. By accepting some reduction in functionality from the definition of the fully distributed DBMS, vendors have been able to develop client/server DBMS that can run exceedingly well on today's level of modern PC's and networks.

It is this author's opinion that the market place for client server approaches is going to be far larger in dollar volume than either true distributed DBMS. Another advantage of the client/server approach (in some eyes) is that the model of one (or a few) database locations appears more manageable than a data location policy which spreads data out over many nodes, where it might be more difficult to manage properly.

III. INTRODUCTION TO CLIENT/SERVER DATABASE COMPUTING

In downsizing, many companies are thinking about implementing applications that before now would have had to reside on mainframes. Before a commitment is made to migrate such an application, assurances about the integrity of the data and application must be made. In addition, PC's and LANs have had a reputation of not offering security that is equivalent to the mainframe. Client server computing is a solution that combines the friendly interfaces of the PC with the integrity, security and robustness of the mainframe. Server databases on PC LANs are implementations of the SQL database access language, that is the same as the standard database language used on mainframes. Once you decide to rethink your applications into running on workstations using shared databases located on servers and connected by networks, you've made the essential decision to building an applications architecture that will be economical, flexible and portable for a long time into the future.

The functionality delivered by today's client server systems is not that different from true distributed DBMS. The key difference is that a client server approach only has a DBMS and Dictionary at 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 idea for client server computing grew out of database machine vendor approaches. Sybase's Robert Epstein had worked for Britton Lee when he came up with the idea of creating a database machine environment, but with a server that was a virtual machine rather than a physically unique piece of hardware. The systems software, then, was separated into a front end (client) which ran the program (which would be written in a 4GL) and a back end (server) which handled the DBMS chores. The advantage of this idea was that the back-end virtual database machine could physically be moved out onto a different piece of hardware whenever desired. What made this approach different from the Britton Lee approach was that Sybase planned for the server to be a generic VAX, UNIX, or PC machine rather than unique custom build database hardware. By moving the database machine into a standard piece of hardware, Sybase picked up the advantage of vastly improving price performance in generic small systems.

At about the same time that Epstein was starting Sybase, Umang Gupta (then a Senior Oracle executive) had come up with the same idea and left Oracle to form Gupta Technologies, a company which has emerged as a leader in client/server DBMS and tools products, especially for the PC environment. Bing Yao, the former University of Maryland professor who founded XDB Systems, was another early implementer of client/server approaches to database computing.

Most other SQL DBMS vendors have jumped into the client server game. An exception is IBM, which while talking about "client server" really means true distributed computing. IBM is building a fully functional distributed architecture for its SQL products, DB2, SQL/DS, SQL/400, OS/2EE. IBM is taking several years to develop this approach.

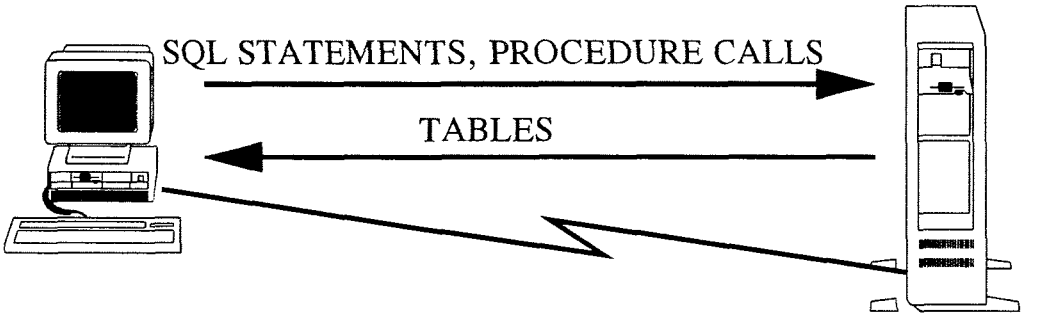
A client/server computing environment consists of three principal components: Client, server, network (see chart).

The client is where the application program runs. Normally, the client hardware is a desktop computer such as an IBM PC, clone or Apple Mac. The application program itself may be written in a 4GL or in 3rd generation languages such as C or COBOL. There are a whole new group of "Windows 4GLs" that allow painting of applications under the leading desktop windows based operating systems.

These products support both windows oriented application development and execution. Leading examples include Powersoft's Powerbuilder, INGRES' Windows 4GL and Gupta's SQL Windows. Using any of these application building approaches results in a runtime configuration in which the I/O and application control come from the client, while the database and associated semantics run on the server. At the desktop level, most software will support the emerging windows based standards. This means Windows 3 for DOS, Presentation Manager and Windows 3 for OS/2, Open Look and Motif for UNIX.

The network is responsible for connecting client and server. Normally, the network consists of some kind of wire along with the communications card in both the client and server boxes. The communications software typically handles different types of communication standards such as LU6.2 and TCP/IP. Most network environments provide support for multiple clients and servers.

CLIENT-SERVER FUNCTIONS



CLIENT

APPLICATION PROGRAM
SCREEN FORMS
GENERATION OF SQL
APPLICATION CONTROL
TASK SWITCHING

NETWORK

HARDWARE/WIRE
COMMUNICATIONS SOFTWARE
MULTIPLE C & S

SERVER

OPTIMIZE & EXECUTE SQL
MANAGE TRANSACTIONS
BUSINESS RULE ENFORCEMENT
STORED PROCEDURES & TRIGGERS
SECURITY
CONCURRENCY MANAGEMENT
LOGGING & RECOVERY
DATABASE CREATION & DEFINITION
DATA DICTIONARY

The server is responsible for executing the SQL statement received from the client. Sometimes the data request is not pure SQL but it can be a remote procedure call which would then trigger a series of already compiled existing SQL statements on the server.

The server is responsible for SQL optimization, determining the best path to the data, and managing transactions. Some server technologies support advanced software capabilities such as stored procedures, event notifiers and triggers. The server is also responsible for data security and validation of the requester.

A server also handles additional database functions such as Concurrency Management; Deadlock Protection and Resolution; Logging and Recovering; Database Creation and Definition.

The idea of managing data on a separate machine fits well with the management approach of treating data as a corporate resource. In addition to executing the SQL statement, the server handles security and provides for concurrent access to the data by many queries.

An important benefit that the set oriented SQL language provides is network efficiency. When using traditional file serving PC LAN approaches, the entire data file must be transmitted across a network to the client machine. Using SQL as a basis for database management solves this problem since only the necessary query response data (a table) is transmitted to the client machine.

SQL on the server also allows the implementation of advanced facilities such as triggers and automatic procedures in the database. As relational DBMS' evolve, they will confer the ability to build rules directly into the database engine. The systems that are built with this approach will be more robust than traditional application based logic approaches.

Although client-server computing is being planned for environments which use minicomputers and mainframes as the server, the largest market is likely to develop with a mix of OS/2, Windows 3, the upcoming Windows NT and MS-DOS on the client and either UNIX, Windows NT, Netware or OS/2 as the base for the server. The server software will provide mainframe level security, recovery and data integrity

capability. Functions such as automatic locking and commit rollback logic along with deadlock detection and a full suite of data administration utilities are available on the server side. Another way of looking at this then is that SQL client-server technology allows cheap PCs to made into "industrial strength" computing engines.

IV. MORE DETAIL ON DISTRIBUTED DBMS

Distributed DBMS is one of the most interesting areas of the large systems DBMS market today. (Large systems here are defined as 80486 on the small end to Cray at the top.) With the emergence of SQL as a standard, the principal ways that DBMS vendors are differentiating their products is by adding function in:

- distributed or client server computing
- support for Object approaches
- addition of database semantics
- adding more relational functionality (typically semantics)

No DBMS vendor can afford to ignore distributed capabilities.

Distributed database software has to provide all of the functionality of multi-user mainframe database software but allow the data in the database 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 wide variety of different application demands. Distributed DBMS should have specialized I/O handling and space management techniques to insure fast and stable transaction throughput. Naturally, these products must also have full database security and administration utilities.

The discussion below first focuses on the basic and then some advanced functions for a distributed DBMS. It isn't useful to view this discussion as a feature checklist, since there is a great disparity between performing these functions at a minimum level and accomplishing them at an advanced level.

Basic requirements for a distributed DBMS

1. Location transparency

Programs and queries may access a single logical view of the database; 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. In order to meet this requirement it is necessary for full local DBMS and data dictionary to reside on each node.

2. Performance transparency

It is essential to have a 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. Technology in this field of software optimization is rather primitive at this time and will be discussed further below.

3. 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 that the software automatically reconstructs the data at that site to make it current.

4. 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 2-phase commit is required.

5. 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.

6. Schema change transparency

Changes to database object design need only to made once into the distributed data dictionary. The dictionary and DBMS automatically populate other physical catalogs.

7. Local DBMS transparency

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

Four ways to distribute data

Most vendors are taking many years to develop software that offers full distributed DBMS capability. As a way of bringing its distributed SQL products to market, IBM has proposed a phased implementation of four discrete steps to distribution of data. These four principal steps are defined below.

Extracts - the ability to extract data simply means that there is a batch process which unloads and reformats operational data into a relational view. For example, IBM's DXT allows for batch unloading of IMS and reformatting into DB2. This extraction is manually managed.

Snapshots - are becoming a popular technique among many vendors. A snapshot is an extract as defined above along with a date and time stamp. The advantage of a snapshot is that after it's defined to the system, it is automatically created and managed. Snapshots are read-only and are effective for providing decision support access to true production data where operations personnel do not want live access to the production data (normally for performance reasons).

Distributed tables - Distributed tables can be thought of as the first level of true, real time, read/write distributed DBMS functionality that meets requirement 5) mentioned above. A system which can support distributed tables will normally manage a single physical copy of data to support the system's logical views.

Replicates - Replicates are a more sophisticated version of distributed DBMS capabilities mentioned under copy transparency above. This can be thought of as support for a single logical view by up to "n" physical copies (of the same data). These data replicates must be updatable (not snapshots). At a minimum, updatability of

physical data replicates will require a software optimizer (as discussed below) and a 2-phase update commit protocol.

Software Optimizers

When there are different physical sites involved, the difference in cost between the best and worst ways of accomplishing a function such as a JOIN can easily be millions to one. Because of this, a distributed DBMS absolutely must have a cost-based software optimizer. Without a cost based optimizer, navigation to data must be under programmer control, violating a basic precept of relational theory (this is what must be done with Oracle version 6.0). Without a cost based optimizer only known queries can be handled, since the performance of an unanticipated query may be impossible.

A reasonable software optimizer has to be intelligent enough to ask tough questions and to develop a correct search strategy based upon the answers to those questions. Examples of types of issues that should be handled are:

1. How busy are the various machines on the network?
2. What are relative speed of these machines?
3. What are the table sizes that have to be accessed?
4. What is the line speed between various nodes of the network?
5. How busy are those lines?
6. How are the tables organized?
7. What are the access patterns in indexes?
8. Where should software optimizer itself run?

   etc.

Two-Phase commit protocol

The goal of the 2-phase commit protocol is to allow multiple nodes to be updated in synchronized fashion as result of a single group of SQL statements which must be committed or rejected together.

The general procedure is as follows:

1. One node is designated as a master; the master sends notice of an upcoming query out to all of the slaves.
2. The slaves respond with ready messages when all of the data necessary for the protocol is available.
3. The master sends out a "prepare" message to the slaves.
4. The slaves lock the necessary data and respond with "prepared" message to the master.
5. The master sends a "commit" message to the slaves.
6. The slaves respond with a "done" message.

For the DBMS software vendor, developing a 2-phase protocol is one of the most challenging tasks in developing a distributed DBMS. Additional complexity in creating this software comes about from the fact that there are different types of failure nodes and the software has to handle and recover from any combination of failure over all environments supported. For the user, operation in an environment requiring a 2-phase commit may be very costly. The cost comes about from the fact that use of a 2-phase commit requires an extra round-trip message over what happens in single computer system.

There are no standards for implementation of a 2-phase commit. Different vendors have come up with different partial implementations. It is likely that we will see a future ISO standard dealing with 2-phase commit protocol.

More advanced capabilities for distributed (or client/server) DBMS

♣ ***Gateways*** Many of the distributed database and client/server DBMS products have optional gateways that allow access to data stored in other DBMS. Lower levels of functionality are for read only access, while higher levels of function can allow write access also. This higher level should be accompanied by a 2-phase commit capability across the different systems (general availability of this capability is still in the future.)

♣ ***Disk mirroring*** is a very useful facility for applications that require extremely high up time. Mirroring allows an organization to keep two exact copies of a database (usually on two separate disks). If one disk fails, then the system will automatically use the other disk without interrupting operations. Mirroring is crucial to many OLTP applications that require fault tolerant operation.

♣***Relational Integrity*** An important server function that support increased productivity in application development is relational integrity. This can include features such as referential integrity or the ability to state business rules directly into the database with stored procedures or program triggers.

♣***Triggers*** are small SQL programs, written in SQL extended language, that are stored in the DBMS catalog. Each trigger is associated with a particular table and a SQL update function (e.g., update, delete, and insert). They are automatically executed whenever a transaction updates the table. You can write triggers to enforce any database validation rule, including referential integrity.

Since they are stored in the catalog and automatically executed, triggers promote consistent integrity constraints across all transactions. The triggers are easy to maintain because they are stored in only one place - the DBMS catalog. Rules are enforced for any application that accesses the database, such as spreadsheet programs.

♣***Multi-Threaded Architecture*** For best performance the distributed (or client/server) DBMS should implement a multi-threaded, single server architecture. Multi-threaded servers perform most of their work and scheduling without interacting with the operating system. Instead of creating user processes, multi-threaded servers create a thread for each new user. Threads are more efficient than processes, and they use less memory and CPU resources.

A multi-threaded server DBMS can service 10 to 40 users simultaneously on a machine as small as a 33MHz 80386 PC with 10MB of RAM.

♣***Symmetric Multiprocessing*** Another advantage for this type of DBMS server is direct support of multiprocessor hardware architectures in a symmetric multiprocessing (SMP) mode. Most operating systems are (UNIX, Windows NT, VMS) or will (OS/2) offer support for this functionality and so there needs to be effective integration between the DBMS and operating system to take advantage of the potentially improved throughput of SMP.

Direct support for SMP (Symmetric multiprocessing) means the DBMS can take advantage of several parallel processors under the same skin (with an appropriate operating system). These processors can be tightly or loosely coupled. As of mid '92, then for example, Borland's Interbase can take advantage of VAX Clusters, which neither Sybase or Oracle can use to full advantage.

♣***Cursors*** A cursor stores the results of a SQL query and allows a program to move forward through the data one record at a time. Sometimes a programmer can move backward within a cursor. Without a cursor, it's harder to program transactions that must browse through data.

♣***Text, Image, Date and other extended data types*** Support for different types of data can make any DBMS useful in a wider variety of applications. To store a picture, it would be useful to have something like IMAGE data types which are binary data Another useful item is TEXT data types which are printable character strings.

♣***Remote procedure calls (RPC)*** allow an application on one server (or client) to execute a stored procedure on another server. Stored procedures enhance performance, since all of the commands can be executed with one call from the application program.

♣***Multiplatform implementations*** Another primary advantage is multi-platform portability and networking. If your software runs on many different vendor's hardware, then you have that much more flexibility in picking. Oracle was built with a marketing approach that outdid all other DBMS products as far as hardware variety supported.

♣***Disk Mirroring*** For companies wanting the reliability of mainframe environments on the PC LAN, a disk or even, server, mirroring capability is necessary. Mirroring implies that dual operations are executed for each computing step, with error reports whenever there is any difference between the results of the dual steps. Mirroring also allows the system to continue to operate at essentially full speed even after one of the processors or disks has failed (non-stop operation, although with more chance of system failure or data corruption). Disk mirroring is supported through the process called "shadowing".

♣***Application Specific Functions*** This capability allows a user to easily extend the range of database commands by adding new functions coded in C to the DBMS kernel. This facility is helpful in the manipulation of BLOB data.

♣***Event Alerters*** An event alerter is a signal sent by the database to waiting programs to indicate that a database change has been committed. Event alerters work remotely, even across multi-vendor networks. Although it would seem to be a simple functional addition to add event alerters to a system that supported the concept of triggers implementation of the technology is made difficult by the need to support an asynchronous, heterogeneous environment.

Event alerters offer the following benefits:

- No network traffic or CPU cycles are consumed by the waiting program.
- Notification is effectively instantaneous, not dependent on some polling interval.
- Event notification works remotely, even across differing platforms. The notification mechanism is managed by the DBMS.
- Unlike a trigger, an event alerter can affect programs outside the database.

♣***BLOB data types*** A BLOB data type (binary large object bin) has no size limit and can include unstructured non-relational types of data such as text, images, graphics, and digitized voice. One way to handle BLOBs is as a single field in a record, like a name, date, or floating point number. It can then be governed by concurrency and transaction control.

The ability to create "database macros" which can be executed by the database engine should be supported within the DBMS. These macros would be implemented as centrally stored, user written procedures that tell the database system how to translate BLOB data to another format. Because they are stored in one place and managed by the database, BLOB macros are simpler to create and maintain than similar code in an application.

♣***Multidimensional Arrays*** Especially in scientific processing or time series commercial types of applications, array support for in the database. is useful. Arrays are stored as a single field in a record, so retrieval is expedited. Arrays are widely used in scientific processing and are very expensive to normalize for a relational DBMS. Normalization of an array typically means creating much added redundant data

to generate separate records for information that is really only different at the field level.

V. MORE ABOUT ON CLIENT/SERVERS

Almost all of the advanced functions listed above for distributed DBMS (such as BLOB data types, RPC's and event alerters) are also available from leading vendors of server DBMS. Repeating what was stated earlier, the primary difference between Server DBMS and distributed DBMS is whether or not each node on the network runs a full copy of the DBMS. Added functions aren't a good way to tell the difference between these two cousin technologies.

Many companies are delivering client/server DBMS and associated tools at this time. The very large and active market of the 1970s and 1980s for mainframe DBMS and 4GLs that featured companies like Cullinet, IBM, Software AG, Cincom and Applied Data Research, has been replaced by a new market. This new environment is built around the client/server model with open availability (connectivity) between tools and DBMS and is competed for by companies like Microsoft, Revelation Technologies. Borland, Sybase, Oracle and Powersoft. The reasons for the current and impending growth of this market are many:

- The architecture is simpler (from a software developer's point of view) than is distributed DBMS and therefore more important (matter of opinion here) capabilities at a lower price point can be brought to market sooner.
- Developers can use PCs instead of timeshare terminals as a primary development platform.
- Even though the PC is used as a principle platform, security, integrity and recovery capability comparable to mini-computers is the result.
- The efficiency of query and transmission of the SQL language greatly reduces the network communication load (for example, from that of a PC LAN/file server based approach).
- Gateway technologies, which are an important component of client-server computing software, will allow PC users to gain access to data located in mainframe and mini computer DBMS products such as DB2, IMS and Rdb.
- The client server model isolates the data from the applications program in the design stage. This allows a greater amount of flexibility in managing and expanding the database and also in adding new programs at the application level.

- The client server model is very scalable because as requirements for more processing come up, more servers can be added in the network or servers can be traded up to the latest generation of micro processor.
- A lot of flexibility comes from a computing environment based upon SQL, because SQL is adopted by almost everyone as a standard. Commitment to an SQL server engine will mean that most front end 4GL, spreadsheet, word-processing, and graphics tools will be interfaced to the SQL engine.
- Client/server computing provides the industrial strength security, integrity and database capabilities of minicomputer or mainframe architectures while allowing companies to build and run their applications on PC and minicomputer networks. The use of this hardware and software combination can cut 90% of the costs of the hardware/software environment for building "industrial strength" applications.

The client/server model offers the user the choice of many different hardware and operating choices as platforms. The hardware choices are too many to be listed here, but the principal choices for operating systems are multi-user, multitasking, protected products such as UNIX, OS/2, Windows NT and Netware. The microprocessor engine driving the hardware is typically a single or dual processor Intel x86 or RISC chips such as SPARC or the MIPS R4000.

The client environment is typically a smaller but still powerful PC, which has the power of running applications built on top of multitasking, single user operating systems such as Windows 3 or OS/2. Although the concept of using a large mainframe such as a VAX 9000 or ES/9000 as a database server to networks is discussed by the mainframe vendors, it is just "slide ware" so far. For these larger machines to play a role in future networks, however, it is clear that they will have to acquire server functionality by acquiring and supporting emerging downsizing standards such as UNIX, Netware, NT's LAN Manager and OS/2's LAN Server.

Performance from a client/server environment

The reader might be skeptical that PC's running server software can perform as well as mainframes, but there is documented evidence to this effect. The most efficient PC server operating system at this time is probably Netware. Tests run according to the Transaction Processing Council's standards have shown products like ORACLE and

Gupta's SQLbase capable of running about 50 TPS on 486 based PC's. This number would not be an unreasonable result for a mainframe running IBM's DB2.

The transaction capabilities of C/S software working with lower end PC servers or Super Servers (minicomputer style cabinets built with merchant microprocessors such as the 80486 or R4000) is quite astounding. For example, on the low end of the hardware scale, both Gupta's SQLBase and Microsoft's SQL Server can run on Intel Corp. 80486-based PCs processing approximately 18 TPC-B (Transaction Processing Council, database Benchmark B) transactions/second. PC hardware can support disks with 12-msec access time and 4M to 6M byte transfer rates. Such a machine can be configured with 600MB of disk for under $10,000. In case you're not familiar with the TPC-B benchmark, it should be pointed out that a rate of 18 transactions/second is adequate to support 400 automated teller machines on a single server.

If you have had a chance to build PC based database applications in the last few years you may be suspicious of any claim that a PC hardware environment could be capable of performing on a level comparable with mini-computer technology. However, it is important to remember that the processing capability of a typical PC has increased by a factor of twenty fold from 1984 to 1992. A PC built around the Intel 80486 microprocessor chip running at 33MHz has forty times the computing power found in a PC/XT.

This level of service can provide on-line transaction processing capability at a cost of $2,000 per transaction/second (TPS). This cost is much less per TPS than existing mini-computer and mainframe systems can provide. Using proprietary mini computers you can expect to spend from $25,000 to $40,000 per TPS. IMS based MVS mainframe environments typically yield a cost of $50,000-75,000 per TPS. Alternatively, using the combination of MVS and DB2 as a transaction processing engine will typically cost over $100,000 per TPS. What all this means is that based upon full development, maintenance, hardware, software and staff costs, SQL client-server computing is likely to result in finished systems that cost only a small fraction of what building transaction systems has cost in the past. Actual case studies confirm this type of important savings in finished, delivered systems.

Of course, there are many applications which are simply too large to contemplate running on (even a fast) PC. Client server architecture allows you to design the

application once and then, without change, port that application to whatever hardware server has the database processing power to manage your database. This fact allows development on PC style servers and then porting to the new generation of "super servers", minicomputers built to run open operating systems and powered by multiprocessing versions of merchant CPU chips. The approach is to take microprocessor based technologies and combine them with high speed buses, channels and parallel computing architectures to create platforms that can run with the fastest mini-computers. Vendors such as Compaq, Pyramid and Sequent are building parallel processing machines using CICS or RISC microprocessor units which are capable of reaching a sustained processing capability of 100's of MIPS. Do not be surprised, then, to see a combination of these new hardware systems with software from companies like Sybase, Gupta, Novell, Microsoft and Oracle delivering computing technologies comparable to IBM's largest machines but at a tiny fraction of the price.

Most existing users have employed client/server database computing architectures to go after the smaller end of data base solutions. The really large, tough performance based applications like retail credit card verification or airline reservations, require reliable processing of hundreds of transactions/second and are still relegated to mainframes only.

But in the future, I expect multiprocessor based client/server architectures to take on mainframe types of applications. It is very reasonable to think of products like Oracle and Sybase in combination with high-end super servers from companies such as Solbourne, Pyramid, Concurrent, Compaq, IBM or DEC. This high end super server hardware is typically built with parallel Intel 386, 486, and/or RISC chips from MIPS or Sun. By configuring a server with a multiprocessor design and an open operating system which supports it (e.g. UNIX, VINES, NT, or OS/2 & LAN MANAGER), a vendor can build a machine with hundreds of MIPS processing power and 250GB of disk data storage at a cost of well under $500,000. Combining this technology with high speed channels and a client/server DBMS allows a configuration of new technology hardware and database server to be considered as a replacement for a $14 million IBM System 390 running DB2. With a potential savings of almost 95%, this would appear to be an offer well worth considering for many situations.

VI. CONCLUSION - A REALITY CHECK

The advantages of distributed processing and distributed DBMS are well documented and considerable, especially for companies that wish to take advantage of new computing styles that feature graphical interfaces and distributed implementation. Migrating to this new technology, however, requires serious investments in training and the building of new types of systems expertise for companies. There are some potential problems associated with taking advantage of the advanced capabilities of distributed databases. The list below is a quick summary of some of the problems associated with this technology.

1. Communication costs can be quite high; using a 2-phase commit protocol generates lot of communications traffic.
2. There is need for gateway technology to handle SQL differences amongst the different DBMS vendors.
3. The predictability of total costs for distributed queries is variable. In other words it is hard to predict ahead of time how much it is going to cost you to get a job done.
4. Supporting concurrency along with deadlock protection is a very difficult technology.
5. Supporting full recovery with fail over reconstruction is very expensive.
6. Performing a JOIN across different physical nodes is very expensive using today's technology and networks.
7. 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.
8. The job of the database administrator is more difficult than in a single computer because all of the existing skills and requirements are still there, plus the integrity, optimizer, communication and data owner issues of the distributed world.
9. Data security issues are not well understood or proven. It would appear that a distributed environment is more susceptible to security breaks than a database contained in one box.

Please recognize that the above list is potential pitfalls that await (in most cases) the advanced user of this new technology. As in the case of most new technologies, the well advised user takes simple steps while the approach is mastered, before moving onto the more complex. Many companies will find the client/server approach to be

simpler to implement initially. Investments made in such an approach will likely migrate forward to a true distributed database, later if desired.

At a rate of 50 TPC-B transactions/second a (currently) large PC is capable of running a SQL DBMS and delivering services comparable to most of the IMS applications in existence today. The ability to create those applications with the ease associated with SQL databases and GUI screen painters is something that we only could have dreamed on in the mid-80s. Prototyping approaches to building those applications means that significant time savings will be realized in building the better looking and more flexible 1990's approaches. Surely the era of PC LAN based systems has arrived and will dominate the systems building paradigm for the foreseeable future.

END