Distributed and Client/Server DBMS: Underpinning for Downsizing

3.1 Introduction to the chapter

One of the key trends in modern computing is the ***downsizing*** and distributing of applications and data. This paradigm shift is occurring because companies want to take advantage of modern micro-processor technology which allows them to benefit from the new styles of software which employ graphical user interfaces (GUI). ***Client/server*** and ***distributed database*** technologies are two fundamental enabling technologies involved in downsizing.

Client/server approaches allow the distribution of applications over multiple computers. Usually the database(s) resides on server machines while applications run on client computers. While the type of computer used as a server varies widely (e.g. you could have a mainframe, mini-computer, or PC), most clients are PCs. ***Local area networks*** (LANs) provide the connection and transport protocol used in linking clients and servers.

A ***distributed database*** offers capabilities similar to client/server databases. The most fundamental difference between the two architectures is that the distribution of data within a distributed database is both pervasive and invisible. In this style, a database management system (DBMS) resides on each node of the network, and allows transparent access to data anywhere on the network. This means that the user is not required to physically navigate to the data.

The distributed database setup is different from the client/server approach in which the application must be aware of the physical location of data, at least to the extent of on which server it's located. With a distributed database, once an SQL query or remote procedure call is directed to the appropriate server, its query optimizer for SQL will handle the internal database navigation. Many of the advanced functions described later in this chapter such as stored procedures, triggers, and two-phase commits, are available in both client/server and distributed DBMS environments.

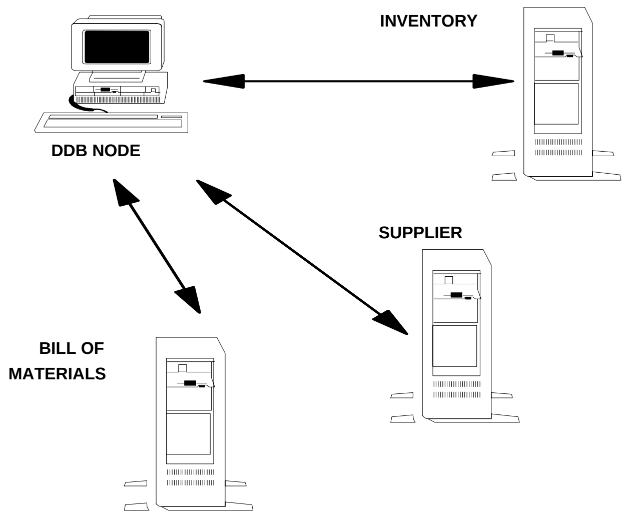
Client/server DBMS and distributed DBMS have much in common as will be discussed in this chapter. Both are based on the SQL language, invented in the 1970s by IBM, and standardized by ANSI and ISO as the common data access language for relational databases. Both are appropriate for distributing applications.

3.2 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 Almaden, California. Most of the original research on distributed database technology for relational systems took place at IBM Corporation's two principal California software laboratories, Almaden and Santa Theresa. The first widely discussed distributed relational experiment developed within IBM's laboratories was a project named R-Star. It is because of IBM's early use of the word *"Star"* in describing this technology that most distributed database systems have *"Star"* incorporated into the name. Today, the market for distributed DBMS is almost entirely based on the SQL language and extensions. (The principal exception is Computer Associates which inherited IDMS and DATACOM prior to relational systems, and has implemented distributed versions both with and without SQL).

Distributed DBMS products can be thought of as occupying the Mercedes Benz echelon of the market place. These products support a local DBMS at every node in the network along with local data dictionary capability. This requirement that a piece of the DBMS exist on each node is the essential difference between distributed databases and client/server systems. In a client/server approach, the DBMS resides on one (or a few) nodes, rather than all of them, and is accessed from a requester piece of software residing on the client.

DISTRIBUTED DATABASE vs CLIENT SERVERS



ONE SOLUTION IS TO USE
A TRUE DISTRIBUTED DBMS
WITH A NODE IN THE CLIENT.
THIS NODE DOES THE JOIN
OPTIMIZATION USING THE
SYSTEM DICTIONARY.
THIS NODE CAN ALSO HANDLE
DISTRIBUTED TRANSACTIONS
AND CAN SUPPORT A GLOBAL
SCHEMA VIEW.

The market for distributed DBMS has grown slowly for two reasons: 1) users aren't sure of how to use the products, and 2) vendors are taking the better part of a decade to deliver a full range of functionality. Another important and unanswered concern is that companies don't know what to expect for communications costs for functions that have historically been run internal to single computers.

3.3 Introduction to Client/Server Database Computing

If distributed DBMS products represent the top tier of the market, then client/server DBMS engines are the Fords and Chevrolets. By accepting a reduction in functionality from what a distributed DBMS provides, vendors have developed client/server DBMS that run exceedingly well on modern PCs 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 that of distributed DBMS.

Much of the impetus for downsizing comes from the fact that many companies want to implement applications that were previously forced to reside on mainframes, onto faster, cheaper PCs. But, before committing to downsize such applications, assurances about the integrity of the data and applications are necessary. In addition, PCs, as well as LANs, have had reputations for not offering a mainframe level of security. Client/server computing is a solution that combines the friendly interface of the PC, with the integrity, security and robustness of the mainframe. Server databases located on PC LANs use implementations of the SQL database access language — the standard database language used on mainframes. Once you've decided to build a client/server environment, you will be on your way 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 too different from that of a distributed DBMS. The key difference is that a client/server approach places the DBMS and DBMS dictionary at certain designated nodes where the data resides. The client program is required to navigate the system and find the correct server node for access to the necessary data. An important advantage of the client/server approach over distributed databases is that having only one (or a few) database locations appears to be more manageable than an architecture which spreads data evenly across many nodes. Managing a distributed database properly would seem to be the more difficult challenge.

3.3.1 The history behind the client/server

The idea for client/server computing grew out of database machine approaches. Sybase's Robert Epstein was working for Britton Lee when he envisioned creating a database machine environment 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 (written in a 4GL), and a back-end (server) which handled the DBMS chores. The advantage of this idea was that the back-end (the virtual database machine) could physically be moved out onto a different piece of hardware if desired. What made this different from Britton

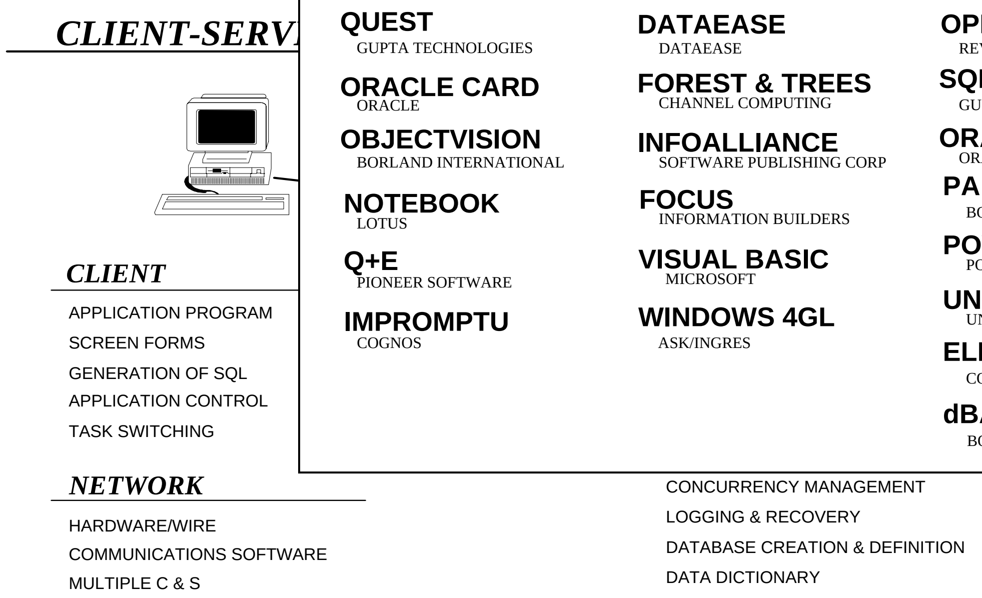
Lee's traditional approach was that Epstein planned for the server to be a generic VAX, UNIX, or PC machine, rather than a unique, custom built database machine. By moving the database machine onto a standard piece of hardware, Sybase picked up the advantage of a vastly improved price performance for generic small systems.

About the same time that Epstein was starting Sybase, Umang Gupta (at that time a Senior Oracle executive) had pictured the same situation and left Oracle to form Gupta Technologies, a company which has emerged as a leader in PC-based, client/server DBMS and tools. Bing Yao, the former University of Maryland professor who founded XDB Systems, was another early developer of client/server approaches to database computing.

By now, most SQL DBMS vendors have jumped into the client/server game. One exception is IBM; when IBM talks about client/server computing, what they are really referring to is distributed computing. IBM is in the process of building a fully functional, distributed architecture for all of 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, and network.

3.3.2 The client



CLIENT/WINDOWS 4GLs

END USER	***OCCASIONAL PROGRAMMER***	***PROFESSIONAL PROGRAMMER***
QUEST GUPTA TECHNOLOGIES	**DATAEASE** DATAEASE	**OPEN INSIGHT** REVELATION TECHNOLOGIES
ORACLE CARD ORACLE	**FOREST & TREES** CHANNEL COMPUTING	**SQL WINDOWS** GUPTA TECHNOLOGIES
OBJECTVISION BORLAND INTERNATIONAL	**INFOALLIANCE** SOFTWARE PUBLISHING CORP	**ORACLE SQL FORMS** ORACLE
NOTEBOOK LOTUS	**FOCUS** INFORMATION BUILDERS	**PARADOX** BORLAND INTERNATIONAL
Q+E PIONEER SOFTWARE	**VISUAL BASIC** MICROSOFT	**POWERBUILDER** POWERSOFT
IMPROMPTU COGNOS	**WINDOWS 4GL** ASK/INGRES	**UNIFACE** UNIFACE
		ELLIPSE COOPERATIVE SOLUTIONS
		dBASE IV, Server Edition BORLAND INTERNATIONAL

The client is where the application program runs. Normally, client hardware is a desktop computer such as an IBM PC, PC clone, or Apple Mac. The application program itself may have been written in a 4GL or third generation language such as C or COBOL. There is an entire new class of Windows 4GLs that allows the painting of applications under leading desktop, Windows-based, operating systems.

Such Windows 4GLs support both windows-oriented application development and execution. Leading examples now on the market include: Powersoft's PowerBuilder, INGRES's Windows 4GL, and Gupta's SQL Windows. Using any of these application building approaches will result in a runtime configuration where the I/O and application controls 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: Macintosh, Windows 3.x for DOS, Presentation Manager, Open Look, and Motif for UNIX.

3.3.3 The network

The network connects the clients and server(s). Normally, networks are based on either Ethernet or Token Ring topologies, and have appropriate interface cards in both the client and server boxes. The communications software typically handles different types of transportation protocols such as SPX/IPX, LU6.2, and TCP/IP. Most network environments provide support for multiple clients and servers.

3.3.4 The server

The server is responsible for executing SQL statements received from a client. Sometimes data requests are not communicated through SQL, but through a remote procedure call which triggers a series of pre-compiled, existing SQL statements.

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 requester validation.

The server will also handle 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 SQL statements, the server handles security and provides for concurrent access to the data by multiple users.

3.3.5 The benefits of using SQL

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 in the database management system on the server solves this problem since only the necessary query response data (a table) is transmitted to the client machine.

Having SQL on the server also allows the database implementation of advanced facilities such as triggers and automatic procedures. As relational DBMS evolve, they will confer the ability to build rules directly into the database engine. 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 mini-computers and mainframes as servers, the largest market likely to develop will have a mix of OS/2, Macintosh, Windows 3.x, Windows NT, and MS-DOS on the client and either UNIX, Windows NT, NetWare, or OS/2 for the server. Server software will provide mainframe levels of 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.

3.4 More Details on Distributed DBMS

Distributed DBMS are where the most interesting action is happening in the large systems DBMS market (mini-computer to super computer). As SQL emerges as the standard DBMS language, the principal methods by which DBMS vendors are differentiating their products is by adding various functions including:

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

Distributed database software needs to provide all of the functionality of multi-user mainframe database software, while allowing the database itself to reside on a number of different, physically connected computers. The types of functionality distributed DBMS must supply include data integrity, maintenance through automatically locking records, and the ability to roll-back transactions that have been only partially completed. The DBMS must attack deadlocks to automatically recover completed transactions in the event of system failure. There should be the 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 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 advanced functions for a distributed DBMS. However, it won't be helpful to use this section 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.

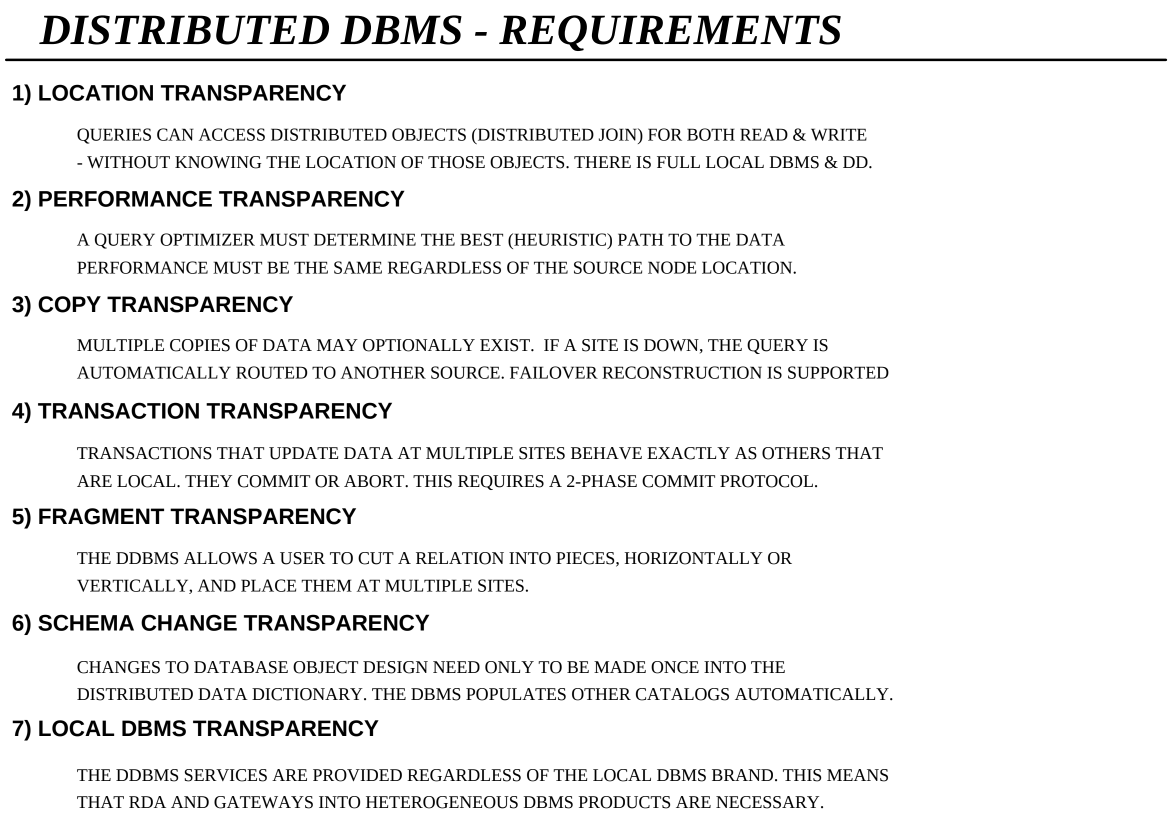
3.4.1 Basic requirements for a distributed DBMS

- ***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 a change in the logical view requires no change of the application programs. There is support for a distributed JOIN. In order to meet this requirement, it is necessary for a full local DBMS and data dictionary to reside on each node.

- ***Performance transparency*** It is essential to have a software optimizer 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. This type of technology is rather primitive at this time and will be discussed later in this chapter.

- ***Copy transparency*** The DBMS should optionally support the capability of having multiple physical copies of the same logical data. Advantages of this functionality include superior performance from local, rather than remote, access to data, and non-stop operation in the event of a crash at one site. If a site is down, the software must be smart enough to re-route a query to another data source. The system should support fail over reconstruction: when the down site becomes live again, the software must automatically reconstruct and update the data at that site.

- ***Transaction transparency*** The system needs to supports transactions that update data at multiple sites. Those transactions behave exactly the same as others that are local. This


DISTRIBUTED DBMS - REQUIREMENTS

1) LOCATION TRANSPARENCY

QUERIES CAN ACCESS DISTRIBUTED OBJECTS (DISTRIBUTED JOIN) FOR BOTH READ & WRITE
- WITHOUT KNOWING THE LOCATION OF THOSE OBJECTS. THERE IS FULL LOCAL DBMS & DD.

2) PERFORMANCE TRANSPARENCY

A QUERY OPTIMIZER MUST DETERMINE THE BEST (HEURISTIC) PATH TO THE DATA
PERFORMANCE MUST BE THE SAME REGARDLESS OF THE SOURCE NODE LOCATION.

3) COPY TRANSPARENCY

MULTIPLE COPIES OF DATA MAY OPTIONALLY EXIST. IF A SITE IS DOWN, THE QUERY IS
AUTOMATICALLY ROUTED TO ANOTHER SOURCE. FAILOVER RECONSTRUCTION IS SUPPORTED

4) TRANSACTION TRANSPARENCY

TRANSACTIONS THAT UPDATE DATA AT MULTIPLE SITES BEHAVE EXACTLY AS OTHERS THAT
ARE LOCAL. THEY COMMIT OR ABORT. THIS REQUIRES A 2-PHASE COMMIT PROTOCOL.

5) FRAGMENT TRANSPARENCY

THE DDBMS ALLOWS A USER TO CUT A RELATION INTO PIECES, HORIZONTALLY OR
VERTICALLY, AND PLACE THEM AT MULTIPLE SITES.

6) SCHEMA CHANGE TRANSPARENCY

CHANGES TO DATABASE OBJECT DESIGN NEED ONLY TO BE MADE ONCE INTO THE
DISTRIBUTED DATA DICTIONARY. THE DBMS POPULATES OTHER CATALOGS AUTOMATICALLY.

7) LOCAL DBMS TRANSPARENCY

THE DDBMS SERVICES ARE PROVIDED REGARDLESS OF THE LOCAL DBMS BRAND. THIS MEANS
THAT RDA AND GATEWAYS INTO HETEROGENEOUS DBMS PRODUCTS ARE NECESSARY.

means that transactions will either all 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 when necessary to answer queries.

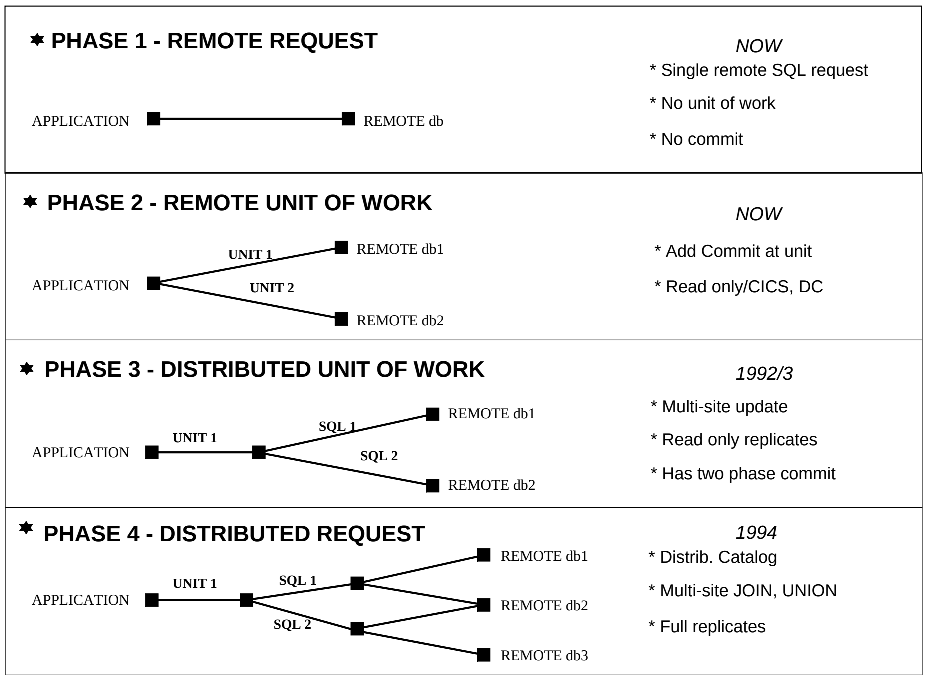
- ***Schema change transparency*** Changes to database object design need only be made once into the distributed data dictionary. The dictionary and DBMS automatically populate other physical catalogs.

- ***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.

3.4.2 IBM's four ways to distribute data

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

IBM's APPROACH TO DISTRIBUTED DATABASE



- ***Extracts*** provide the ability to extract data. This simply means that there exists 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 provide an alternative method for decision support access to production data.

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

- ***Replicates*** are a more sophisticated version of the distributed DBMS capabilities classified under copy transparency. 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 two-phase update commit protocol.

3.4.3 Software Optimizers

When a DBMS is spread over many different physical sites, the cost difference between the best and worst ways of accomplishing a function such as a JOIN can easily be a million to one. Because of this, a distributed DBMS absolutely must have a cost-based software optimizer. Without a cost-based optimizer, navigation to data is under programmer control, violating a basic precept of relational theory (this is what must be done with several earlier RDBMSs such as Oracle prior to 7.0). Without such an optimizer, only known queries can be handled, since the performance of an unanticipated query may be extremely poor.

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 the types of issues that should be dealt with are:

1. How busy are the various machines on the network?
2. What are relative speeds of these machines?
3. What are sizes of the tables that have to be accessed?
4. How are the tables organized?
5. What is the line speed between various nodes on the network?

6. How busy are the lines between the various nodes?
7. What are the access patterns in indexes?
8. Where should software optimizer itself run?

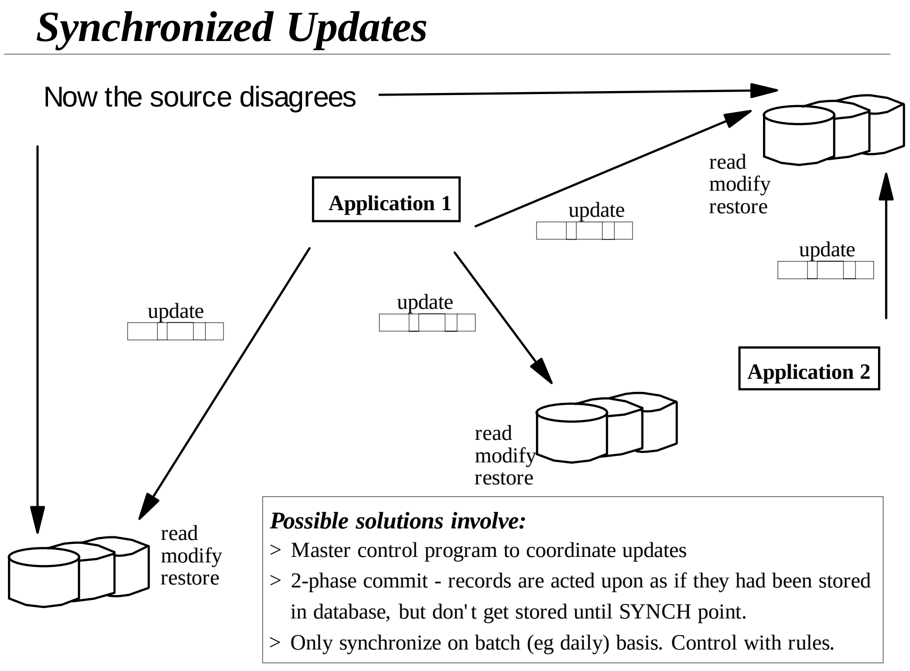
3.4.4 Two-phase commit protocol

The goal of the two-phase commit protocol is to allow multiple nodes to be updated synchronously as result of a group of SQL statements which are either committed or rejected together.

The general procedure for a two-phase commit 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 a *"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 two-phase protocol is one of the most challenging tasks. The additional complexity in this type of software comes from the fact that there are different types of failure nodes, and the software needs to recover from any combination of failures over all of the supported environments. For the user, operation in an environment requiring a two-phase commit may be very costly. The extra cost is incurred since a two-phase commit requires an extra round-trip message above the normal amount of messages that occur in single computer systems.



Possible solutions involve:
> Master control program to coordinate updates
> 2-phase commit - records are acted upon as if they had been stored in database, but don't get stored until SYNCH point.
> Only synchronize on batch (eg daily) basis. Control with rules.

There are no standards for implementing a two-phase commit. Various vendors have offered different, partial implementations. It is likely that we will see a future ISO standard dealing with two-phase commit protocol.

3.4.5 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 provide for read-only access, while higher levels of function allow write access also. This higher level should be accompanied by a two-phase commit capability across the different systems (general availability of this capability is still in the future.)

  Distributed access is a technology that is closely related to distributed DBMS. Distributed access is about the building of gateways that allow one DBMS to access data stored in another. This can properly be thought of as a subset of the technologies being delivered by vendors selling distributed DBMS or client/server DBMS technologies. The demand for distributed access, of course, is greatest in popular mainframe file and database environments such as IBM's IMS, DB2, VSAM, and DEC's Rdb. This is because local DBMS capability is not a requirement for distributed access. Instead, most vendors provide a piece of software known as a requester to be run on the client side of the RDA environment. Some of the products in this market are not finished gateways, but tool kits so that users can build custom gateways.

- ***Relational Integrity*** An important server function that supports 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 using stored procedures or program triggers.

- ***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 an 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 triggers are stored in the catalog and automatically executed, they promote consistent integrity constraints across all transactions. Triggers are easy to maintain because they are stored in only one place. They result in rules that are enforced for any application that accesses the database, such as spreadsheet programs.

- ***Multi-Threaded Architecture*** For the best performance, 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 — 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 33 MHz 80386 PC with 10 MB of RAM.

- ***Symmetric Multi-processing*** Another advantage of DBMS servers is direct support of multi-processor hardware architectures in a symmetric multi-processing (SMP) mode. Most operating systems either currently (UNIX, Windows NT, VMS) or soon will (OS/2) offer support for this functionality. Therefore, there needs to be effective integration between the DBMS and operating system to take advantage of the potentially improved throughput.

  Direct support for SMP means the DBMS can take advantage of several parallel processors under the same skin (with an appropriate operating system). These processors can be either tightly or loosely coupled. As of mid-1992, 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, programmers are also able to move backward within a cursor. Without a cursor, it's harder to program transactions to 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 of binary data. Another useful item is TEXT data types which are printable character strings.

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

- ***Multi-platform implementations*** Another primary advantage of a robust DBMS is multi-platform portability and networking. If your software runs on many different vendor's hardware, then you have that much more flexibility. For example, Oracle was built with an approach that has outdone 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 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. Disk mirroring is supported through the process called "shadowing." This is a very useful facility for applications that require extremely low amounts of down-time - if one disk fails, then the system will automatically divert and use the other disk without interrupting operations.

- ***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.

- ***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 that indicates that a database change has been committed. Event alerters work remotely and can span across multi-vendor networks. Although it would seem to be simple 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:

- The waiting programs consumes neither network traffic nor CPU cycles.
- 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 running outside of the database.

- ***Multi-dimensional Arrays*** In scientific processing or time-series commercial types of applications, array support for the database is useful. Arrays are stored as a single field in a record so that retrieval is expedited. Arrays are widely used in scientific processing and are very expensive to normalize for a relational DBMS — normalization typically means creating redundant data to generate separate records for information that is really only different at the field level.

3.5 More on Client/Servers

Almost all of the advanced functions listed in the previous section on distributed DBMS (such as BLOB data types, RPCs, and event alerters) are also available from leading DBMS server vendors. To repeat our previous definition, the primary difference between a server DBMS and a distributed DBMS is whether or not each node on the network has a full copy of the DBMS. Added functionalities 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. The domination of this new marketplace is being battled for between companies including Microsoft, Revelation Technologies, Borland, Sybase, Oracle, and Powersoft. The reasons behind 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) as the capabilities can be brought to market sooner and at a lower price.
- Developers can use PCs instead of time-sharing terminals as primary development platforms.
- Even though the PC is used as the principle platform, security, integrity, and recovery capabilities comparable to mini-computers are the result.
- The efficiency of SQL queries and transmissions greatly reduces the network communication load (from that of a PC LAN/file-server-based approach).
- Gateway technologies, which are an important component of client-server computing, 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 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 to the network, or servers can be traded up for the latest generation of micro-processors.

- A lot of flexibility comes from a computing environment based upon SQL since the language is a standard. Commitment to an SQL server engine will mean that most front-end, 4GL, spreadsheet, word-processing, and graphics tools will be able to interface to an SQL engine.
- Client/server computing provides the industrial strength security, integrity, and database capabilities of mini-computer or mainframe architectures, while allowing companies to build and run their applications on relatively inexpensive PC and mini-computer networks. The use of this hardware and software combination can cut 90% of hardware and software costs when building "industrial strength" applications.

The client/server model offers users choices between many different hardware and software platforms. The hardware choices are too expansive to be listed here, but the principal choices for operating systems are multi-user, multi-tasking, protected products such as UNIX, OS/2, Windows NT, and NetWare. The micro-processor 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 powerful PC, that has enough power to run applications on top of multi-tasking, single-user operating systems such as Windows 3.1 or OS/2.

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. For these machines to play a role in future networks, however, it is clear that they will have to adopt server functionality by acquiring and supporting emerging downsizing standards such as UNIX, NetWare, LAN Manager for Window's NT, and LAN Server for OS/2.

PLAYERS IN THE SERVER MARKET

GUPTA TECHNOLOGIES, INC.	SQLBase
IBM	OS/2EE
INFORMIX SOFTWARE INC.	INFORMIX ONLINE
ASK/INGRES DIVISION	INTELLIGENT DATABASE
MICROSOFT/SYBASE	SQL SERVER
NOVELL	NETWARE SQL
ORACLE	ORACLE SERVER
SYBASE	SQL SERVER
XDB SYSTEMS INC.	XDB-SERVER
BORLAND/INTERBASE	INTERBASE
PROGRESS SOFTWARE	PROGRESS
COMPUTER ASSOCIATES	IDMS/R, DATACOM
DEC	Rdb, ACMS

3.5.1 Performance from a client/server environment

The reader might be skeptical of the claim that PCs 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 in abidance of the Transaction Processing Council's standards have shown that products like ORACLE and Gupta's SQLBase are capable of running about 50 transactions per second (TPS) on 486-based PCs. This number would not be an unreasonable result for a mainframe running IBM's DB2.

The transaction capabilities of client/server software working with low-end PC servers or super-servers (mini-computer style cabinets built with merchant micro-processors such as the 80486 or R4000) is quite astounding. For example, at the low-end of the hardware scale, both Gupta's SQLBase and Microsoft's SQL Server can run on Intel 80486-based PCs processing approximately 18 TPC-B TPS (Transaction Processing Council, database Benchmark B). PC hardware can support disks with 12-milli-second access time and 4 MB to 6 MB transfer rates. Such a machine can be configured with 600 MB 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 TPS would be 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 between 1984 and 1992. A PC built around the Intel 80486 micro-processor chip running at 33 MHz has forty times the computing power of a PC/XT.

This high level of service can provide on-line transaction processing capabilities at a cost of $2,000 per 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 between $25,000 and $40,000 per TPS. IMS-based MVS mainframe environments

__DATABASE SERVER PERFORMANCE__

LOW END

✳ 486 PCs, LOW END RISC, 12 ms ACCESS/4MB TRANSER RATE
✳ 10 - 20 MIPS @ $6,000 TO $18,000
✳ 8 - 15 TPC-B/SEC
✳ 90 WORKSTATIONS SIMULTANEOUSLY A SINGLE SERVER
✳ 250 ATM's ON A SINGLE SERVER
✳ ETHERNET - 100 TPS ACROSS NETWORK

HIGH END

✳ PARALLEL CISC OR RISC GIVES 100's OF MIPS
✳ SCSI AND IPI CHANNELS - COMPARABLE TO 3090 CHANNELS

RESULT:

* OLTP AT $1K - $4K/TPS

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 of 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 architectures allow you to design the application once and then, without change, port that application to whichever server has the database processing power you need to manage your database. This allows application development on PC-style servers, with the porting to the new generation of "super servers", mini-computers built to run open operating systems powered by multi-processing versions of merchant CPU chips. The approach is to take micro-processor-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 micro-processor units capable of reaching a sustained processing capability of 100s 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.

As a first project, it is clearly better to use client/server computing for mostly-read or decision support environments. The very large, tough performance-based applications, such as retail credit card verification or airline reservations, require reliable processing of hundreds of transactions/second and are still relegated to mainframes only.

In the future, I expect multi-processor-based client/server architectures to take on mainframe types of applications. It is very reasonable to envision 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 multi-processor design and an open operating system which supports it (e.g. UNIX, VINES, NT, OS/2, or Lan Manager), a vendor can build a machine with hundreds of MIPS processing power and 250 GB of disk data storage for 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.

3.6 Conclusion - A Reality Check

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

1. Communication costs can be quite high and using a two-phase commit protocol tends to generate a considerable amount of communications traffic.
2. There is the need for gateway technology to handle the SQL differences among different DBMS vendors.
3. The predictability of total costs for distributed queries is variable. In other words, it is difficult to predict how much it will cost to get a job done.
4. Supporting concurrency, in addition to deadlock protection, is very difficult.
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, reasonable for single computers, are difficult and expensive across distributed networks (e.g. the enforcing of semantic integrity restraints).
8. The job of the database administrator is more difficult because, above and beyond their current functions, they need to understand the integrity, optimizer, communication, and data owner issues of the distributed world.
9. Data security issues are neither well understood nor proven. It would appear that a distributed environment is more susceptible to security breaks than is a database which is contained in one box.

Please recognize this as a list of 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 would take small steps while the approach is mastered, before moving onto the more complex conversions/implementations. Many companies will find the client/server approach to be simpler to implement initially. Investments made in such an approach will likely migrate towards a distributed database if later 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 about in the mid-1980s. Prototyping approaches in building those applications means that significant time-savings will be realized in better looking and more flexible 1990s approaches. The era of PC LAN-based systems has arrived, and will dominate the systems building paradigm for the foreseeable future.

QUESTIONS:

Students

1. What is the fundamental difference between a distributed database and a client/server design?
2. Is the criticism that downsized, PC-based database architectures aren't as robust and secure as mainframe environments valid?
3. In a client/server environment, which computing functions are located on the client(s) and which can be found on the server(s), and what are the types of hardware used for each of these machines?

Academic

1. Do you believe that there will be a role for mainframes in downsized computing environments?

Practitioners

1. Why, at this point in time, are more companies choosing to implement client/servers architectures rather than distributed databases?
2. Given that downsizing is a fundamental paradigm shift in the computing industry, what are the effects going to be on both the PC and mainframe markets?