Abstract

This report, *The Foundation for Downsizing: Distributed and Client/Server Database Management Systems (DBMS),* provides a comprehensive overview of the what, why, and how of distributed and client/server computing. It explains the benefits of this technology and its usefulness in both decision support and transaction processing environments. Client/server and distributed DBMS technologies are related, but not identical. This report explains the differences between the two approaches.

Other topics covered include a criteria list for describing the functionality expected of a true distributed DBMS. Some advanced levels of functionality are described as well. Examples of successful case studies are used to prove the point that client/server and distributed DBMS technology is definitely ready for serious transaction processing workloads.

The report concludes with a list of advisories and cautions for MIS managers preparing to take the client/server route.

About the Author

Dr. George Schussel, President of DCI, is a world-renowned authority on distributed and client/server DBMS technology, and acts as a consultant to Fortune 500 companies in this capacity.

Introduction

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 microprocessor technology that allows them to cut costs and add flexibility, while also allowing them to benefit from the new styles of software which employ graphical user interfaces (GUIs). Client/server and distributed database technologies are two fundamental, enabling technologies involved in downsizing.

The benefits of client/server computing are so compelling that most companies find them impossible to ignore. These benefits are:

1. *Cost Savings* - By distributing processing over a number of microprocessor-based computers, networked together, rather than a few larger machines, users can take advantage of computer instruction cycles selling for $100/MIPS (one million instructions per second) on PCs, versus cycles selling for $60,000/MIPS (discounted prices) on mainframes.

2. *Scalability* - The modular nature of client/server approaches means that such networks can be easily expanded by adding more nodes, or migrating to newer processors on existing nodes. This offers a developer or user the tremendous advantage of keeping application software constant, as the size of the application moves from a small local area network (LAN) to environments as large as supercomputers.

3. *Faster Application Development* - Development of client/server applications is now available with database-oriented fourth-generation development environments such as INFORMIX-4GL. These advanced higher-level tools offer an easy-to-use application development environment. Applications can be built in one-quarter of the time, at less the cost, and by less experienced programmers.

4. *GUI Interfaces* - There is little doubt that graphical interfaces such as Windows, Motif, and Presentation Manager offer many tutorial, usage, and productivity benefits for users. In addition to straight DOS and other character-based environments that can be supported on the client, many client/server users are building on these GUI bases for their client-side computing.

5. *Interoperability within the Desktop* - As Windows evolves to become a standard on the desktop, facilities like the Windows clipboard allow the tight coupling of typical desktop applications such as word processing with data processing. For example, the results of an SQL-based query (a data results table) can be cut and pasted directly into a Word document or Excel spreadsheet.

6. *Robustness* - One of the disadvantages of downsized systems based on file server approaches was that the security and integrity functions of true DBMS-based systems were not available on file servers. Client/server approaches, on the other hand, are based on SQL relational DBMS servers, and offer all of the robustness, security, and data integrity of the traditional mainframe computing environment.

Client/server approaches are usually implemented with individual applications running over multiple computers. The database(s) resides on server machines, while the applications run on client computers. While the type of computer used as a server varies widely (e.g., mainframe, minicomputer, workstation, or PC), most clients are PCs. Local area networks provide the connection and transport protocol used in linking clients and servers.

The client/server approach calls for a connection between the application running on the client, and the database running on one server. And, it calls for the division of processing load between the two, such that the client system handles all the application and user interface logic, while the server handles all the data-related operations. (See "The Components of a Client/Server Architecture," page 5, for a detailed breakdown of tasks handled by each.)

A distributed DBMS offers the capabilities of client/server DBMS and more. The most fundamental difference between the two architectures is that the distribution of data within a distributed database is both pervasive and invisible, and involves storing data across more than one server, yet accessing the data as if it were stored in one location. To facilitate this, a DBMS server resides on each database node of the network and allows transparent access to data anywhere on the network. Therefore, the user is not required to physically navigate through the data.

To clarify the distinction, with a distributed DBMS, the "home server" is connected to, and handles the calling of, other servers as needed in a distributed transaction. Whereas in a client/server configuration, the "home server" handles all transactions itself, without the involvement of other servers. The client, in the client/server system, can disconnect from its "home server" and establish a connection with another server to connect to different databases (held on other servers), but there are no distributed transactions between these various servers. In a distributed system, once an SQL query or remote procedure call is directed to the "home server," its query optimizer for SQL will handle the internal database navigation to involve other servers in the transaction, as necessary, to satisfy the client request. Many of the advanced functions described later in this report, such as stored procedures and triggers, are available in both client/server and distributed DBMS environments; while functions that relate to the synchronization of data stored across multiple servers, such as two-phase commits, pertain to distributed DBMS environments only.

Note: In the remainder of this paper the term DBMS represents the system software that controls data (i.e., the database server), while the term database represents the actual data itself.

Client/server DBMS and distributed DBMS have much in common; 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 also appropriate architectures for distributing applications, while distributed DBMS is a necessary architecture for distributing the physical data.

Background on 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 DBMS 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's because of IBM's early use of the word *"Star"* in describing this technology that most distributed DBMS systems have *"Star"* incorporated into their 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 marketplace. These products support a local DBMS at every database node in the network along with local data dictionary capability. The disbursement of physical data across multiple database server nodes is the essential difference between distributed DBMS and client/server systems. In a client/server approach, the DBMS and the physical data reside on one node, rather than multiple nodes, and are accessed from a requester piece of software residing on the client node.

The market for distributed DBMS has grown slowly for two reasons: 1) users aren't sure 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 of communication costs for functions that have historically been run internally on a single, centralized computer. Now, however, with the imminent widespread availability of 100 megabits per second capability across LANs (with fiber and/or copper), concerns about communication costs and availability of advanced feature sets for distributed databases are disappearing. The growth in usage of distributed DBMS software in the 90s is likely to be significant.

Background on Client/Server Database Computing

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

The ability to separate the application logic from the database server processing, across two different computers, has led to more horsepower available at the client system to run the new GUI interfaces, providing users with easier, more intuitive access to the information they need. At the same time, advanced DBMS server capabilities, like two-phase commits and distributed JOINs across multiple servers, are becoming available for distributed DBMS. It's very likely that both markets will merge over the mid-term future.

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, more flexible, smaller systems (often employing PCs and workstations). But, before committing to downsize such applications, assurances about the integrity of the data, and the ability to build sophisticated

applications are necessary. In addition, PCs and LANs have reputations for not offering a mainframe level of security. Client/server computing is a solution that combines the friendly interface of the PC or workstation 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 users have decided to build a client/server environment, they will be on their way to building an applications architecture that will be economical, flexible, and portable well into the future.

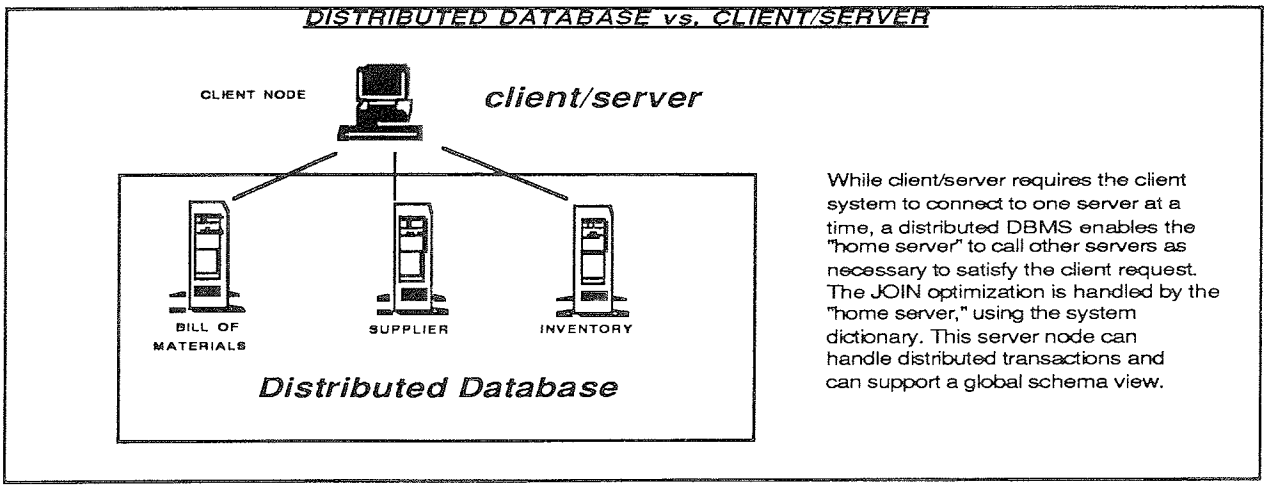
The functionality delivered by today's client/server systems is not too different from that of a distributed DBMS. The key difference is that a client/server approach places the DBMS and DBMS dictionary at a designated node where the data resides. The client system is required to establish a connection with the correct server node for access to the necessary data. This connection can be made transparent to the applications, but it is not transparent to the client system. Conversely, a distributed DBMS is able to provide transparent access to data spread across multiple servers, with no knowledge required on behalf of the client system as to the whereabouts of that data beyond its connection to the "home server." This advanced functionality enables the database administrator to optimize the loading of data across the servers without making any changes at the client systems.

The History Behind Client/Server Architecture

The idea for client/server computing grew out of database machine and relational approaches. One early visionary was Robert Epstein. While working for Britton Lee, Epstein envisioned the creation of a database machine environment with a server that was a virtual machine rather than a physically unique piece of hardware. The system software was then 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 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, this approach picked up the advantage of the vastly improved price performance to be gained from generic systems.

About the same time that Epstein was honing his ideas and starting Sybase to market them (mid 1980s), Informix Software, Inc., a well established relational DBMS vendor, delivered on a re-architecture of its product line, built around the client/server model and distributed DBMS capabilities.

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. However, they are taking several years to develop this approach.

The Components of a Client/Server Architecture

A client/server computing environment consists of three principal components: client, server, and network.

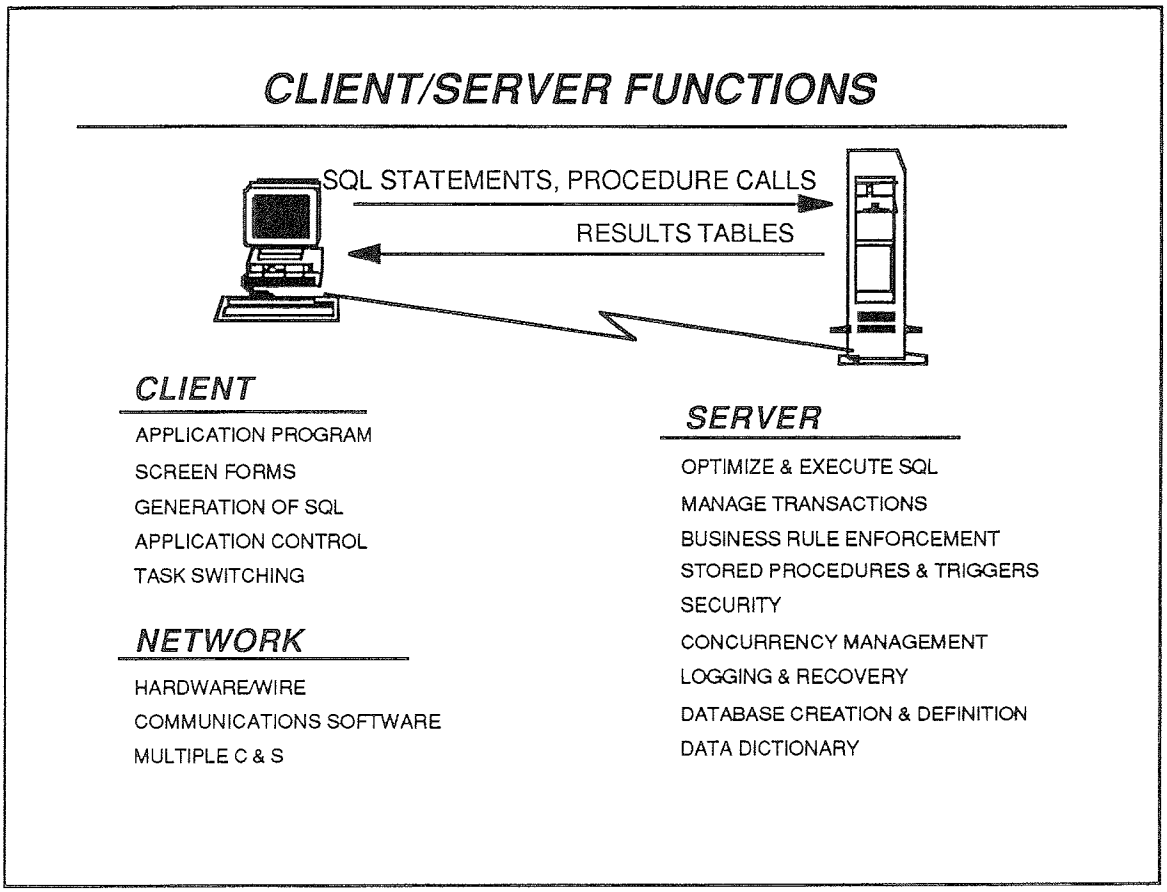
The Client

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 now 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™, JYACC's JAM®, Uniface, and Gupta's SQLWindows. Using any of these application building approaches will result in a runtime configuration where the application logic and controls come from the client, while the database I/O and associated semantics run on the server. At the desktop level, most software will support the emerging windows-based standards: Windows 3.x for DOS, Macintosh, Presentation Manager, Motif, and Open Look for UNIX.

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 machines. 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.

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 that triggers a series of pre-compiled, existing SQL statements.

The server is responsible for SQL optimization, determining the best path to the data, maintaining data integrity, and managing transactions. Some server technologies support advanced software capabilities, such as stored procedures, schema-based integrity constraints, and triggers. The server is 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 recovery, and 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.

The Benefits of Using SQL

An important benefit of the set-oriented SQL language is network efficiency, resulting in enhanced DBMS performance. 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 the database management system on the server solves this problem as only the necessary query response data (a subset of a table(s)) is transmitted to the client machine.

CLIENT/WINDOWS 4GLs

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

Having SQL on the server also allows the implementation of advanced facilities, such as triggers and automatic procedures at the database server. As relational DBMSs evolve, they will confer the ability to build application rules directly into the database server. 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 servers, the largest market likely to develop will have a mix of Windows 3.x, Windows NT, MS-DOS, OS/2, and Macintosh on the client, and either UNIX, Windows NT, NetWare®, or OS/2 on 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, is that SQL client/server technology allows inexpensive PCs to be made into "industrial strength" computing engines. This advantage, coupled with a scalable RDBMS server that can be implemented on a PC server, and moved if necessary to a superserver or minicomputer, will allow users to build flexibility and expansion into their system to accommodate changing needs over time.

More Details on Distributed DBMS

Distributed DBMSs are where the most interesting action is happening in the large systems DBMS market (minicomputer to supercomputer). As SQL emerges as the standard DBMS language, the principal method used by DBMS vendors to differentiate their products is to add various functions including:

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

Distributed DBMS software needs to provide all 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, disk space management, and security. The DBMS must prevent deadlocks and automatically recover completed transactions, as well as roll-back incomplete ones in the event of system failure. It should also have the capability to optimize data access for a wide variety of different application demands. Additionally, 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 focuses on the basic, and then advanced functions for a distributed DBMS. However, do not 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.

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 to 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 database 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 report.

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

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.

Advanced Requirements for a Distributed DBMS

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.

Fragmentation Transparency - The distributed DBMS allows a user to cut relations into pieces horizontally or vertically, and place them at multiple physical sites. The software has a capability to re-combine these

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) 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.

4) 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.

5) 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.

6) FRAGMENT TRANSPARENCY

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

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.

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 evaluate many potential options, and to develop a correct search strategy based upon the results of that evaluation. Examples of the types of issues that should be evaluated follow.

Questions that pertain to the data and its structures:

1. What are the absolute and relative sizes of the tables that have to be accessed?
2. How are the tables organized? Is there an index? How many levels of indexing?
3. What are the access patterns in the indexes?
4. How many rows will result from this query?

Questions that pertain to the extra cost of moving data across a network:

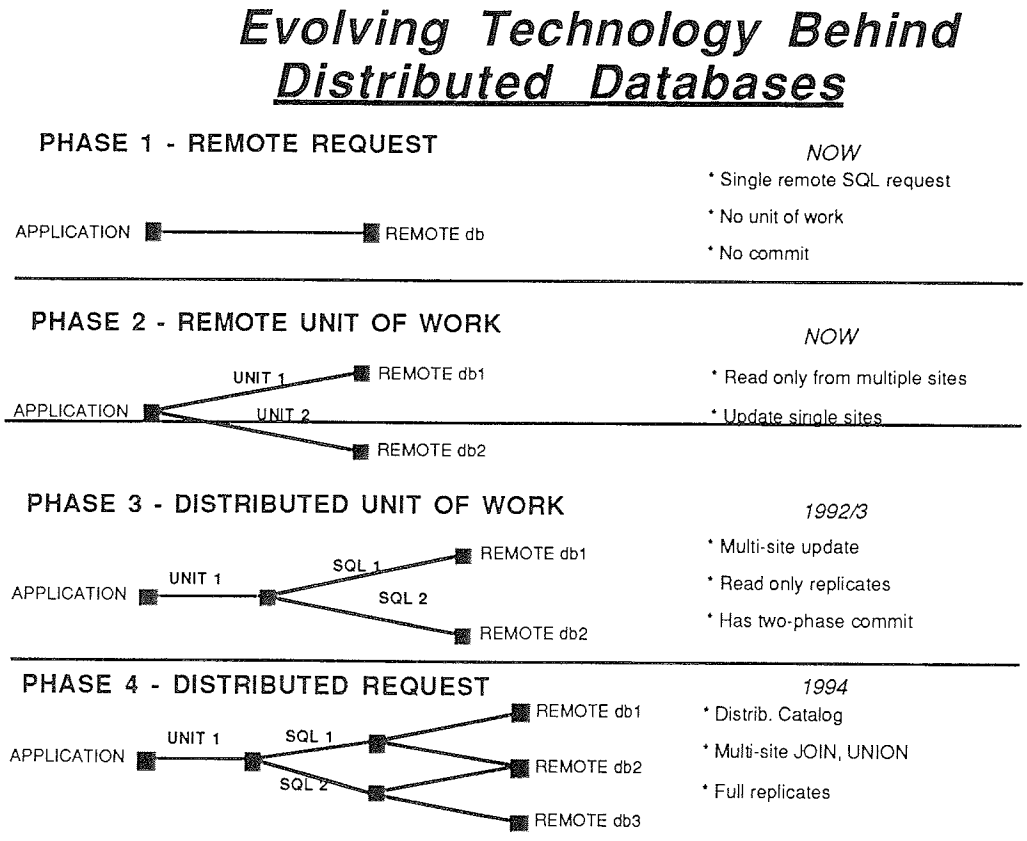
1. What is the distance between the nodes?
2. What is the line speed between these nodes?
3. What are the relative speeds of these node machines?

Based on the answers to the above questions, the optimizer decides where the actual work should take place, and instigates it.

Two-Phase Commit Protocol

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

The general procedure for a two-phase commit follows.



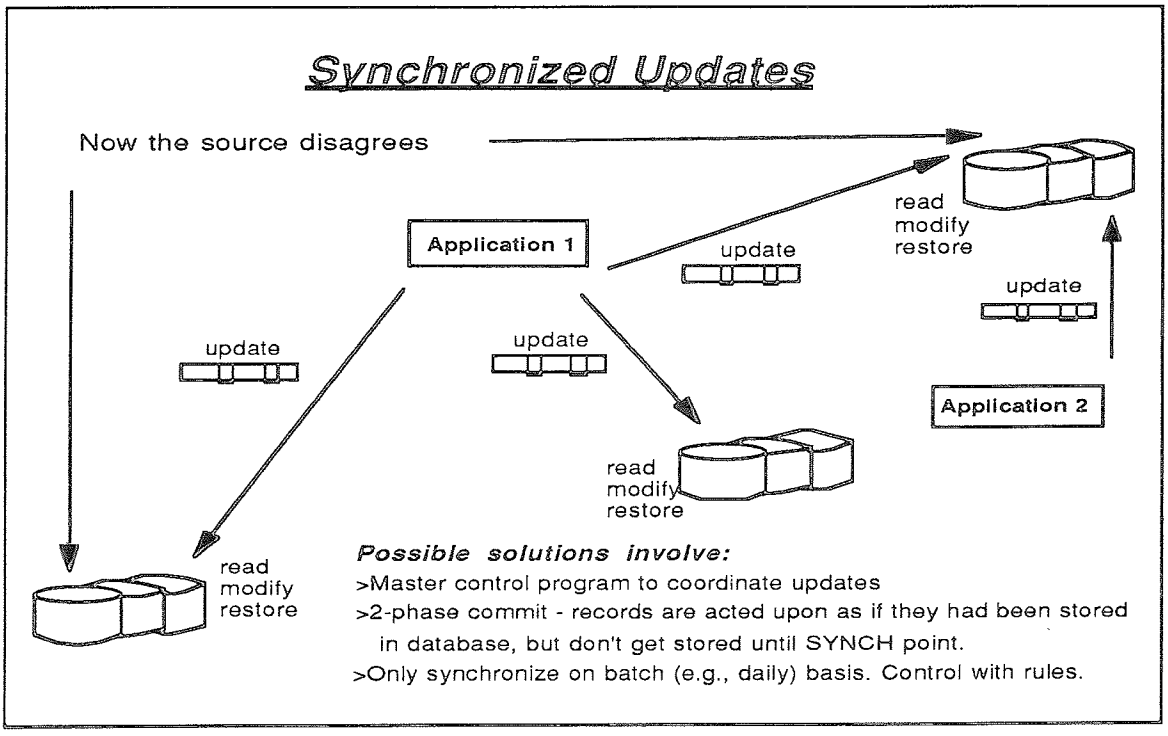
1. One node is designated as a master; the master sends notice of an upcoming query out to all of the slaves.
2. The slaves respond with ready messages when all of the data necessary for the protocol is available.
3. The master sends out a *"prepare"* message to the slaves.
4. The slaves lock and log 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 commit 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 modes, 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 because a two-phase commit requires an extra round-trip message above the normal number of messages that occur in single computer systems.

Standards for implementing a two-phase commit are slowly emerging. Both the ISO and X/Open standards groups have established distributed transaction processing (DTP) standards, and have established a "presumed abort," two-phase commit algorithm as part of this DTP standard. Various DBMS vendors have started to base their implementations on these emerging standards, while others are offering different home-grown implementations.

More Advanced Capabilities for Distributed or Client/Server DBMS

- *Gateways* - Many of the distributed DBMS and client/server DBMS products have optional gateways that allow access to data stored in other DBMSs. Lower levels of functionality provide for read-only access, while higher levels of functionality allow write access, as well. 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*** - A technology that is closely related to distributed DBMS. Distributed access refers to 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 is greatest for 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. Instead, most vendors provide a piece of software known as a requester to be run on the client side of the distributed 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. This can include features such as referential integrity, or the ability to state business rules directly into the database using schema extensions (the ANSI-SQL approach), stored procedures, or program triggers.

- ***Integrity Constraints*** - Integrity constraints are provided to enforce data integrity at the server, thereby providing consistent enforcement across individual applications. The ANSI-SQL standard on integrity constraints provides for the following areas:

  - default values, where no explicit value is supplied;
  - check constraints, where each row entered or updated must pass the check constraint; and
  - referential constraints, which enforce the parent-child (or master-detail) relationship. This will cause the database server to deny any request from the client which would result in the existence of a "child" without the existence of its corresponding "parent" record.

The ANSI standard prescribes that these integrity constraints all be defined as extensions to the schema, rather than relying on programmers to correctly code a stored procedure to handle the multitude of instances of enforcement.

- ***Stored Procedures and Triggers*** - Stored procedures are small SQL programs written in SQL and the stored procedure language, that are stored in the DBMS catalog. Stored procedures can be called by the application logic, and are used to apply common logic across application programs and to reduce network traffic by applying one call across the network that can activate execution of the logic at the server. Triggers can be set to launch a stored procedure when data is modified. Each trigger is associated with a particular table and an SQL DML function (i.e., update, delete, and insert). Triggers are automatically executed whenever a transaction implements the triggering action against the table. Users can write stored procedures and attach triggers to enforce any database validation rule, but triggers are most appropriate for entering business rules that aren't within the scope of the ANSI standard integrity constraints discussed previously.

Since stored procedures and triggers are stored in the catalog and executed at the server, they promote consistent integrity constraints across all transactions. This results in rules that are enforceable for any applications that access the database, such as spreadsheet programs. They are also an aid to maintenance because they are stored in only one place, rather than embedded within many different applications.

- ***Multi-Threaded Architecture*** - For the best distributed, or client/server performance, the DBMS should implement a multi-threaded 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, and share multiple threads within a smaller number of processes. Threads are more efficient than processes, and they use less memory and CPU resources. A multi-threaded DBMS

server can service 10 to 40 users simultaneously on a machine as small as a 33 MHz, 80386 PC with 10 MB of RAM.

- ***Symmetric MultiProcessing*** - Another advantage for DBMS servers is direct support of multiprocessor hardware architectures in a symmetric multiprocessing (SMP) mode. Currently, most operating systems (UNIX, Windows NT, VMS), and soon 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.

- ***Cursors*** - A cursor stores the results of an 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 Byte or Image data types for 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 software runs on many different vendors' hardware, then there is much more flexibility. For example, Informix has been implemented on approximately 400 different hardware systems.

- ***Disk Mirroring*** - For companies wanting the reliability of mainframe environments on a PC LAN, a disk, or a server, database 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 binary large object (BLOB) data type 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 server, should be supported within the DBMS. These macros are 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 DBMS server, BLOB macros are easier 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.

UNIX today, and NT, NetWare, and OS/2 in the future, will support symmetric multiprocessing on the server. This will allow scalability of database applications well up into supercomputing performance categories. Today, Sequent Computer, and its multiprocessing server product line is probably considered the leader in high transaction rate database-oriented processing, with HP, Sun, and Pyramid following closely.

The client environment is typically a smaller, but powerful PC that has enough power to run applications on top of multitasking, 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.

Performance from a Client/Server Environment

Users 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 conformance with the Transaction Processing Council's standards have shown that a sampling of available RDBMS servers are capable of running about 50 transactions per second (TPS) on 486-based PCs.

This number would not be unreasonable for a mainframe running IBM's DB2. Mid-size banks with over 100,000 transactions per day have run complete on-line teller systems against an SQL DBMS server running on a single processor 486 server. These cases were for a mixed DOS and NetWare environment with a DOS server running the database and a NetWare server handling file

PLAYERS IN THE DOWNSIZING OPEN SYSTEMS SERVER MARKET

GUPTA TECHNOLOGIES, INC.	SQLBase
IBM	OS/2EE DATABASE MANAGER
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
PROGRESS SOFTWARE	PROGRESS

and $40,000 per TPS. IMS-based MVS mainframe environments typically yield a cost of $50,000 to $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. This means 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 have cost in the past. Actual case studies confirm this type of important savings in finished, delivered systems.

Of course, there are many applications that are simply too large to contemplate running on even robust PCs. Client/server architectures allow you to design the application once, and then without change, port it to whichever server has the database processing power needed to manage the database. This allows application development on PC-style servers, and porting to the new generation of "super-servers," or minicomputers built to run open operating systems 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 minicomputers. Vendors such as Compaq, Pyramid, and Sequent are building parallel processing machines using CISC or RISC microprocessor units capable of reaching a sustained processing capability of hundreds of MIPS. Do not be surprised to see a combination of these new hardware systems with software from companies like Informix, Sybase, Microsoft, and Oracle delivering computing technologies comparable to IBM's largest machines, but at a tiny fraction of the cost.

As a first project, it is clearly more comfortable to use client/server computing for mostly read-only, 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 per second and are still largely relegated to mainframes only. However, as mentioned earlier, there is no shortage of serious transaction processing applications that have already been successfully implemented on top of client/server SQL environments.

In the future, expect multi-processor-based client/server architectures to regularly take on mainframe types of applications. It is very reasonable to envision products like Informix, Oracle and Sybase in combination with high-end super-servers from companies such as Sequent, Pyramid, HP, Sun, Compaq, IBM, or DEC. This high-end super-server hardware is typically built with parallel Intel 486, 586, and/or RISC chips from MIPS, DEC or Sun. By configuring a server with a multiprocessor design and an open operating system that supports it (e.g., UNIX, 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 servers 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 computing environments.

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 DBMSs. 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 o handle the SQL differences among different DBMS vendors. This is not always available today.

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 complete a job.

4. Supporting concurrency, in addition to deadlock protection, is very difficult.

5. Supporting full recovery with fail over reconstruction is expensive.

6. Performing a JOIN across different physical nodes is 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 database administrators is more difficult because, above and beyond their current functions, they need to understand the integrity, optimizer, communication, and data ownership 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 a database which is contained in one machine.

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 should take small steps while the approach is mastered before moving onto the more complex conversions or implementations. Many companies will find the client/server approach to be easier 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 per second, a (currently) large PC is capable of running an SQL DBMS and delivering services comparable to most of the IMS applications in existence today. The ability to create these applications with the ease associated with SQL databases and GUI screen painters is something that users only could have dreamed about in the mid-1980s. Prototyping approaches in building these 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.

Users should pay close attention to picking a software partner. The vendor(s) who provides a DBMS and development tools should be selected most carefully. In an era of open and replaceable hardware and operating systems, DBMS and development tools vendors will be a most important element of long term system strategies.