June 1992

LAN Technology

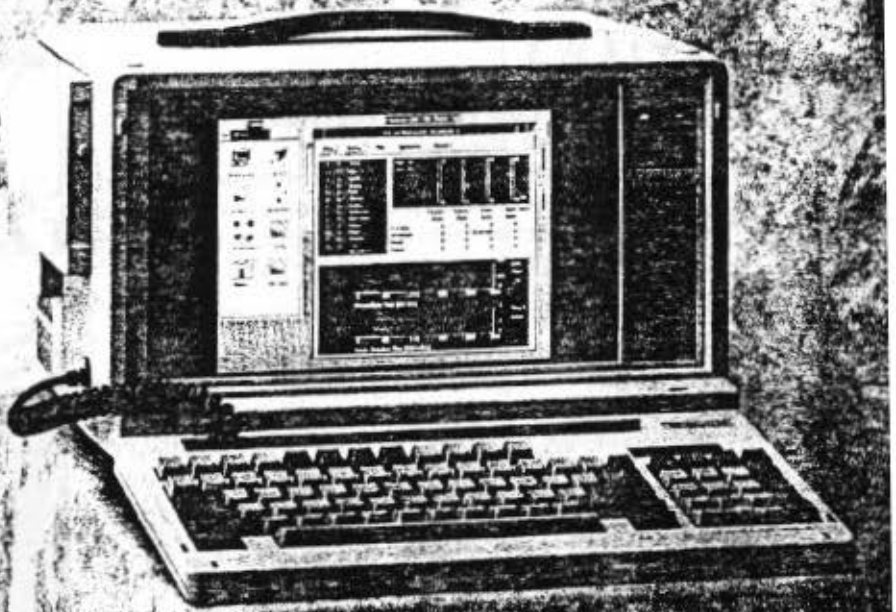
THE TECHNICAL RESOURCE FOR NETWORK SPECIALISTS / AN M&T PUBLICATION

Test an FDDI Backbone

Integrating AppleTalk Phase 2
Make Your Remote Macs Part of Your Enterprise LAN

Do Lunch and More
Calendaring Software Makes Meetings Easier

Gotcha! A Summer Virus
How to Find and Eradicate a LAN Disease



A Good Testbed Will Help You See The Light

$3.95 (4.95 in Canada)

LAN Technology
THE TECHNICAL RESOURCE FOR NETWORK SPECIALISTS

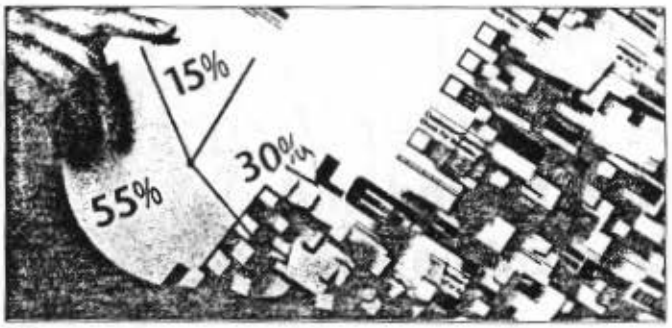
June 1992
Volume 8 Number 6

FEATURES

REVIEWS


59


41



32

COLUMNS

DEPARTMENTS

COVER PHOTO BY CARTER DOW

LAN Technology (ISSN 1042-4695) is published monthly by M&T Publishing, Inc., 411 Borel Avenue, San Mateo, CA 94402; (415) 358-9500. Second-class postage paid at San Mateo and at additional entry points. **POSTMASTER:** send address changes to LAN Technology, P.O. Box 56456, Boulder, CO 80322-6456. **Change of address:** Please send old label and new address to LAN Technology, P.O. Box 56456, Boulder, CO 80322-6456. **Customer Service:** For subscription orders and changes of address, call toll-free within the U.S.: (800) 456-1654. Foreign countries call: (303) 447-9330. **Subscription Rates:** $29.97 for one year, $56.97 for two years. Foreign orders must be prepaid in U.S. dollars drawn on a U.S. bank. Canada and Mexico: $40.00 per year. All other foreign orders: $65.00 per year. **Foreign Newsstand Distributor:** Worldwide Media Service Inc., 115 E. 23rd St., New York, NY 10010; 212-420-0588.

The Origin of the Server Species

Can file servers evolve into database servers ...or vice versa?

by David McGoveran

Charles Darwin theorized that diverse biological species emerge after eons of trial and error. The evolution of the server species happened much more quickly. There are file servers, database servers, FAX servers, and others emerging from network technology's primordial soup. But similar to Darwin, I believe that we must look at all network servers as a family if network managers are to install and manage the most effective server technology on their LANs.

ILLUSTRATION: CAROL EHRLICH

I reject the commonly held view that file servers are radically different from database servers. Similarly, today's marketing parlance that reserves the phrase "client-server" for database servers, or even restricts its use to LAN implementations, is imprecise and confusing (see sidebar, page 68). Rather, it's important to focus on the common features of file servers and database servers and develop an evolutionary view of servers in general and database servers specifically.

Historically, servers emerged because certain operating system functions, such as file services, could be efficiently placed on a separate platform. The first database server, for example, evolved from the simple idea of an intelligent disk drive at Britton Lee Corp. in 1979.

Both file and database servers belong to the overall genus, if you will, of client-server architecture. Many of the underlying concepts are the same, and the concerns they raise are similar. Thus, the file server user can perceive a database server as a functional extension of what he or she already knows, a natural evolution in the use of servers.

The classification of client-server architecture types has been described before (see references at the end of this article). Network managers need to understand the strengths and weaknesses of these different types in order to best deploy various servers throughout their LANs.

The Family of LAN

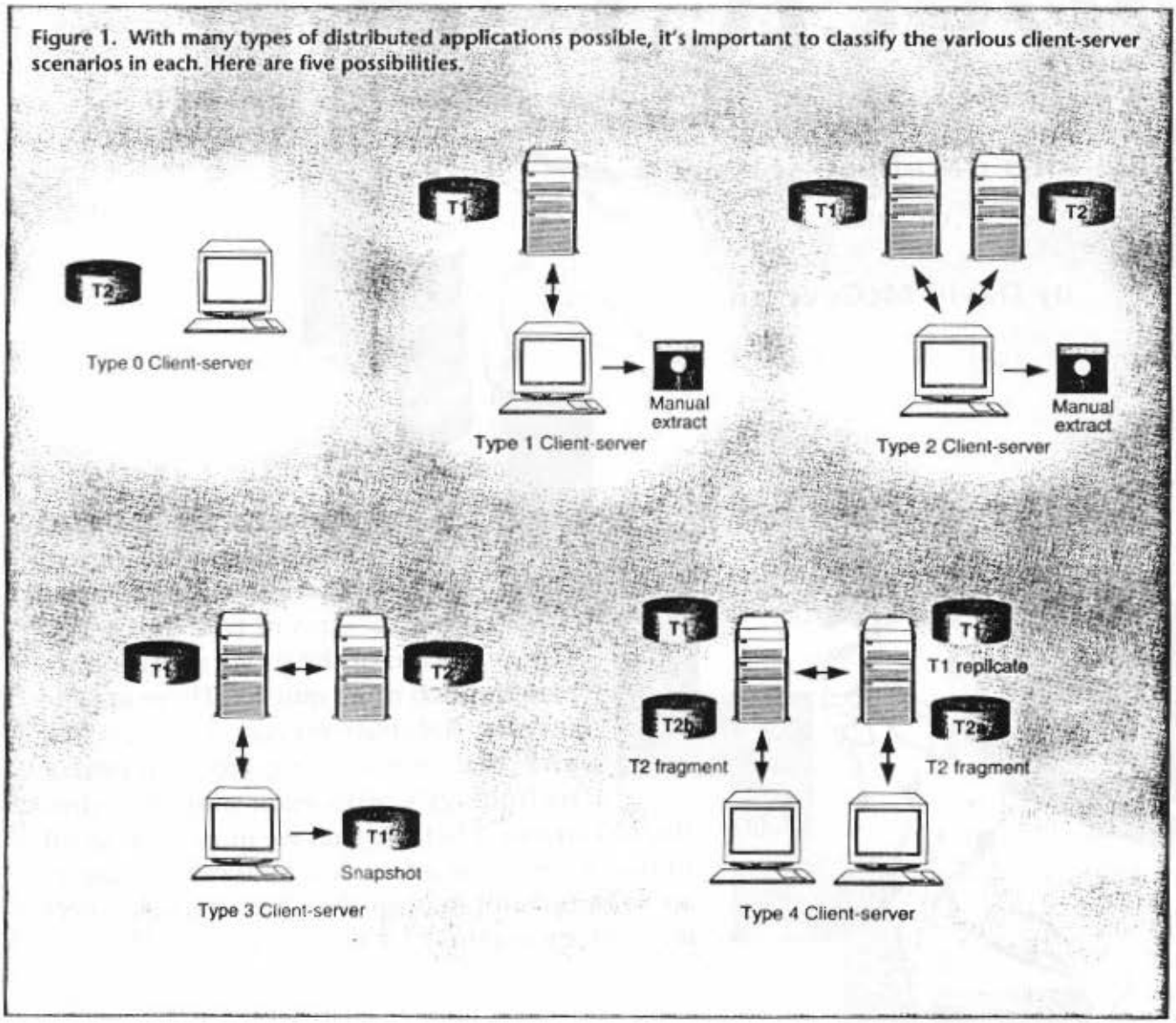
The first area of classification defines the degree of intelligence a server has over shared data. It defines the procedures a request must follow. It is loosely similar to a biological genus and, accordingly, contains "classes" that I dub as follows:

- Class 1: file servers
- Class 2: database servers
- Class 3: hybrid servers

Any of these server classes might accept either low-level or high-level requests. Low-level requests, such as opening an Ethernet socket in a network file service, can compromise one of the key benefits of client-server architectures. In particular, low-level requests are not as likely to be network, operating system, or hardware independent. If the client application is not isolated from the details of such requests (for example, through an independent API), portability along with both vertical and horizontal scaling of the application can be degraded.

One way to isolate applications is through the use of remote procedure calls (RPCs). With a database server, RPCs can take the form of stored pro-

Figure 1. With many types of distributed applications possible, it's important to classify the various client-server scenarios in each. Here are five possibilities.



cedures or database procedures. These are packages of SQL statements stored in the database in compiled form. They are invoked by name, accept parameters, and return results. Stored procedures are the preferred method of sending requests to a database server.

Classifying Client-Server Application Types

Before discussing each server class, it's important to understand how a client-server architecture can maintain data coherence regardless of how it is distributed on the network. This second area of classification I call server types. Server types differentiate how server classes can be used by different kinds of client-server-based distributed applications.

There are many types of distributed applications. Five scenarios are noted in Figure 1. For simplicity and ease of identification, call them:

- Type 0: local server
- Type 1: remote server
- Type 2: multiple remote servers
- Type 3: distributed transaction server
- Type 4: distributed database server

Although I will concentrate on servers at remote locations, it is important to keep in mind that the server need not be physically remote for any client-server application. A type 0 client-server application is one in which the client process and the server process reside on the same physical computer. In this configuration, the server can still conserve resources through shared processing to multiple applications. Communication is usually managed via a network, but shared memory can be used. If it is, any client-server optimization must have no effect on application logic; it should be one form of client-server communications that can be selected at the discretion of the system manager. Network communications can be used, but usually at the cost of slightly degraded performance due to the cost of communications overhead between client and server processes.

To a type 0 client-server application, all functions of a connection to a local data source are available. When multiple clients or servers are run on a single hardware platform, multiple processors may be used to improve performance. Nonetheless, type 0 servers do not work with a distributed processing architecture.

A type 1 client-server application can access data dynamically using remote request processing or remote transaction processing; hence the server is a remote server. More often, processing in this type of environment is done by accessing subsets of the corporate data that are then stored on the client-server system. These subsets can be created by regular manual extracts. The server can be either a single remote physical server or a local LAN server.

A person can use a type 1 application to access corporate data from a single remote server, but users are normally restricted to read-only access. The amount of dynamic access is kept low because of the potential performance hit on the remote system. Dynamic access is minimal, too, because the corporate data is frequently not presented the way a user wants it. He or she may, for example, want to see summary information or consolidated data, rather than detailed, raw data. This summary or consolidation would be created on the client by a manual extract from the server and then processed locally. The local data may become inconsistent if corporate appli-

cations are simultaneously maintaining the data on the server.

A type 2 client-server application extends type 1 functions by adding access to multiple remote locations through distributed transaction processing; hence the server portion has multiple server processes. The data accessed at any one location is usually independent of the data stored at another; for this reason, the client is the point at which distributed transactions are processed and managed. Because the remote locations are not connected by a network link, neither location can act as a coordinator in a two-phase commit protocol. (A two-phase commit protocol establishes the ground rules for each distributed transaction. First the coordinator, an independent software module, ensures all participants in the transaction are ready to commit or roll back, then it tells them to perform one or the other.) For this reason, type 2 processing only allows a client transaction to update data at one remote location. Even though a transaction can update data only at a single location, it is still possible for deadlocks to occur, and for this reason distributed concurrency controls, such as file locking, are required. As before, data distribution would normally be handled by manual extracts of data from remote servers.

A type 3 client-server application adds a two-phase commit protocol between locations and, therefore, lets client transactions update data stored at multiple remote locations. These extensions permit data stored at one remote location to be related to data stored at another location, thus providing the first elements of a distributed database capability. The server is a distributed transaction server. Unlike type 2, the server is the point at which distributed transactions are processed, although distributed transactions must still be managed or coordinated at the client. Companies are moving toward this type of client-server application. As this transition takes place, data distribution by manual extracts will be replaced by automated snapshots of the data.

As with type 1 and type 2, a type 3 client-server application can be used by end users to access remote corporate data. This type of architecture is also ideally suited for engineering and scientific applications that need to access and maintain distributed data. The functional and performance requirements of this type of application are not normally as stringent as those for corporate applications.

Type 4 client-server applications involve a server that is a true distributed database application; hence a file server will not suffice and so the server is a distributed database management system (DBMS) server. It permits distributed request processing in which the server both processes and manages distributed transactions. Here, the DBMS server offers both data fragmentation and replication, allowing faster access for read processing. Significant data modification operations can lower performance. Distributed database architecture requirements for this type of application are extended to include location transparency, global optimization, distributed integrity control, and distributed administration.

One of the key strengths of client-server architecture is the ability to use workstation end-user and development tools.

Strengths and Weaknesses

As with most technologies, there are strengths and weaknesses to using any client-server architecture, regardless of whether the data server is a file server or a database server.

The key strengths are:

- the savings in host processing power
- independent scalability of client and server platforms (including the ability to combine client and server software on one platform)
- code modularity through shared services (a server provides access to code that can be shared by multiple applications)
- the ability to use workstation end-user and development tools

With the move toward more sophisticated and user-friendly workstation tools, the last of the four points will become the dominant reason for using a client-server architecture.

Improperly used, the client-server architecture can cause some difficulties. These potential weaknesses are:

- performance overhead
- complex information systems management

Distributed or remote processing of any kind costs more in terms of performance than local processing. Network costs cannot be ignored. Even though a relational DBMS reduces the amount of requests and data that must be sent over the network, the speed of the network plays a major role in the transaction response times seen by the workstation user. Network protocol software must handle the information at either end of the communications link, and use of a network operating system is not unusual. Network operating systems use both host and client computer processing power.

On the host side, the processing power required by the network operating system (for example, VTAM in an IBM environment) may or may not be offset by the processing power saved by offloading the application processing to a client workstation. If network processing costs on the workstation are too great, performance is degraded for application processing. On the other hand, presentation logic for advanced graphical user interfaces (GUIs) can easily consume mainframe resources if workstations are not used. Clearly, network processing costs are significant in determining the performance of client-server applications. In evaluating the potential network load, the number of transactions per second and the volume of returned data must be taken into account, along with whether or not RPCs can be used.

The key to good performance is to improve the efficiency of the server request handling and to minimize the amount of data sent across the network. Server request handling efficiency can be improved by sending fewer requests of a higher level, and by tuning the server. The amount of result data sent across the network might be reduced in either direction by arranging for the server to send only data needed by an application or sending update data to the server only where necessary.

If requests to the server return more than one record, a block of result data can be sent to the client workstation,

stored in a buffer, and processed without further network interaction. This technique is referred to as *blocking*. A problem arises, however, if the client application wants to update the retrieved data. If the server has not locked the set of result rows, data integrity is jeopardized because other applications could have updated the data since it was retrieved. Possible techniques for solving this problem include the following:

- to use blocking and to lock the complete request result on the server until the client issues a commit
- to use blocking and to lock each row on both the client and the server as it is fetched by the client application
- not to use blocking, and instead to send the data across the network to the client, one row at a time, locking each row as it is fetched by the client application
- to use an optimistic concurrency control mechanism that checks for update collisions at commit time

Key issues are then:

- network overhead
- server request handling efficiency
- minimizing the number of requests
- minimizing the amount of data sent over the network

Management Issues

Although DBMSes using client-server architectures have been on the market for some time, most DBMS processing is still done using host-based applications. The data that these applications access is stored in central databases. The use of mini- and mainframe computers for this centralized application processing is becoming increasingly more expensive, especially when compared with the price/performance of microcomputer systems. For this reason, offloading minicomputers and mainframes to cheaper microcomputers or workstations, called downsizing, is of increasing interest. End-user computing, applications development, and corporate processing are all candidates for downsizing. Even if these types of processing are moved to a workstation, users still need to access host data. A client-server architecture is one way to resolve the issue of how applications and tools can get easy access to this data. In a centralized development and operational environment, all your programs, data, and data definitions are stored in one place. When you distribute application development, application processing, or data, the simplicity afforded by such centralized storage and control is lost.

For example, new problems include:

- management of multiple program libraries
- management of multiple data definitions
- server monitoring and performance tuning
- backup and recovery of distributed data
- network management

These problems have an impact on personnel and operations as well. Suffice it to say that these areas must be considered when implementing client-server applications.

Pros and Cons of File Servers

Whether a file server is distinct from a database server architecture is not really relevant, but you can waste a lot of time listening to the debate on the subject. I approach the issue with the following question: "What is the perceived benefit of a file server application that is not a perceived benefit of a database server application?"

The answer consists of four parts:

- out-of-the-box network support using network file services
- lower-cost "server" hardware required to go from single-user to multiuser shared file systems
- access to dBASE, Paradox, and other popular data files
- simplicity and familiarity of applications design and development

These issues are not specific to client-server technology. They have to do only with file server methods vs. database server methods, and then only at the level of file access methods and access control (such as lock management).

Figure 2a shows a single-user file server application. It is divided into two parts, the file client or application piece (FC) and the file server portion (FS). The file client consists of application-specific code and a file server engine similar to database management code. These may be cleanly layered in the code or may be tightly integrated or coupled.

In a file client/file server product, the file server portion can be moved off to another machine with a network intervening, as shown in Figure 2b. In this configuration, there is no change to the software as long as shared file access (or network file services) has been used from the beginning. For example, in a DOS environment, DOS file-access calls are then transparent whether the file is local or remote. File locking is handled by locking the entire file or by locking byte ranges. There is no differentiation between read locks vs. write locks at this level, so file server engines may implement a kind of semaphore system that lets them manage byte-range locks more intelligently.

When multiple users access shared files, each file server engine is replicated on the FC (see Figure 3). It detects contention at the file-lock level and responds by waiting until the resource is free. There can be no scheduling of multiple users, minimal concurrency control, no cache management, and no lock manager in the DBMS sense because there is no single engine to which all the required information is available. Unless each data file is locked for exclusive use and some client-side indexing technique is used, all data must be moved across the network before relational, filtering, sort, or merge

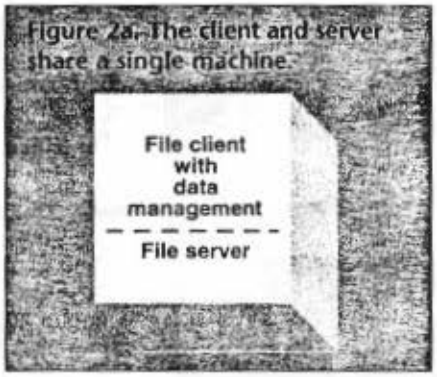


Figure 2a. The client and server share a single machine.

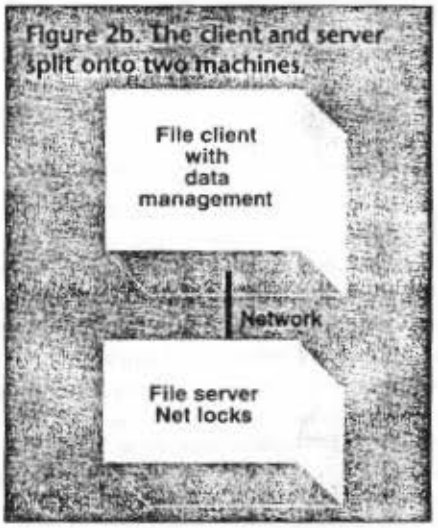


Figure 2b. The client and server split onto two machines.

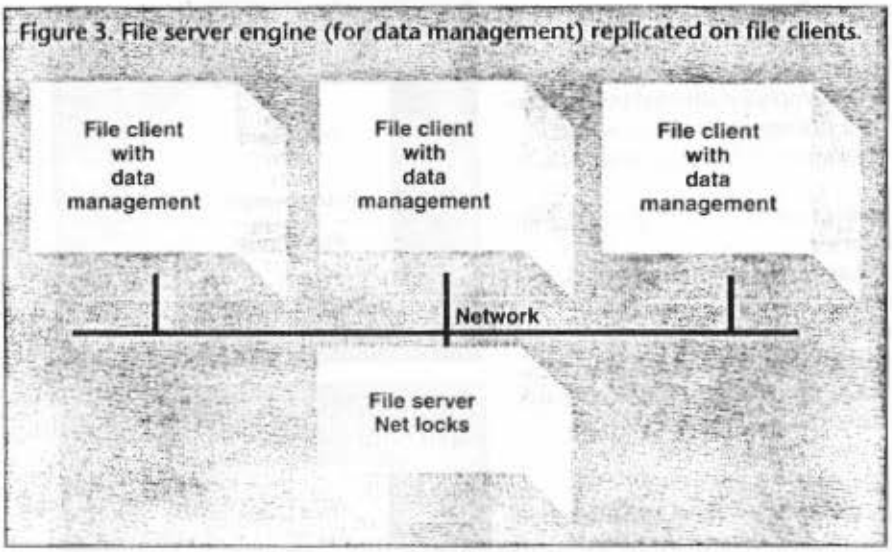


Figure 3. File server engine (for data management) replicated on file clients.

operations can be applied. This situation forces heavy network traffic.

File servers have minimal intelligence regarding request optimization, concurrency and lock management, and the scheduling of requests. Typically, each of these traits, especially request optimization, is handled as a result of the cleverness of the file client in anticipating the best way to process the data. In addition, proper organization of data in files and possibly even the storing of redundant data files can help minimize the amount of data an application must pass over the network. In the first case, only the data needed by a particular application request is stored in a single contiguous block, regardless of the existence of any related data. In the second case, redundant storage of data used by more than one user may help with concurrency problems.

Of course, these techniques may introduce integrity and synchronization problems that must be overcome by more careful application design and by additional processing. As to the scheduling of requests by a file server, this is usually handled in a simple FIFO manner based on the ability of the file client to wait for locked resources.

Figure 4a. Client/database on a single machine.

Figure 4b. Client/database split onto two machines.

The Pluses and Minuses of Database Servers

In the single-user environment, the client/database server looks similar to the single-user file server solution (Figure 4a). However, the internal division between database client and database server places data management functions with the server code rather than with application code. The database client consists solely of application-specific code. Now the server can have a lock manager, multiuser cache management, and scheduling. With the database engine on a separate machine, the configuration looks like Figure 4b. Now network traffic can be reduced. In a multiuser configuration, such as that depicted in Figure 5, redundant data management engine code can be eliminated. Keep in mind that there are no particular architectural limitations on the number of servers that a client can access, either with a database server or with a file server.

Database servers offer some unique opportunities for improving request handling. For example, client applications that communicate with a remote relational database server often use a database language such as SQL. The database server, after processing a SQL client request, sends back to the client only the data that satisfies the request. This is much more efficient in terms of network load than a file server architecture, where the complete file is often sent from the server to the client.

The set-level processing aspect of SQL also aids performance. A client application can, with a single SQL statement, retrieve or modify a set of database server records, rather than having to issue sep-

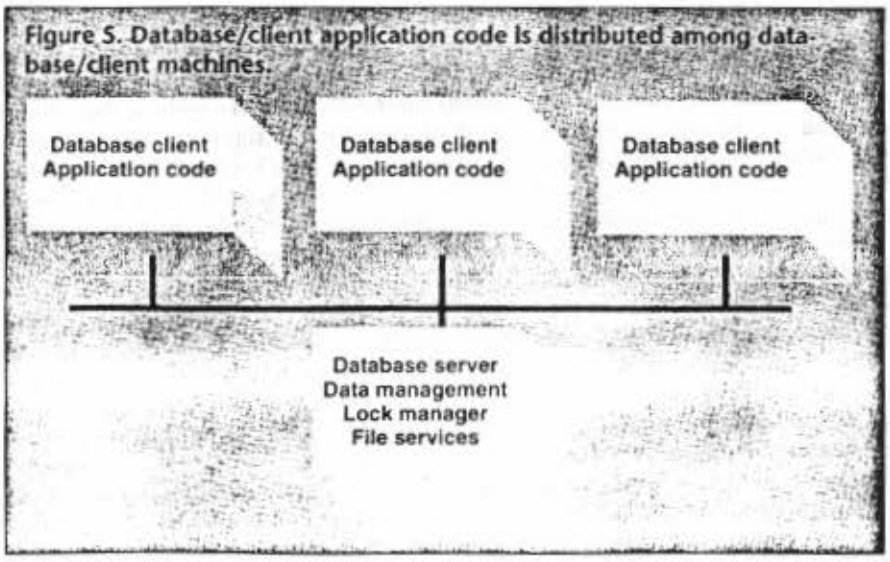


Figure 5. Database/client application code is distributed among database/client machines.

arate sequential requests for each desired record of each of the base tables, as in older database systems. Client-server SQL statements work most efficiently when doing data modification since data results need not be sent over the network. However, because SQL can create a results table that combines, filters, and transforms data from base tables, considerable savings in data communication are effected even for data retrieval. Only the specific rows and columns of data needed by the application need to be sent over the network.

With some DBMSes, the number of SQL statements being sent across the network by client applications can be reduced by storing a named group of related SQL statements and associated program logic in the database server as a stored procedure or stored program. The stored procedure can then be executed by a single request from the client application with the added advantage that the stored procedures can be shared by multiple client applications. The problem here is that there is no agreed standard for defining or invoking stored procedures (the ANSI SQL committee is considering such a standard). Furthermore, not all DBMSes support stored procedures and those that do vary in the permitted capabilities. For example, neither Rdb/VMS nor DB2 support them at the present time while Sybase and Ingres (and more recently, Informix and Oracle) do.

Even though SQL and stored procedures help reduce network traffic compared with non-relational access (which retrieves entire records from each file or table), the rows of a results

Terminology of the Genus Server

Like many other data processing technologies, client-server computing is littered with confusing terms, such as distributed processing, distributed database, and cooperative processing. The usual comparison between file server and client-server is not proper. Client-server is an architecture. Any system that separates application-specific code from service code has a client-server architecture. (When I use the term client-server, I mean the architecture, not a particular product or server technology.)

In the most commonly recognized form of client-server, the server is a relational database engine (a database server); applications residing on separate hardware and connected to the server via a LAN make requests to the server. The popularity of this particular class of client-server configuration often leads one to use client-server to mean this kind of product. Unfortunately, this confuses the technical differences between client-server and non-client-server with those between database client-server and non-database client-server. The server can be a file server as easily as it can be a database server.

Let's establish some definitions.

Server — It is up to the server to manage synchronization of services and communications once a request has been initiated. A server as discussed here is a logical process that provides services to requesting processes. A server can be complicated in its implementation; however, this complexity is hidden from other parts of the application. The logical process can be composed of a single physical process, or of numerous communicating and perhaps distributed processes. In fact, a server may depend on other servers of which it requests services. Ideally, the server has responsibility for managing all the requests it receives from other processes, including request-queue management, buffer management, execution of the service, results management, and notification of service completion. In general, it does not send results to the requester until the requesting process tells it to do so. The use of the term server in this article should not be confused with a piece of hardware, a special-purpose processor dedicated to running server software. There are many kinds of servers, including network, file, terminal, and database servers. A database server is responsible for processing database requests.

Client — Processes that request services from a server are called the clients of the server. There is no such thing as a client without a server. A process can only be called a client if it is a client of some server — it is not a client by virtue of its own structure. One characteristic that distinguishes a client from its server is that the client may initiate a communications transaction (as distinct from a database transaction) with the server but the server never initiates a communications transaction with the client. (This does not preclude "event notification" wherein the server notifies interested clients of some server-detectable event.) It is the task of the client to initiate communications, request specific services, acknowledge services-completion notification, and accept results from its server. Although the client can request either synchronous or asynchronous notification of service completion, it does not manage synchronization of services and communications. In a client-server architecture, many clients can share a single server. In this article, the term client should not be confused with hardware, such as processors connected to hardware servers.

Client-Server Communications — Communications between client and server in a particular installation can involve a variety of mechanisms: LAN, WAN, or operating system task-to-task communications services through such methods as mailboxes or shared memory. However, a client-server architecture is independent of these methods and the physical connection between them. A client-server architecture supports *transparent reconfiguration or even replacement of the client-server communications interface* so that applications and database processing need not be altered. In particular, note that the client and the server need not be on physically distinct processors or nodes. If the designer initially locates a client on the same physical machine as its server and uses shared memory for communications and then later places the two elements on geographically separated machines and connects them through a satellite, the architecture should accept the change transparently.

File Client/File Server — File client/file server is a class of client-server in which the server manages files. Typically, the file client makes requests to the file server to open and close files, to read and write records of some file, and to request a lock on a file or portions of a file. The file server manages the requests and handles network connections with multiple clients.

Database Client/Database Server — Database client/database server is a class of client-server in which the server manages a database; the server is a particular form of DBMS that can manage sessions with multiple clients. Typically, the database client makes requests to the server using the native database language, such as SQL.

Most client-server products accept hybrid architectures; a specific installation may use features or contain elements of both client-server and other architectures. Terms such as peer-to-peer, application-to-application, and cooperative processing all refer to systems in which the application is split into two or more parts, neither of which is strictly a server or a client except in satisfying a specific request. You should be aware of this difference, but explaining the details are beyond the scope of this article. The client-server definitions given above let us analyze a given installation and product and distinguish its client-server architectural features from its peer-to-peer architectural features. Nonetheless, mixed architectures make the job of discussing a product's architectural features difficult and confusing if we are restricted to this simple understanding and vocabulary.

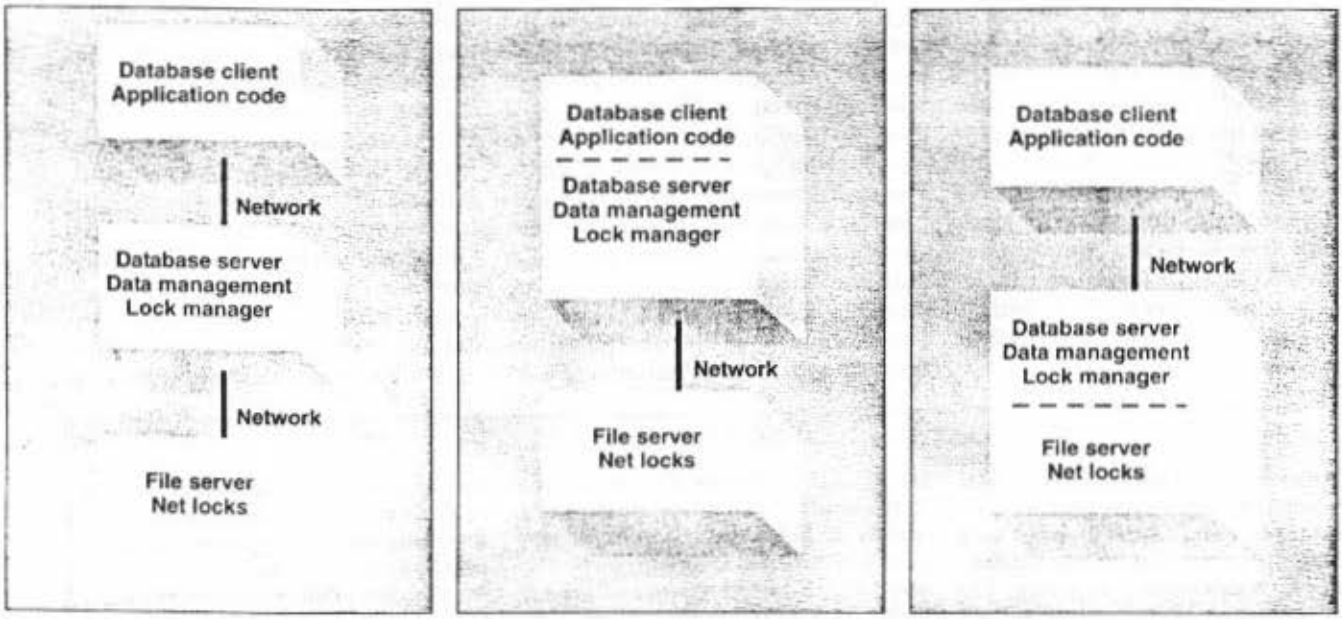


Figure 6a, b, and c. The three-part choice for an architecture in a hybrid client-server environment.

table still have to be sent over the network from the client to the server. Most relational applications use a cursor to fetch data, one row at a time, across the network. This constant fetching involves considerable communications overhead. That overhead can be significantly reduced if blocking is used, but the associated synchronization problem for updates must be solved.

Implementing the Hybrids

The class 3 client-server architecture is the hybrid group. It doesn't matter whether the configuration is built on file or database servers. You implement one, the other, or both depending on your need to share code and data on the network. What is needed for a hybrid client-server is a three-part architecture as shown in Figure 6a: application-code-only client, database-management-only server, and a file server for file access.

Notice that this architecture can have either a file server or a database server configuration (Figures 6b and 6c). In addition, it has a configuration that is both.

The requirements for implementing the hybrid architecture are fairly straightforward. In particular, it requires that the file-access routines are in separate modules and that no other module depends explicitly on file locations. Second, it requires that the lock manager is similarly isolated and that no other module depends explicitly on direct access to physical locks. Given these constraints, it is easy to modify the DOS file-access calls to support network files. Similarly, the lock manager's critical data structures can be separate from the network file server. This last requirement should be a configuration option, since there is an associated performance penalty (file server users do not seem to object to these added performance costs); it should only be used in a configuration that requires multiuser remote file access.

Hybrids offer a natural migration path for users, starting with the single-user, single-machine configuration, then moving to a multiuser file server, and then to a multiuser database client-server. The middle step could be skipped if a file server is not a desirable option for a particular installation. At no time do applications have to be rewritten; only the installation parameters need be changed. Access to network files is available out of the box. The cost in going from single user to multiuser can be the same as the traditional file server path.

Finally, it is also possible to add access to dBASE or other files in one of three ways. The easiest approach is import and export, a capability that is often provided by DBMSes. The second approach is file conversion that simply externalizes the import/export function into a utility. The last approach is direct access without conversion as performed by the R:Base file server engine and Coromandel's Integra database engine. Direct access without conversion requires writing an access method that makes the foreign files look like a native file structure. The price for this level of transparency is a possible performance hit and the fact that relational DBMS access to data can be subverted.

Darwin's ideas about natural selection help explain the diversity in nature. I think that it's natural for network managers to have a selection in defining their file and database server resources on a network. Why should a client-server product force you to decide between a pure file server or pure database when both can be offered? You shouldn't have to. Hybrids work well and offer tremendous flexibility.

I recommend this approach for any vendor developing so-called client-server or file server products. An immediate benefit to the vendor is the ability to be a player in a larger market because more customers could add their products to their networks. Users get more diverse and powerful systems to choose from. As in nature, the more choices your network has, the fitter it will be. And with fitness comes survival. ■

Much of this article is based on a chapter from a forthcoming book by the author, An Advanced Guide to Client-Server Applications.

References

D. McGoveran and C.J. White. "Clarifying Client-Server," *DBMS*, Volume 3, Number 12, November, 1990

C.J. White, "Using Distributed Database: Application Types," *InfoDB*, Volume 4, Number 1, Spring 1989