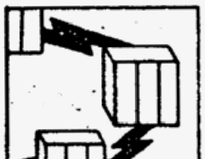
Jim Gray and Sandy Metz, Tandem Computers Inc., Cupertino, Calif.

Solving the problems of distributed databases

True distributed databases—where dispersed records look to users as one unit, without centralized control—are now appearing. Here's how it is done.



Despite the increasing number of computers within many companies today, the full value of all this hardware—the potential return on investment—is often not realized because the diverse computing resources cannot share information. However, recent advances in the area of distributed databases (DDBs) are now making it possible for all corporate data to be accessible through a single resource.

Such schemes permit companies that have even widely dispersed data repositories to retain the advantages of locally controlled data. A true distributed database represents a decentralized scheme for data management wherein files are spread through a collection of autonomous nodes that communicate with one another via a common language. The purpose of such a decentralized database is to make all the data that is available to the corporation as a whole also conveniently available to individual users. This data availability can, for example, facilitate the local management of day-to-day tasks while also providing a basis at the corporate level for planning future strategies.

Though the nodes in a distributed database can exist in one room or building, these nodes are usually geographically separated. The DDB can therefore link a worldwide corporation into a single operating entity, with vital information available in a timely fashion wherever it is needed (Fig. 1). With a properly implemented distributed database, critical data can be stored, updated, and retrieved, independent of the location of either the data or the user.

The term "distributed" database has been used to describe some data management schemes that really offer only a subset of true distributed database capabilities. One example is a centralized database that is accessible from remote nodes. This can more precisely be called a shared database, which provides, in reality, only distributed access to centralized data. Another scheme features individual databases residing on computers that are linked in a network. While these are, in a literal sense, "distributed" databases, the data within each is still inherently centralized.

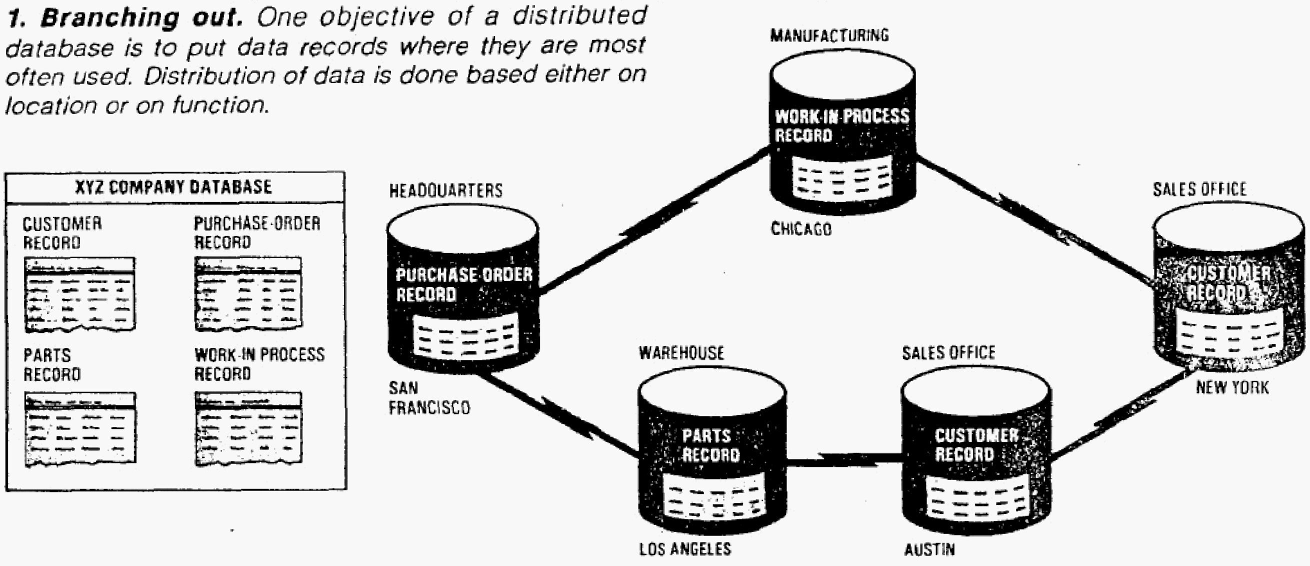
There are several technical considerations that make a truly distributed data management scheme attractive—the main one being the sheer size of many databases today. Linking diverse data files into a single resource often provides additional capacity that is increasingly hard to find with the single, centralized approach. A decentralized data management "network" could consist of literally hundreds of individual processors located around the world, with the data in each available to every node.

Only the data that is used on a daily basis need be kept at a local node; other useful information in the database is accessible remotely. In this way, data availability can be guaranteed by placing critical data at the local node. Naturally, placing data next to its most frequent users speeds response time in retrieving this data.

The autonomy of nodes in a distributed database allows each organizational entity to manage its information in its own way. And since each node is independent, and the data location transparent to the user community, the database configuration is modular and, therefore, flexible. Network nodes can be added, deleted, and rearranged without significantly affecting data access and usage.

From a management standpoint, linking data into a single resource provides a way to track the status of the corporation as a whole with convenient access to network-wide data. At the same time, control of local data resources can be kept at the divisional or depart-

***1. Branching out.** One objective of a distributed database is to put data records where they are most often used. Distribution of data is done based either on location or on function.*



mental level. The existence of a corporate-wide database need not impact the efficiency of local data management and retrieval activity.

Another big plus to management is the flexibility provided by a decentralized data management scheme. The data distribution can be designed to reflect the changing needs of a business: When information needs change, the database can too.

Distribution design

Several non-issues with a centralized database, such as how the data will be distributed, become critically important in a decentralized environment. There are two main approaches to distributing data: Decentralize by function, or decentralize by location.

The selection of the best decentralization method is based on the particular application, or the way data will be used. If the data will typically be accessed repeatedly by the same users, then decentralization by function could be the more appropriate. Examples of this would be putting manufacturing materials lists at the appropriate manufacturing plants and customer information at sales locations.

Partitioning customer information on a node-per-region basis is an example of decentralizing by location. This method might be used for data pertaining specifically to a sales region or other geographically based entity within the corporation.

Another key issue that has to be resolved in evaluating the feasibility of a distributed database is the degree of decentralization. For example, function and maintenance of individual nodes can be decentralized while the operation and control of the collective database and network remains centralized. Or it may be preferable, depending on the situation, to further decentralize operation and control while keeping the design of the database and network architecture centralized. At the extreme, it may be desirable to decentralize everything, except the "global protocol" architecture.

An analogous example of maximum decentralization is the international telephone network. Each telephone company independently implements the common protocols of the international phone network (such as for dialing and billing), and the only centralized function is the architecture of these protocols. Within each company, design and architecture are typically centralized, while operation and control are delegated to the operating regions. These regions, in turn, delegate most operation and maintenance to the individual exchanges, which operate and maintain their own local hardware.

Searches

A major challenge in designing and managing a distributed database results from the inherent lack of centralized knowledge of the entire database. It is difficult and often undesirable to maintain information concerning the entire database in any one place, but this requirement seems inevitable in order to manage requests such as, "Where is file A?"

One solution to this dilemma involves the concepts of global, local, and semiglobal data. Global data is information that is common to and shared by all sites. Examples of global data are an item master file of parts that comprise a company's parts catalog and a bill-of-materials file that describes a product's structure.

Local data is information that is uniquely important to the individual site using it, although it is accessible to all sites. Examples of local data are items in stock and work in process. Local data retains the same format as corresponding data has at other sites.

Semiglobal data is used in internodal—and often intersite—transactions. This might be the case for, say, an interplant materials transfer. In this case, a request by one site for materials from another is placed, processed, and monitored. The process requires that all data and status information pertaining to the request be resident at both sites. But since this information is of no use to any third party, it is duplicated only at the two nodes that use it.

Information is made available to the entire network by partitioning or replicating the data files. Partitioning a data file means splitting it into records and then distributing the records so that each record resides at exactly one network node (Fig. 2A). Replication means duplicating data records at more than one node (Fig. 2B). Local data can be partitioned, but global data must be replicated.

Data is partitioned to put it close to the sites that use it. An example might be storing bank account data at the home branch of the bank customer. This has the effect of reducing message traffic and message delay, and of distributing work. In the case of an airlines reservation network, data is partitioned by corporation. Most transactions submitted by one airline deal only with that airline and therefore run on a single node. Transactions that deal with other airlines are routed to other airlines' nodes, as appropriate.

Replication also serves the purpose of bringing data closer to the user, and has long been used to improve data availability. If one copy of a file is lost, for whatever reason, another can be accessed at a remote node. Global data is replicated at all sites. In a geographically distributed database network, replication also provides the benefit of improving response time by eliminating long-haul message delays.

Updating

Partitioned data is most efficient when the data must be kept current, which generally means that it is updated frequently. The single copy of each data item makes updating an efficient process. However, nonlocal "read" operations are more expensive, making partitioning less efficient for data that is widely used but updated infrequently. In the Tandem scheme, a database record manager allows files to be partitioned among network nodes based on single field values within files, such as "part number" or "customer name."

Replicated data is most efficient when multiple reads of the data are expected, but updates are not as frequent. The data is duplicated at nodes where high-volume reads are expected, producing high availability and good response time. When replicated data must be updated, however, an update to a record at one node should cause an identical update at all other nodes where that record resides. If any one replica is unavailable, there could be problems.

A variety of schemes can be employed for updating replicated data, even though the copy of the record may be temporarily unavailable at one or more of the nodes. One technique requires that a majority of the replicas be read and updated as part of each transaction, though the definition of "majority" varies with the application. This scheme has the advantage of tolerating some nodal unavailability, but it is not practical for either very small or very large networks.

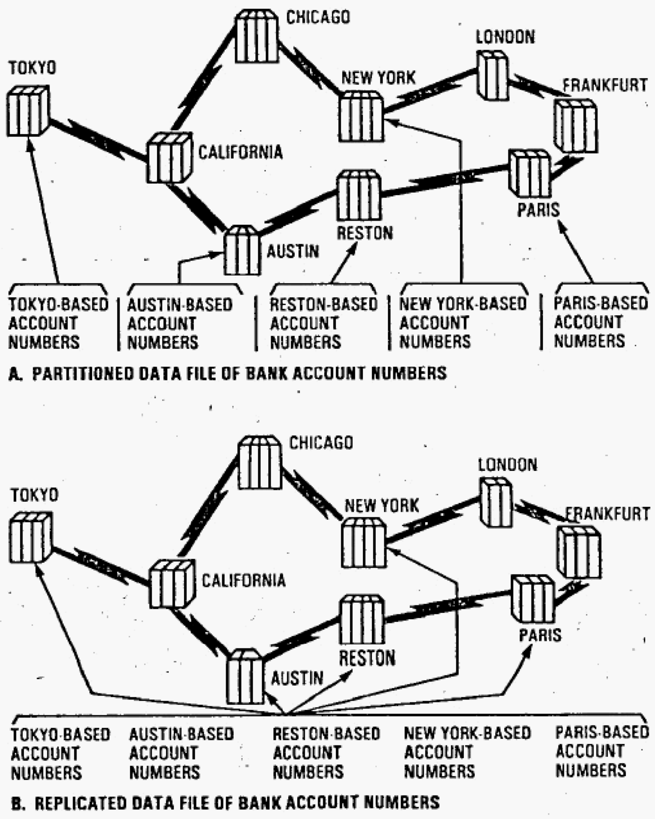
In a very small network of, say, two nodes, having either node unavailable prevents an update of a majority of the nodes. In larger networks, delays in completing the update transaction are proportional to network size: As the network grows, transactions will take longer to complete. One example of a file manager that uses a majority-update scheme is an experimental database network built at Xerox Research (see references for additional information).

Another method for updating replicas is the "as soon as possible" (ASAP) method. This technique involves designating one replica, the "master copy," on either a record-type or case-by-case basis, which ensures that the file at its node is updated. The updates are then asynchronously sent to the other replicas. This approach sacrifices consistency for availability and response time. Tandem's internal distributed database application, called Empact, is one that uses ASAP updates for frequently used data, and consistent updates for critical data.

A different method involves a time-based technique, in which there is a master copy of the data record, and its replicas (or slaves) are "snapshots" of the master as of a specific time. The slave copies are periodically updated, and each replica is "time-stamped" to indicate its degree of currency. This technique is appropriate for files that change very slowly and for which currency is not critical to business operations. IBM's experimental "R" System provides this time-stamping of replicas.

When retrieving the time-stamped replicas, the degree of currency can be specified in the query. It may

2. Replication. *Local data files may be partitioned (A) at the same site. Global files, on the other hand, are replicated in each network node (B).*


CHICAGO
LONDON
TOKYO
NEW YORK
FRANKFURT
CALIFORNIA
PARIS
RESTON
AUSTIN
TOKYO-BASED ACCOUNT NUMBERS
AUSTIN-BASED ACCOUNT NUMBERS
RESTON-BASED ACCOUNT NUMBERS
NEW YORK-BASED ACCOUNT NUMBERS
PARIS-BASED ACCOUNT NUMBERS
A. PARTITIONED DATA FILE OF BANK ACCOUNT NUMBERS
CHICAGO
LONDON
TOKYO
NEW YORK
FRANKFURT
CALIFORNIA
PARIS
RESTON
AUSTIN
TOKYO-BASED ACCOUNT NUMBERS
AUSTIN-BASED ACCOUNT NUMBERS
RESTON-BASED ACCOUNT NUMBERS
NEW YORK-BASED ACCOUNT NUMBERS
PARIS-BASED ACCOUNT NUMBERS
B. REPLICATED DATA FILE OF BANK ACCOUNT NUMBERS

not always be necessary to read the most current copy, so some time and communications costs might be saved by reading a copy that is physically closer, but with an older time-stamp.

Relational

With data distributed all about a network, the retrieval method must be convenient and fairly simple to the user. This means that the database manager must keep track of all data locations in a manner that is transparent to the user. This requirement, combined with the flexibility needed to move data from node to node as information requirements change, makes a relational model almost a necessity in a distributed database environment.

A relational database stores data in two-dimensional tables of rows and columns containing related information (Fig. 3). Information is entered into the database by creating the tables and filling them with pertinent data. Expanding the database is a matter of adding new tables or adding new entries to existing tables.

Unlike hierarchical and network databases, the structure of a relational database is not determined and fixed when the database is defined. Data items are logically linked by the data management software on an as-needed basis, so data items are not dependent on other items (Fig. 4).

Connections between records are based on "soft pointers" (called keys), rather than "hard pointers," such as record addresses. This distinction allows the data at a node in a relational database to be reorganized without affecting other nodes. The relational data structure, it can be said, is dynamic and flexible, which makes it particularly suitable for a distributed environment.

Maintaining data integrity

A clear concept of a transaction is essential in coordinating multiple updates to distributed data. The multiple nodes and multiple copies of data items can mean distributed chaos if transactions are not carefully implemented and monitored. A transaction is an operation in which application procedures, such as banking operations, are mapped into transformations (by executing programs) that invoke database actions. These include: Read the customer, account, and teller records; write the account, teller record, and a memorandum record; and send response messages to a terminal. The result of this process should be that the database is moved from one consistent state to another.

The key properties of a transaction are:

- Consistency—the transaction is a consistent transformation of the database state (for automated teller or banking transactions, that money is neither created nor destroyed)
- Atomicity (transactions are "atomic")—either all the actions invoked by the transaction occur, or else the entire transaction is nullified (in the banking case, that no account is left in a partially updated state)
- Durability—once a transaction is completed, its effects cannot be nullified without running a compensating transaction (funds removed from an account would have to be redeposited to be accessed again).

All of these criteria and requirements must be upheld uniformly across the network in order for a distributed database to work. Database management packages that consider a single database action to be a transaction, therefore, are unsuitable for a distributed environment.

There are several techniques available for maintaining consistency, atomicity, and durability in a centralized environment, including concurrency control and transaction backout (reversing the effect of a partially completed transaction). These techniques can also be applied in the distributed environment, but their man-

3. Relational. *A relational database differs from the hierarchical database in that common elements in the file permit records to be logically connected.*

CUSTOMER

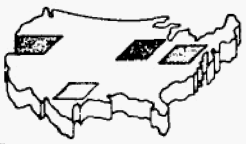
CUSTOMER NO.	NAME	CITY	CREDIT LIMIT
101	ACME	NY	500
102	SMITH	ATLANTA	1,000
[illegible]	JONES	SF	1,000

ORDER DETAILS

ORDER NO.	STOCK NO.	QTY.
123	A43	10
123	G04	100
124	E96	50

PURCHASE ORDERS

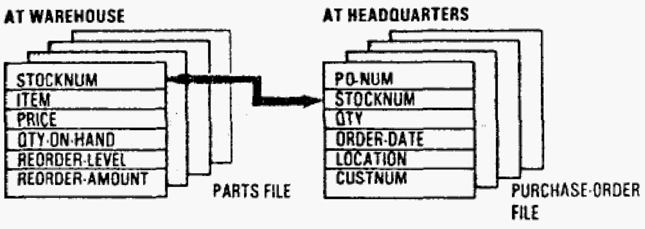
P.O. NO.	STOCK NO.	QTY.	ORDER DATE	CUST. NO.
27560	A43	10	1/10/82	102
13892	A76	43	1/30/82	101
42610	C61	2	2/2/82	216



PARTS

STOCK NO.	ITEM	PRICE	QTY. ON HAND	QTY. ON ORDER
A43	WIDGET	2.98	200	10
A76	TRIBBLE	4.39	40	500
C61	PLINTH	1.53	800	0

4. Linking up. *In this DDB transaction, a warehouse parts file is linked with a headquarters purchase order file to determine how many items are in a particular order.*



QUERY:

```
OPEN PARTS, PURCHASE ORDER
LINK PARTS TO PURCHASE ORDER VIA STOCKNUM
LIST BY STOCKNUM
    ITEM,
    QTY-ON-ORDER,
    ORDER-DATE,
    WHERE LOCATION = "SAN FRANCISCO";
```

RESULTS:

STOCKNUM	ITEM	QTY-ON-ORDER	ORDER-DATE
A76	TRIBBLE	500	05-19-82
G04	BLIVET	500	05-19-82
H73	WHATSIT	1,000	04-28-82
K88	BLURB	400	03-06-82
M36	PLAZZER	50	04-15-82
T05	KRUPUS	535	02-17-82

agement on a network-wide scale becomes much more complex due to the added communications considerations.

To ensure database integrity in a distributed transaction, all messages between nodes must arrive safely, and the sending node must be made aware that each message has in fact arrived. Both requirements can be met by using a "two-phase commit" protocol.

"Committed" transactions

A two-phase commit protocol uses a commit coordinator program to centralize the decision to commit or abort a transaction. The commit coordinator has a communications path to all the participants of each transaction. These participants, it should be noted, can be processes, autonomous components within a process, or both.

The commit coordinator asks all the participants to enter a "prepare" state, from which each participant can either commit or abort its part of the transaction. Once all participants are in the prepare state, each will transmit a message indicating this to the commit coordinator, which in turn can send a commit or abort message to all the participants (Fig. 5).

Once the commit coordinator sends the commit message, it waits for an acknowledgment from each participant before terminating the transaction. Use of this two-phase commit protocol helps ensure the integrity of a distributed transaction.

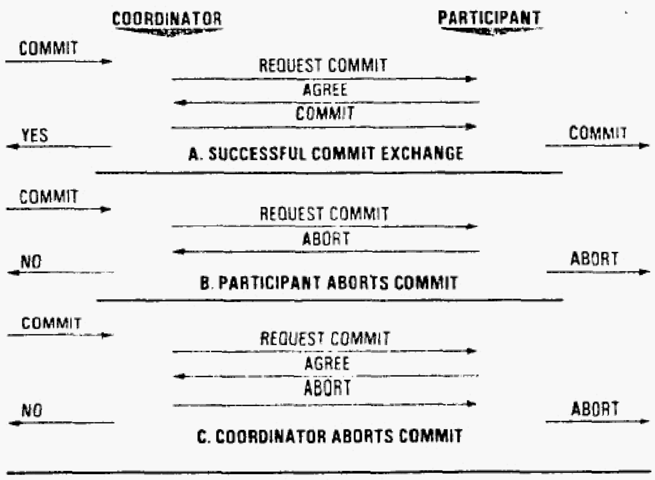
Distributed administration

Management of a worldwide database must be both distributed and centralized. Certain aspects of the database are common to the entire network and therefore must be designed and controlled by a central organization. A prime example of this is the global record format.

Local database functions can be controlled at the local node to provide site autonomy, which is one of the basic goals of a distributed database. An example of one such local function is a report format.

A hierarchy of control can therefore be imposed, with network-wide functions being managed by a central organization and control of other database activities being distributed in a hierarchical fashion. The key requirement, however, is that each level use the protocol of the global architecture for all its inputs and outputs. Each organizational level has an administrator, who publishes and controls the protocols of his component of the database network. And while a great degree of autonomy can be exercised, the structure of levels and control at this level should parallel the structure of the overall organization.

5. Commitments. *A dialog between the commit coordinator and a participant (A) ensures that transactions will be completed. The commit coordinator has a path to all participants, any of which may abort (B and C).*



DDB selection

The choice of a distributed database management system is naturally dependent on the application requirements. However, care should also be taken to implement sufficient flexibility into whatever database network is constructed, to account for rapidly changing application requirements.

Requirements of a true distributed database include the ability to distribute data files between at least two computer nodes: to provide location transparency between data and users; to retain data file relationships (even when the files are located at separate network nodes); and to ensure transaction integrity in the distributed environment. The two commercially available distributed transaction management systems that most closely meet these requirements are IBM's CICS/ISC and Tandem Computers' Encompass. The ISC feature of IBM's CICS provides for distributed transactions and the ability to access remote files, but it does not transparently handle data partitioning or replication.

Data partitioning requires direct action by an operator with IBM's CICS/ISC, while this is done automatically—transparently to the operator—with Encompass. Manual intervention is also required with the IBM product for data replication, but Encompass requires manual intervention only for files resident on a remote node.

Another selection criterion is flexibility, since one of the purposes of a distributed database is to allow for the changing information needs of a corporation. The ability to add nodes, delete nodes, and reconfigure the distribution of data without changing application programs is a requirement.

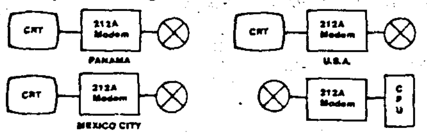
Inherent in all of these requirements are a reliable data communications and networking capability, and the use of a relational database model. Without this base on which to build, no distributed data management network can be successful.

Beyond these basic requirements are some features that will enhance the usefulness of the database throughout a lifetime of changing requirements. One way of achieving this goal is through the use of highly reliable hardware and network software. Even though the database must be designed so that a failure at one node cannot prevent access to critical data, the distributed network will be much more efficient if extraneous hardware and software failures can be kept to a

THE DATA DETECTIVE # 8

The Case of the Chauvinistic 212A

A large bank in New York City installed a 212A on one port of a computer to allow remote branches to access a new specialized data-base. The first branches to use the new system were in the eastern part of the United States. Their calls were answered properly and the system functioned quite normally. Since the system worked so well, it was decided to allow overseas users to access it as well. Therefore, identical 212 modems were sent to and installed in Panama, Mexico and the Caribbean, but when Panama or Mexico called New York the New York modem would answer the call, send answer tone and then, after a few seconds, hang up for no apparent reason. Strangely enough, calls from Puerto Rico were accepted by the modem. Clearly, this 212 had strong nationalist tendencies and could understand Spanish. Changing modems in Panama and Mexico had no effect, so clearly, the problem was in New York but why would or how could a simple modem discriminate between calls based on point of origin?



A closer examination of the sequence of events showed that foreign modems heard answer tone but never received carrier from N.Y. after that. Therefore, after a few seconds, they ceased sending originate carrier which, in turn, caused a loss of carrier disconnect in New York. For some reason, answer carrier was not reaching the foreign locations even though answer tone did. Voila! On the long foreign calls there was an echo suppressor on the line that was not disabled, so that when originate carrier was on the line, answer carrier going in the other direction was attenuated. But, answer tone is supposed to disable echo suppressors and prevent this sort of thing from happening. A quick measurement of the modem's answer tone frequency showed it to be 2425 Hz instead of 2225 Hz (U.S. Domestic), or 2100 Hz (CCITT V.22), too high to disable echo suppressors. Changing the modem in New York ended the chauvinistic proclivities of this banking system.

NEXT MONTH:
The Terminal With A Magnetic Personality!

Contact Glasgal for all your data communications service and hardware needs. Unlike company reps or direct salesmen, Glasgal, as an authorized distributor, can offer expert total network configuration counsel, multibrand system pricing, and same day shipment of orders or repairs for over 35 carefully selected brands of modems, multiplexers, switches, dial back-up units, data compactors, patches, cables, protocol converters, DSU's, call calling units, cabinets, testers, line drivers, port selectors, local area networks, etc., from AJ, ARK, Astrocom, Avanti, Cermetek, Coastcom, Concord Data, CRC, Dataprobe, Datec, Develcon, Electrodata, Electronic Enclosures, GDC, Hadax, Halcyon, Icot, IMM, Micom, Microcom, Microframe, Network Products, Paradyne, Prentice, Racal-Vadic, Remark, Rixon, Symplex, Tellabs, Ven-Tel, VIR, Western Datacom, etc.

GLASGAL
COMMUNICATIONS, INC.

Free 132 Page Illustrated Catalog

GLASGAL
COMMUNICATIONS, INC.
207 Washington Street, Northvale, NJ 07647
(201) 768-8082

minimum. The ideal, of course, is to maintain data availability in the face of component failures or temporary inaccessibility of some network nodes.

Evolution

The linking of highly reliable computers into a single distributed database is not easy, but progress in this area and the availability of proven products is making this once blue-sky objective possible to achieve. That computer networks will move in this direction is inevitable.

Information management schemes, now computer-based, are replacing traditionally paper-based ones. But these earlier operations were not totally inefficient—the paper was invariably located at the point where it was most often used. The move to centralized data management procedures changed all that, though it came about more from a need to optimize expensive computing resources in the earlier days of computer technology than from the desire to centralize information resources.

With the cost of hardware rapidly decreasing and the reliability of data communications steadily increasing, the time has come to return to an information management operation that puts the data back where it is needed, as long as it can be done without sacrificing the advantages of a centralized database. Distributed databases are therefore the logical continuation in the evolution of computer usage for information management. And this evolution has been considerable: from compact data storage, to early database management systems, to the on-line access of centralized data, to remote data processing, and now, finally, to the distributed database management system, which promises to provide accurate and consistent data to all users, acceptable response time, and availability—even through otherwise catastrophic communications and hardware failures. ■

■

Additional reading/references:

Gifford, D.K., "Weighted voting for replicated data," *ACM Operating Systems Review,* 13.5, December 1979, pp. 150-162.
"Information storage in a decentralized computer system," Xerox Publication CLS-81-8, Palo Alto, Calif., 1981, pp. 21-75.
Houston, George B., "Tightening up software on a distributed database," *Data Communications,* April 1983, p. 133.