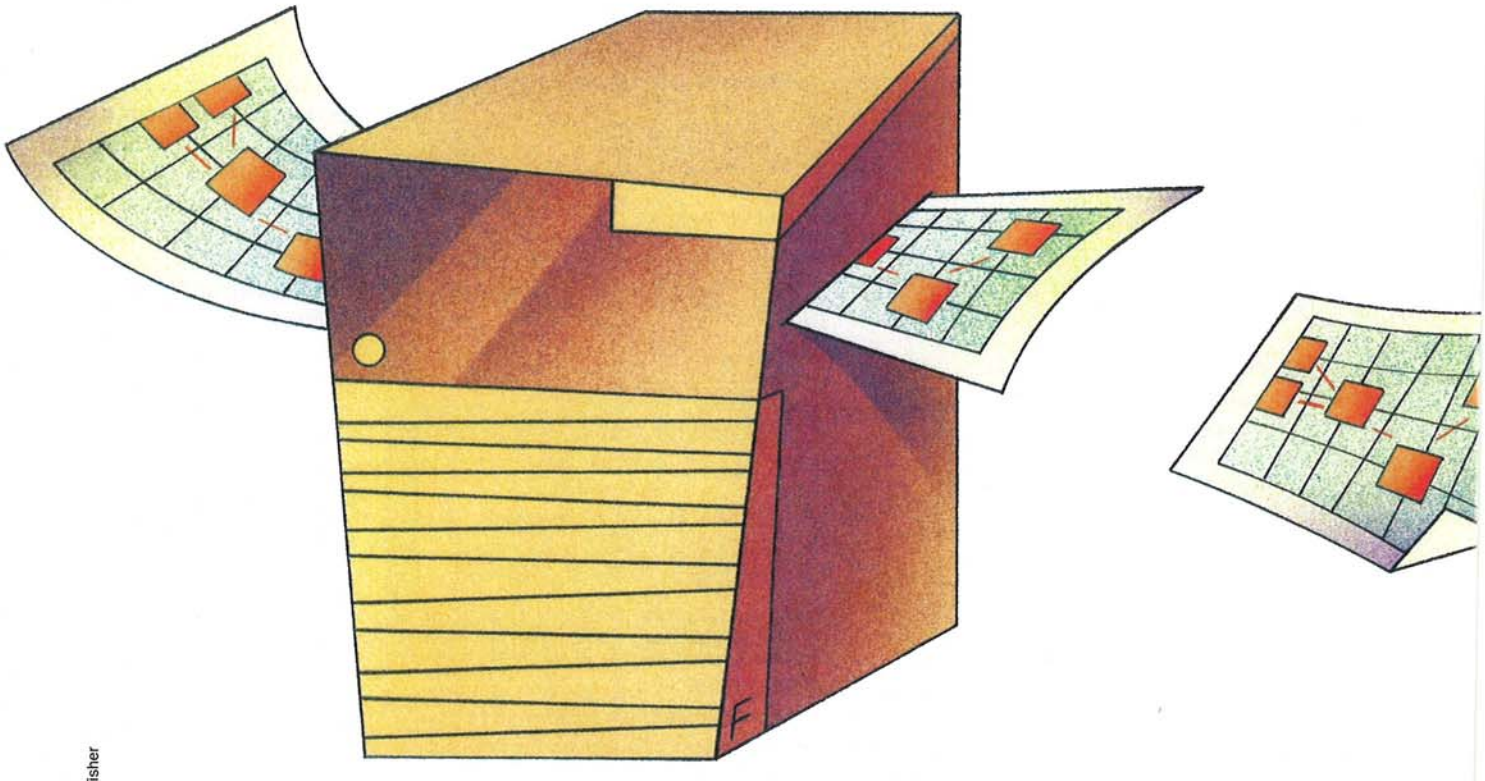
BY GEORGE SCHUSSEL

DATABASE REPLICATION: *WATCH THE DATA FLY*

Individual workgroups can now have their own replicated databases. For IT, this means never having to say "sorry" for network delays



Mark Fisher

This is the first of a two-part series on database replication. This article focuses on the overall importance and benefits of replication and provides an introduction to the two forms of replication services. Next month, we will examine in detail the differences between these two forms of replication: data warehousing and high-end transaction-processing class replication servers.

The business paradigm of the nineties seeks to place more decision-making authority in the hands of those closest to customers. To do this, corporations are reorganizing along their lines of business rather than aligning along the functions of business. For IT, which many corporations view as one of the critical success factors in business-process reengineering, this has meant a series of dramatic fundamental changes.

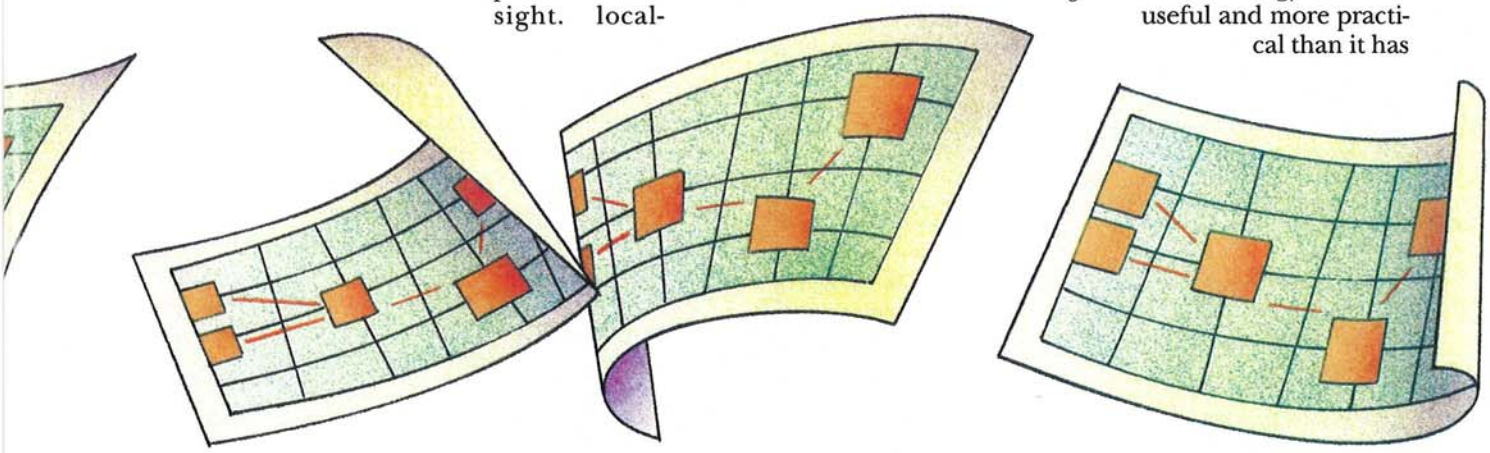
Hand in hand with the new business paradigm has come a strategic shift to distibute computer systems closer to the lines of business. Such change has culminated in a new client/server IT paradigm that for many IT decision makers has been a colossal, growing headache encompassing issues of performance, reliability, security, and cost. And there's no letup in sight.

As distributed operational applications become more widely used across large enterprises, pressure will continue to increase to maintain local copies of key corporate data in order to provide better response time for local queries. As a result, corporate databases—or at least the data residing in those databases—will have to migrate out from the secure and highly optimized sanctuary of the glass house into the distributed and frequently chaotic world of open systems.

Replication, or the copying of data in databases to multiple locations to support distributed applications, is an important new tool for businesses in building competitive service advantages. Replication provides users with their own local copies of data. These local, updatable copies can support increased local-ized processing, reduce network traffic, and, in some cases, they can provide distributed, non-stop processing.

While replication or data copying can clearly provide users with much quicker access to local data, the challenge for IT is to provide these local copies of corporate data in a manner that maintains the same data integrity and operational management that is available with a monolithic, central data repository scheme. For example, if the same inventory records exist on two different systems in two different locations—say, New York and Chicago—the system needs to ensure that the same product isn't sold to two separate customers.

Fortunately, there are new replicator facilities from several vendors that are beginning to make just this technology far more useful and more practical than it has



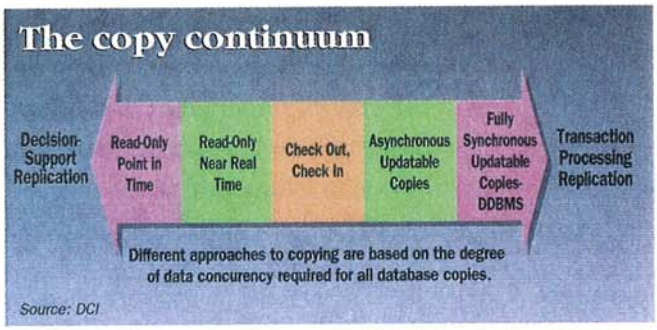
been in the past. A true distributed database management system (DBMS), as defined by most industry consultants, requires the system to support updates at any node on the network.

The concepts behind such technology were pioneered during the late 1970s in the IBM research project R*Star. IBM's subsequent delivery of distributed DBMS products has been part of a 10-year evolving technology known as Distributed Relational Data Architecture (DRDA).

Breaking the 2-phase bottleneck

The first well-publicized distributed DBMS product, announced in 1987, was INGRES/Star. (INGRES was recently acquired by Computer Associates and renamed CA-INGRES). Largely as a marketing ploy, Oracle also announced distributed DBMS capabilities in 1987; however, the first Oracle product to reasonably support distributed database processing is Oracle 7.

The key technology employed to maintain data coherency and integrity in a distributed DBMS is the 2-phase commit. In a distributed DBMS, an update transaction to a database by one client may appear to that client as a simple atomic transaction, while in reality it may involve a host of systems, place a plethora of SQL message packets on the network, and—while transparent to the originating client— the process may be far from transparent to other clients on the network.



Different approaches to copying are based on the degree of data concurrency required for all database copies.

A typical database update begins with reading the "before" image of the data to be modified and continues with establishing locks on the data, changing the data, logging the change, and releasing the locks. This scenario becomes exponentially more difficult when a database is fragmented and distributed across several servers.

Before a server in a distributed database scheme can update its own local data, it must initiate a master process—the two-phase commit— with all of the other servers on the network that also maintain that data. A 2-phase commit begins with a synchronized locking of the data on all of the servers involved. Each server then updates the information, sends a confirmation message back to the master process, and waits for instructions to commit the transaction from the master process. Only after all of the database servers confirm the update, will the master process commit the transaction and release access to the data once again on the network.

All modern distributed DBMS products offer methods for implementing a 2-phase commit. Nonetheless, these procedures are proprietary for each distributed DBMS. There is an XA standard from X/Open which has been implemented in several transaction monitors, but it hasn't been implemented as part of any vendor's DBMS technology.

As a result, the degree of automation support is also different from vendor to vendor. IBM, Oracle, and CA-INGRES all offer a high level of transparency to implementing a 2-phase commit. On the other hand, the Sybase replicator takes a more programming-oriented approach that requires the user to handle some of the handshaking issues, such as having to code DBLib or RPC calls into the application.

Nonetheless, when a synchronized 2-phase commit is combined with data locking, logging and recovery, the

The 10 ingredients

Expert Chris Date has formulated 10 rules that need to be in place for a distributed DBMS.

Rule #1. Local data is managed independently of other sites.

Rule #2. Users don't need to know the location or a path to the data.

Rule #3. No DBMS site is more important than another.

Rule #4. No planned activity should require a shutdown.

Rule #5. A table that has been fragmented will appear as a single table to users.

Rule #6. Redundant data is managed, accessed, and updated transparently.

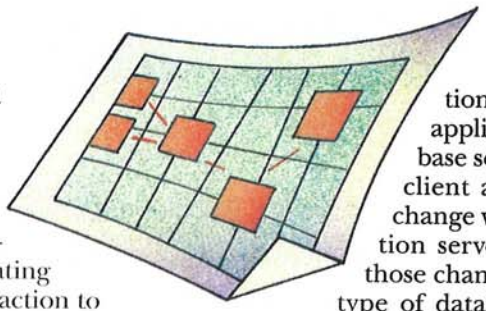
Rule #7. Distributed queries are optimized over the entire network.

Rule #8. Transactions that update multiple sites run with concurrency control and recovery control in case of failure.

Rule #9. There is independence from the hardware, operating system, network, and DBMS.

Rule #10. There is distributed access to the data dictionary.

Source: Chris Date



necessary ingredients for building a distributed database with absolute data synchronization are in place. Unfortunately, this approach to distributed computing is intolerant to failure: Any failure in the network or on any of the local participating databases will cause the entire transaction to fail. At 50 or more nodes, a tightly coupled 2-phase commit process for updating is probably impractical.

Because of this intolerance, distributed DBMS is not typically used to create and manage replicates. The distributed DBMS is more useful where data integrity across multiple sites must be guaranteed. In these environments, the real failure would be to permit updating some nodes in the presence of outages of others.

Replication to the rescue

Replication is the best current solution for many applications because it can be cheaper and more reliable than the alternative of a distributed DBMS engine. A distributed DBMS uses a 2-phase commit to couple together all updates to all locations participating in an update into one very secure transaction. Replication uncouples a local transaction from the process of updating any distributed copies of the data and relies instead on a behind-the-scene process to coordinate all of the multiple updates necessary to synchronize all of the local databases around the network.

Of the many different approaches to replication, each is well suited to solving certain classes of problems. The different types of technologies, in fact, span a scale of approaches. On one side of the copy continuum are approaches that are well suited for supporting decision making, browsing, and research on LAN- based PCs or other platforms. On the other side are classes of technologies that are appropriate for supporting operational systems whose principal role is allowing real-time transaction processing in widely distributed locations.

The type of replication strategy that is most appropriate depends on the problem or application. Decision-support applications often are well supported by technologies that employ simple table copying or snapshot technologies. These technologies can support multiple schemas or data views and are normally set up so that the copies are read-only. These decision-support replication (DSS-R) solutions are often referred to as data warehousing.

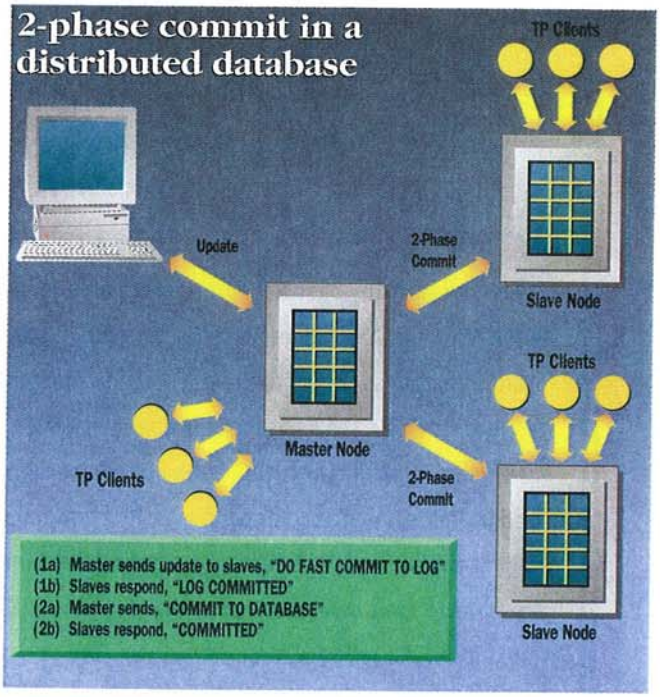
At the opposite end of the technology spectrum from DSS-R are replication approaches that are designed to replace and even improve on distributed on-line or distributed DBMS technologies. These approaches offer near real-time updating against copies of data that may be located in many locations.

As with all replication schemes, transaction processing replication (TP-R) uncouples the synchronous distribution of data copies from the originating application. In a TP-R environment, database servers update their local database for client applications and then register that change with a replication server. The replication server then asynchronously propagates those changes to other database locations. This type of database replication is appropriate for many production systems and normally only imposes a requirement to maintain a single global database schema.

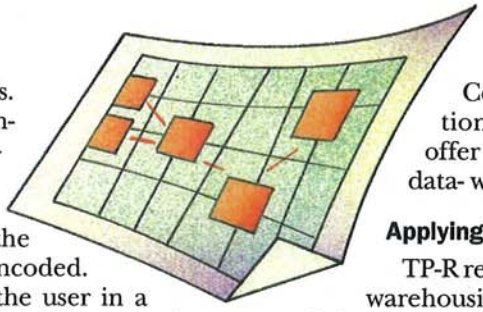
Between the two extremes of DSS-R and TP-R there are many possibilities of combining features and functions for a customized distributed solution. When evaluating replication options, IT decision makers need to take into account requirements for currency, local updates, data enhancement, and history maintenance, among other considerations.

Warehouse shopping

Data warehousing applications are the least demanding on replication services. These applications usually are characterized by a need for data copies that are consistent for a single point in time that often is not the current time. Those who use decision-support systems usually need a historical series of data values over a period of time. In period accounting or trend analysis, a stable data source is essential, and stability often is defined at



In a distributed database, all database nodes are updated synchronously, and a failure at any one node will force a rollback of the transaction at all nodes.

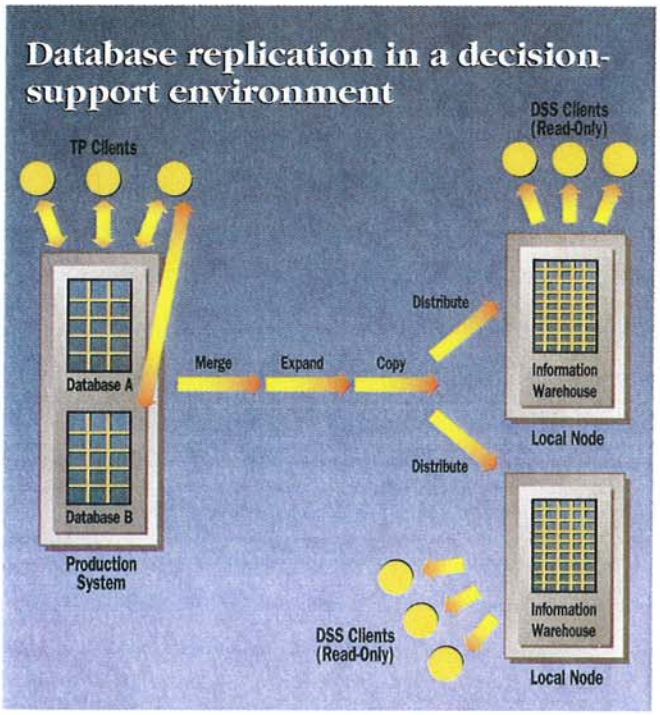


monthly or even quarterly intervals.

With close ties between decision-support systems and data warehouse applications, GUI forms of presentation are becoming a distinct requirement. As a result, the warehouse data often can't be encoded. Instead, it must be presented to the user in a form that's comfortable and familiar. To ensure the usefulness of the data maintained in a warehouse copy, the DBMS must often perform derivation, aggregation, and transformation functions on the raw data before copying it into the data warehouse.

Often data warehouse applications are read-only. Updates, as they occur, are performed only on the source production system database from which the warehouse copy was created. It is possible, however, to have an environment where updates can be processed against both production and warehouse databases. This is done by keeping the two in a synchronous state with a 2-phase commit update against both source and target data. Normally, this is not a good idea because of specialized tuning for the read-only copy that allows it to perform better in decision support. Transaction- processing updates will likely interfere with its job efficiency.

IBM is probably the leader in offering the technology needed to support data warehousing. Digital Equipment Corp., Hewlett Packard, and Information Builders are other companies that offer important technology for supporting data- warehouse approaches.



Decision-support systems require the least amount of data concurrency. Most often, the clients of a DSS data warehouse have read-only access, and the information is decoded, and calculated fields are expanded.

Applying TP-R

TP-R requires a very different technology than warehousing. Production systems nee d the current state of data, not its history. Any node must therefore allow updates to production data.

Unlike DSS-R schemes, TP-R propagation of updates to secondary locations need to occur as soon as possible following the changes to the local database. Often that propagation is done in near real-time with a separate 2-phase commit to each target copy location.

This update scheme differs significantly from that of a distributed DBMS, which employs a single 2-phase commit to synchronously lock all of the data copies until all locations respond with a "committed" message. In a TP-R replication scheme, the single 2-phase commit carried out by the distributed DBMS is replaced by "n" (where "n" is the number of separate data locations) distinct and separate 2-phase commits that are carried out by a replication server.

Once the replication server is notified by the local DBMS that a transaction is waiting to replicated, the server examines the distribution queues and schedules multiple subtransactions to update each of the target databases. Since the target databases are typically remote, the replication server uses a separate 2-phase commit protocol when moving transactions from the distribution queue to each individual target database. Should any target database be off-line or other wise unavailable for updating, its associated update transaction will remain in the distribution queue until a time when the target database can be synchronized with the source.

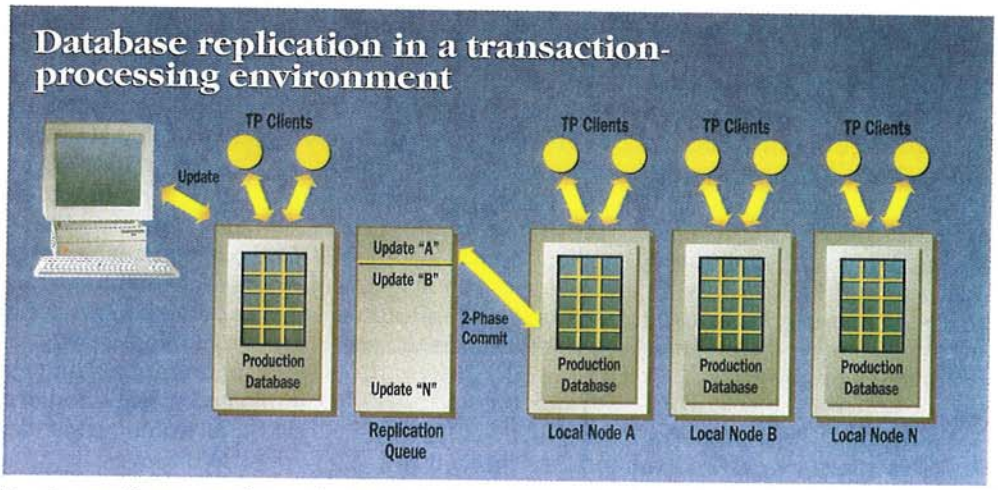
The main benefits arising out of this TP-R approach are faster overall system processing, faster local commits of transactions, and the potential for significantly reduced network traffic. More importantly, this TP-based replication approach is more fault-tolerant than distributed DBMS and therefore more appropriate for many applications.

Imagine a retail operation where sales offices are widely distributed and inventory is kept at a few major supply depots. If the supply depot information is replicated at the sales offices, then it's possible for the sales office to accept tentative orders even if the network link to the local supply depot is broken. The sales office can accomplish all of the processing necessary for a sale except for a final confirmation without access to the central source inventory data.

Applications like order processing and hotel or airline reservations that do not require absolute data synchronization are excellent targets for TP-based replication approaches. Any application that can deal with some

inconsistency among the different data nodes for short periods of time may benefit from the use of a replication server in place of a full-blown distributed DBMS. After all, airlines and hotels overbook intentionally.

TP-R systems are easily able to maintain transaction consistency for updates that span multiple tables at one or more target sites. To ensure efficiency for the input and processing of data, these systems frequently maintain replicated data in an encoded stateand subset that data among servers, since each production location often has no need to access all of the global data.



In a transaction-processing environment, database replication is done in near real-time to maintain very high data concurrency. Replication, however, does not occur as a synchronous 2-way commit to "n" nodes, but as asynchronous 2-phase commits.

The leaders in TP-R approaches are Sybase and CA-INGRES. Sybase's architecture is built around a master/slave concept, while CA-INGRES is based on a peer-to-peer model. A transaction managed through a replication approach is considered successful if it is committed at one site in a peer-to-peer system or at the master site in a master/slave approach.

A copy is a copy is a copy, or is it?

DSS-R and TP-R offer two different approaches to providing asynchronous replication of production databases. In both cases, the process which transmits the updates has to be reliable by ensuring that the copies get to the targets. In addition, the process needs to be valid by ensuring that the necessary data integrity is maintained at the target. It is here, concerning the integrity and validity of data, that the two replication approaches, DSS-R and TP-R, usually part technological company.

In TP-R environments, the integrity of data at the target site is maintained by applying database copy updates one transaction at a time. Changed data from one user transaction can span multiple tables at each location to be updated. At each database server in a TP-R environment, all or none of its tables is updated in any one transaction. In this way, the local data on each network node remains consistent across all tables at all times.

In contrast, DSS-R approaches update each local database table-by-table. All tables that may have been affected by one transaction are not committed in the same unit of work under DSS-R.

The DSS-R approach is usually far more efficient in computer and network resources, especially since it allows for the net result of a series of updates to be transmitted rather than the propagation of all the individual changes themselves. However, this "netting out" isn't appropriate for transaction-based environments.

More importantly, both DSS-R and TP-R approaches to database replication can work in environments where a distributed DBMS approach just wouldn't work.

Imagine a situation with 100 target database nodes, only 90 of which are available. A replication server would successfully perform 90 separate 2-phase commit transactions, each with two branches. The remaining 10 transactions would be re-queued and attempted at a later time. A distributed DBMS would attempt to perform one 2-phase commit transaction with 100 branches and fail, forcing all of the transactions to be rolled back.

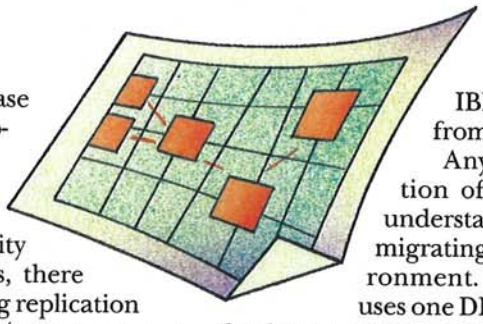
Even if all 100 nodes were on line, the distributed DBMS would hold locks on all 100 targets until all 100 were willing to commit. In any environment that includes an unreliable WAN or any environment that contains many nodes that cannot afford to be blocked, the distributed DBMS solution just won't work.

Replication in an open environment

Replication technology can be instrumental in allowing more efficient usage of a company's computers and network. As companies migrate to decentralized operations, they naturally want their IT support to follow along the same lines.

As the workload is distributed, there are significant cost savings attached to using multiple smaller machines to process work. Replication, done intelligently, can reduce network traffic and allow the user to derive benefit from what would otherwise be unused CPU cycles. Another way to look at this is that replication allows easy local data access at remote sites.

This kind of capability provides for a higher level of customer service than what could be provided by a sys-



tem operating off a single central database with communication links to the distributed sales offices.

For a distributed operation, then, replication of both TP-R and DSS-R types allows for higher system availability than a monolithic model. Nonetheless, there remains one very large caveat concerning replication technology in the world of open client/server computing.

Today, there are no standards that apply to replication across diverse products. And there are no standards bodies working on this issue.

Factoring in the role of the vendor

All of the major DBMS vendors are moving toward opening up their replication capabilities to foreign data sources. Digital, Oracle, Sybase, and IBM are focusing their attention on links to each other and other relational DBMS products. IBM, CA-INGRES, and Sybase have published their 2-phase commit protocols, which allows users of their products to participate in heterogeneous distributed database solutions with products from other vendors.

Both Sybase and CA-INGRES have links to non-relational data sources in their target replication capability. Normally, if the vendor supports a DBMS gateway to any foreign data source, then that data source can also serve as a target for replication.

Such gateways to non-relational data sources don't require special coding like RPCs and are valuable in allowing the integration of new distributed systems with older legacy applications. For both Sybase and CA-INGRES, external non-relational data sources include both VSAM and RMS files.

As a general rule, replication from a foreign DBMS into a replication environment like CA-INGRES or Sybase is only available now if the user is willing to program that functionality. One important exception is an IBM offering that allows replication from IMS into the DB2/DRDA world.

Anyone contemplating the acquisition of replication technology should understand how your vendor will assist in migrating to a heterogeneous DBMS environment. Almost no organization today uses one DBMS exclusively. Heterogeneity in database and file management approaches is likely to increase in the future.

Gateway solutions, of course, are not the same as a replication and 2-phase commit process that transparently operates over multiple databases. The real world is multivendor, multidepartment, and multinetwork. Replication technology that can operate well across heterogeneous DBMSes is something that DBMS users will want.

George Schussel is president of DCI, an Andover, Mass., consulting firm that sponsors a number of symposiums, including the DCE Developers Conference. Starting next month, Schussel, a frequent speaker on database and client/server development issues, begins a new column called Convergent Views in Client/Server Today. CSI