DATABASE REPLICATION: *PLAYING BOTH ENDS AGAINST THE MIDDLEWARE*

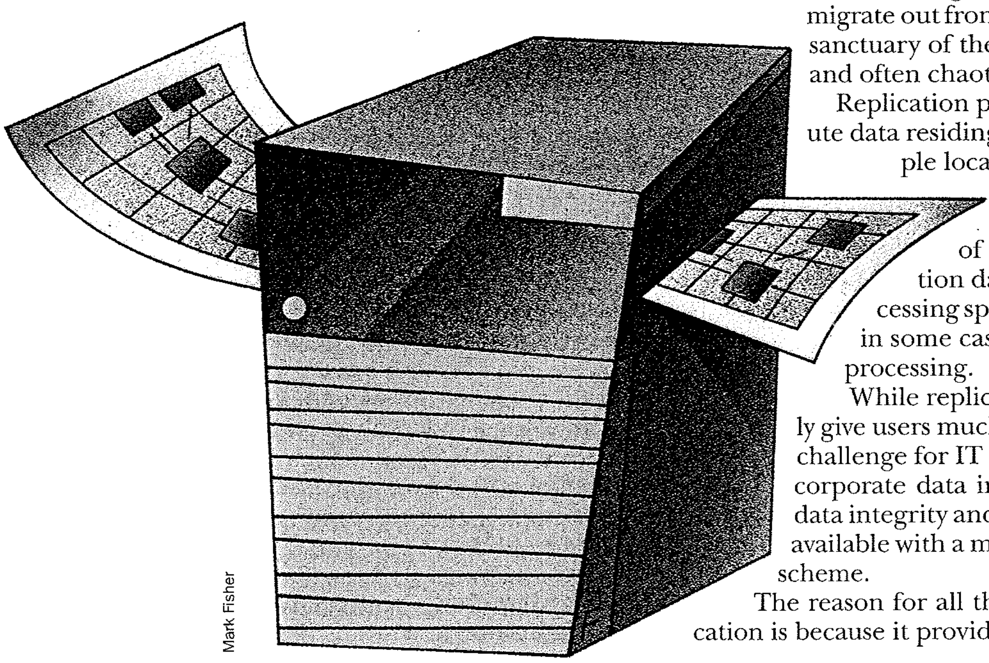
This is the second of a two-part series. Database replication is designed to provide the data-availability benefits of distributed databases, while avoiding the inherent bottleneck of synchronized 2-phase commits. The first article in the series focused on replication's overall importance and benefits, and provided an introduction to the two forms of replication services. This month, we examine in detail the differences between these two forms of replication: data warehousing and high-end transaction-processing class replication servers.

Using middleware constructs like message queues, database replication servers offer advantages traditional distributed database architectures can't deliver

BY GEORGE SCHUSSEL

As distributed operational applications become more widely used across large enterprises, pressures are increasing to maintain local copies of key corporate data to improve 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 often chaotic world of open systems.

Replication provides a way to copy and distribute data residing in corporate databases to multiple locations for use in distributed applications. Replication provides users with autonomous control of their own local copies of production data in order to enhance local processing speeds, reduce network traffic, and, in some cases, provide distributed, non-stop processing.

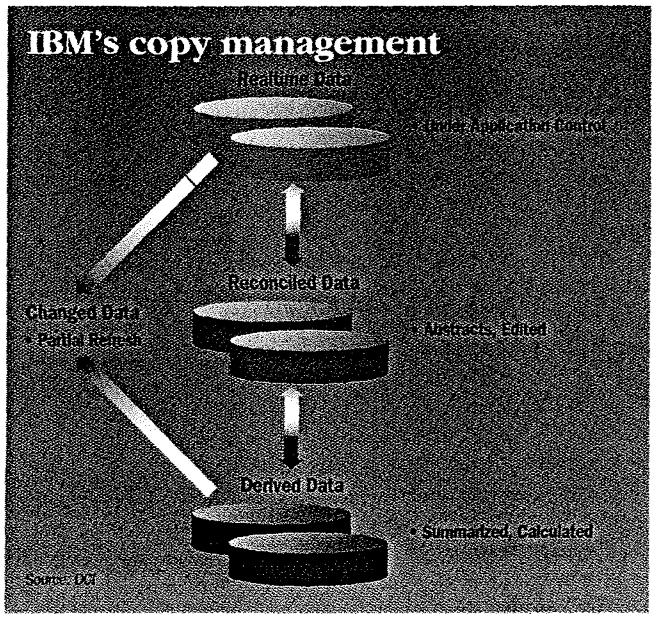

Mark Fisher

While replication, or data copying, can clearly give users much quicker access to local data, the challenge for IT is to provide these local copies of corporate data in a way that maintains the same data integrity and operational management that is available with a monolithic, central data repository scheme.

The reason for all the attention being given to replication is because it provides the best current solution for a cheaper and more reliable alternative to a distributed DBMS engine.

A distributed DBMS uses a 2-phase commit to couple 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. It relies instead on a behind-the-scenes middleware process to coordinate all of the multiple updates that are necessary to synchronize all of the local databases around the network. CA-OpenIngres, for example, relies on message queuing to manage the replication process.



IBM uses a three-tiered approach to provide data warehousing functionality for decision support.

Although many DBMS vendors are talking about replication offerings, it would be a mistake to assume that replication is a commodity. Different architectural approaches toward implementing replication provide fundamentally different capabilities.

Each approach to replication 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 realtime transaction processing in widely distributed locations.

Which type of replication strategy is most appropriate depends on the problem or application. Decision-support applications are often well supported by technologies that employ simple table copying, or snapshot technologies. Moreover, these technologies can support multiple schemas or data views, and they 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.

DSS-R approaches to replication usually are built on various technology variations of table copying. Tables at the target location are created one at a time, drawing from one or more source tables or files. DSS-R copies are inherently read-only.

Most approaches provide for transaction-consistent data within a table, but are not concerned with transaction consistency across sets of target tables. A common environment is for tables to be updated after the close of business, so fully consistent environments are established by the morning.

Decision-support replication schema

The typical decision-support application has a requirement for consistent period data sources and not necessarily for data that is up-to-the-minute current. DSS-R approaches, then, don't typically worry about keeping the data current: Daily or less often is typical scheduling for updates.

Consistent, stable data for a given period is the highest requirement for these types of applications.

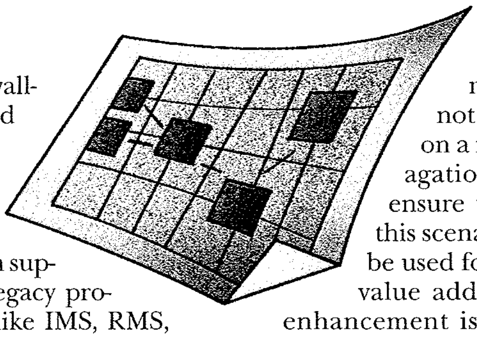
More importantly, decision-support systems maintain historical records that end users will not typically need to update. Such data stores are tuned explicitly for query processing.

Frequently, such tuning adds more indexes and expands stored data by storing values that would have been calculated in the production database. As a result, continuous propagation of updates would interfere with the query tool's ability to provide reasonable performance above and beyond the additional load that would be created on the replication server.

The replication server in a DSS-R environment should therefore provide various timing options, which can cre-

Stonebraker's steps for failover reconstruction

1. Understand what is broken.
2. Understand how the break occurred.
3. Determine how to fix the damage and reinstate the broken pieces.
4. Bring back the broken pieces on line.
5. Make sure that recovery of the databases results in consistent data across those databases.



ate copies based on timed events (wall-clock time or the passing of a fixed time interval), based on application events (completion of an end-of-day reconciliation), or based on explicit operator requests. Other important requirements for decision support include the ability to access legacy production system data from sources like IMS, RMS, VSAM, and flat files and to provide for sophisticated manipulation and enhancement to that data.

An example of data enhancement is what IBM has implemented in its Information Warehouse, which is a sort of three-schema architecture for decision-support purposes. Recognizing that operational systems frequently are not correctly structured for supporting queries, IBM offers reconciled copies and derived data that summarize and add calculated values to copies of data made available for decision support.

These copies can be updated at any time and according to criteria established by the database administrator.

Time-based data is also important, particularly where trend analysis is desired. For this capability, the maintenance of data histories is important. Such histories can include complete records of all activities to a table, summaries based on point-in-time source data, and summaries based on changed data.

DSS-R approaches are very useful in situations where companies are downsizing and the distributed applications need to share data with host legacy systems.

The assumption of DSS-R is that updates will be made at the single source sites, not at the data copy sites. Sometimes, source data is in a central host, but other times it can be located in remote locations that own distinct data fragments. The data copies, however, are read-only.

The predominant technology for DSS-R replication is some form of extract, manipulate, and further processing. These runs are typically batch jobs that occur after on-line transaction processing has ceased. It is much simpler to ensure that consistent transaction data is copied when the source tables are not being updated.

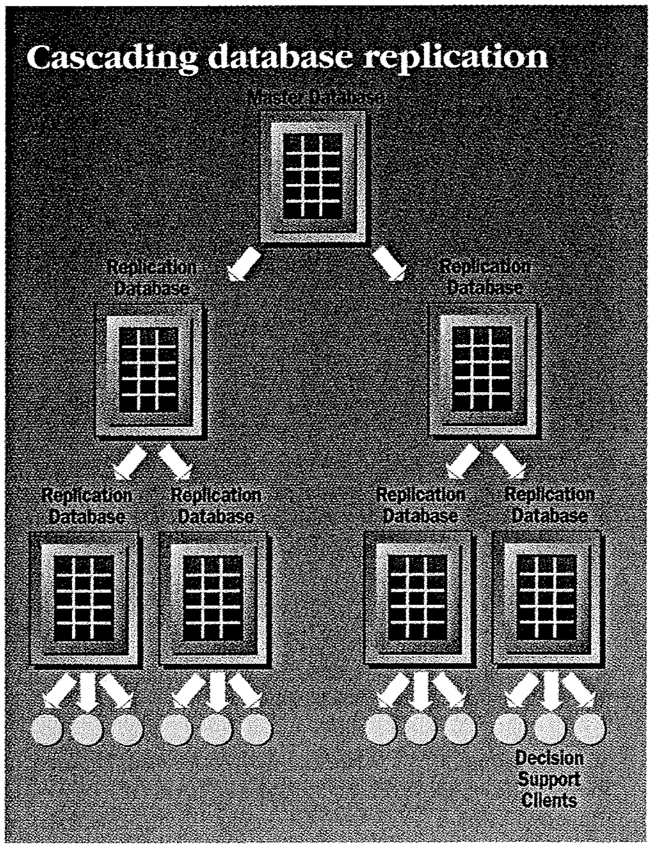
A common application model for such data copying occurs at companies with many branch offices.

In such a scenario, there may be hundreds or even thousands of database servers. With so many copies to be made, efficient distribution requires support for cascading replicates where copies can be made from other copies in a very coarsely-granular, parallel-processing scheme, where the central host feeds a small number of distribution nodes, which in turn feed multiple branch-office sites with data that is properly subsetted for their local operations.

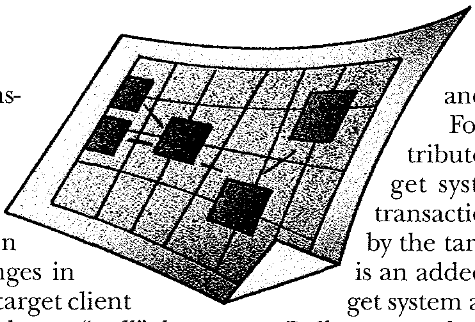
Alternatively, DSS-R may be provided through propagation of source table changes to the target. In large database environments with multiple 100s of gigabytes, transmission of a full refresh table copy is not economically or technically feasible on a nightly basis. As a result, change propagation is the only solution. In order to ensure that consistent data is propagated in this scenario, a 2-phase commit process should be used for the changed data transactions. The value added to the data by manipulation or enhancement is very important in DSS-R environments. Sources are typically legacy systems. The replication solution should provide the ability to restructure the data from legacy formats into the relational model.

Tools should provide support for performing relational joins of data from multiple sources, for calculating new values, for aggregating data and for transforming encoded data into descriptive forms.

An important point to keep in mind is that while one of the principal benefits of DSS-R is the aggregation of data, or denormalization as it is sometimes called, this process should not be done when the replicated database is updatable. The reason for this will be discussed further



By cascading database replication updates from the master database over multiple replication servers, the processing of updates for sites with hundreds or even thousands of nodes can be rapidly speeded up.



on, in the section that deals with transaction-processing replication schema.

Where change capture and propagation schemes are used, there is a choice in the distribution model: whether to "push" the changes in the data from the source host to the target client systems as the changes occur, or whether to "pull" the changes in the data from the source system as the target clients request the changes.

Optimization: push and pull

In general, the push model is best for continuous, almost realtime propagation. The pull model, which provides greater flexibility in reformatting and combining the data on the source system, is best for looser currency requirements. The pull model also allows more control and flexibility in timing network traffic.

For example, push systems typically distribute every transaction to the target. Target systems must therefore process every transaction. If only summary data is required by the target system, then data transformation is an added processing cost required of the target system as part of the replication process.

Pull systems, however, provide the opportunity for aggregation prior to distribution. This is effective both where only summary data is required, and where replication of numerous changes to database hot spots (areas within the database that receive the most update activity) can be deferred until they can be aggregated and the effects of all of the changes netted out.

For decision-support or other static data applications, the need for near realtime information may not be important. For these applications, the needs for multiple schemas or data views, for efficient query processing, and for a consistent stable database over a specific period of time can be best satisfied with a decision-support replication schema.

What to do, what to look for in a database replication server

Set up a plan. Understand the rules for distribution of data **before** implementation begins.

Make sure that your distributed-database administrator has good forms-based CA-OpenIngres, or GUI-based–Sybase System 10 utilities to help configure the database and manage the network.

Consider:

—how you specify enhancements to the data, and whether you will have to learn a new language for this function

—how the replication setup is handled, and how much automated support is provided

—what support is provided for automatically handling failure managementand how much intervention by the database administrator will be required

Make sure your utilities can tell you which tables, columns, and rows are located at the various nodes and to which nodes transactions are routed.

You should be able to change the database configuration on the fly without bringing the database or replication operation to a standstill.

There should be a mail-based error notification system to allow management of the distributed enterprise from any node on the network.

Transaction processing replication schema

If you are distributing production operational systems, DSS-R technology isn't likely to work for you. A transaction processing replication (TP-R) approach that can maintain near realtime transaction integrity at data copy sites is essential. TP-R replication is primarily concerned with creating a single image of a database across distributed autonomous sites and preserving database integrity in near realtime processing. The overall integrity of databases is preserved by forwarding data changes resulting from single user transactions.

TP-R approaches have been implemented with two fundamentally different architectures by Computer Associates in CA-OpenIngres and Sybase in Sybase System 10. CA-OpenIngres has built its replicator on a peer-to-peer architecture approach. Sybase System 10 uses a master/slave approach.

Nonetheless, Computer Associates maintains that because CA-OpenIngres was designed to be operated in a peer-to-peer environment, it can function in a master/slave mode as well.

In addition, Computer Associates touts the ability for CA-OpenIngres to be configured in a hybrid configuration called central/branch in which updates can be made at any local database. However, replication of all updates is managed by a central server.

Adding even further to the confusion, Sybase claims that it is possible to configure Sybase System 10 in a peer-to-peer scheme. Nonetheless, the amount of middleware that would be required to implement such a schema at an end-user site is nothing short of prodigious.

In the master/slave architecture, every table or table fragment is assigned to a primary or master site. In a master/slave scheme, data is replicated only in one direction: from master to slave. Updates to the database must successfully complete at the master before the transaction is considered to be a success, as far as the application is concerned. Sybase System 10 uses asynchronous stored proceedures in a master/slave topology to redirect updates made at a local database first to the master system so that the update can be replicated back to the local database.

This can present a problem for remotely generated transactions. This is because those processes cannot update their local or other sites, until they are first routed synchronously through their primary tables.

If the primary table's database server fails or access to that server from the network is denied, replication does not occur, and the transaction is queued. Under the Sybase System 10 scheme, a user issues an update via a stored procedure, which is passed from the local replication server to the primary or master replication server. The primary database server executes the stored procedure, and that change is then replicated by the primary replication server and passed back to the local replication server. On receiving the transaction from the primary server, the local replication server can then update the local database server.

On the other hand, updates in peer-to-peer approaches can be made to any data location and then copied into other locations.

A transaction is successfully completed as soon as any one or combination of locations is able to update one complete copy of the affected data. Peer-to-peer allows all locations to own and manipulate any data, broadcasting changes as required.

Implementing such an update-anywhere replication strategy for vendors is not an easy task. The primary stumbling block is how to resolve the inevitable conflict that occurs when two users attempt to update the same record on different servers at the same time.

While the Sybase architecture is master/slave, the vendor states that its Sybase System 10 can be set up to support a peer-to-peer replication server approach.

However, if you want to build a peer-to-peer architecture with Sybase technology, you will have to write your own software for collision identification, resolution, and recovery.

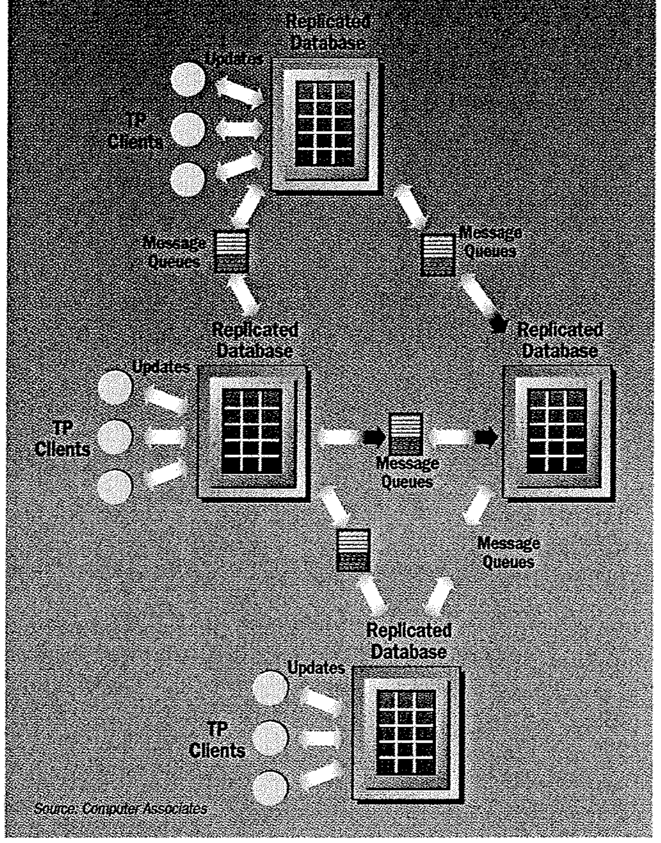
Fundamentally, the master/slave approach to TP-R can be characterized as simpler for vendors to implement because it eliminates the potential problem of update collisions. For users, the relative implementation simplicity of a master/slave schema can often result in improved performance of applications over a peer-to-peer implementation, thanks to lower DBMS overhead. On the other hand, the master/slave schema introduces a single point of failure that can lower overall system availability. Computer Associates with CA-OpenIngres is the only vendor at this time that has a true peer-to-peer replication architecture, which is the most general and the most powerful approach to TP-R replication.

It is closest in capability to a true distributed DBMS in that there is no limitation on where data can be located or updated. Moreover, as a result of a peer-to-peer replication server's use of many individual 2-phase commits to broadcast data changes asynchronously from the originating application, a peer-to-peer replication scheme is more fault-tolerant than a distributed DBMS.

Collisions with peer-to-peer architecture

The possibility of collisions remains a nagging problem with the peer-to-peer replication approach. A collision occurs when a record that is physically replicated at

Peer-to-peer replication under CA-OpenIngres



CA-OpenIngres uses message queuing as a medium to stage near realtime database replication in a transaction-processing environment. In a peer-to-peer schema, CA-OpenIngres permits updates to be made at any node and then relies on that node to send the proper update messages to keep all of the copies of the data synchronized.

two or more sites is repeatedly updated during the asynchronous latency period of a replication update. In other words, after the first update has happened at one site, a second update occurs and is processed at another site before the propagation of the first update has been completed. Therefore, while a peer-to-peer approach provides the most general solution for transaction distribution, it requires software for collision resolution.

When a collision occurs, there is no way to construct an application-independent approach that can recover all different types of databases. However, the replication server can and should have collision-resolution logic.

Therefore, from the moment any transaction is committed, the replication server must keep track of all of the processes that happen during the processing and distribution of that transaction. That is because in the event of a collision, this information must be available to properly resolve the collision.

To resolve that conflict, the replication server should support multiple options from which the database administrator can choose.

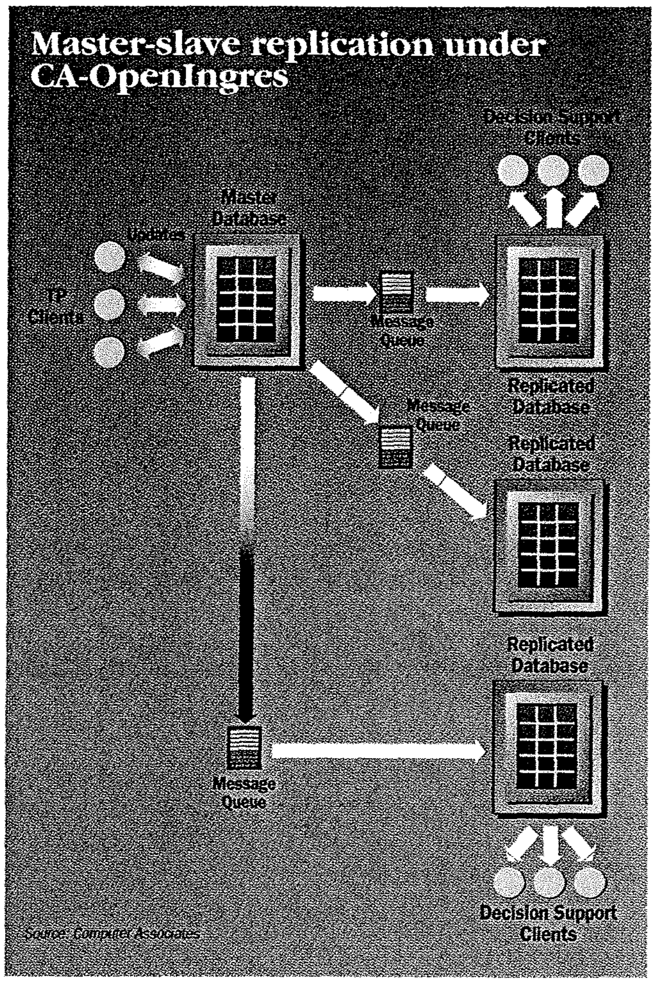
Examples of resolution possibilities include: giving priority to the initial update; rolling back any later conflicting transactions giving prioity to the last update; overwriting any earlier conflicting transactions and sending messages to designated parties; resolving the conflict by firing a user-specified trigger; and, finally, halting the replication process and sending a message to the database administrator.

In order for a number of these conflict-resolution strategies to work, it would be very helpful if a distributed time service was available.

Unfortunately, current replication servers don't provide this service and instead rely on the separate operating system clocks. If they are not synchronized, errors will result. An important new facility for this service is OSF's Distributed Computing Environment (DCE), which provides the necessary synchronization.

Experience to date with users of peer-to-peer replication indicates that if the replication timing chosen is ASAP and if the databases have been properly designed for replication, the volume of collisions is likely to be quite low.

The conflicts that do occur can be handled by rules in a collision-resolution software module that logs entries for manual review. Future capabilities for replication servers in this area may include expert systems to help resolve collisions.



Using the message queue construct, CA-OpenIngres can also implement a master/slave topology in which all updates are promulgated from the master database.

The fault-tolerance advantage

One key benefit of all replication approaches is added fault tolerance for a distributed computing environment.

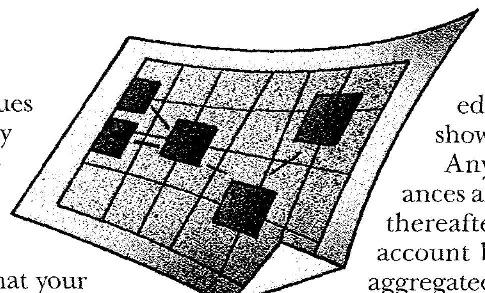
Fault tolerance provides the overall system with a capability of continuing to function when a piece of the environment is down.

When something breaks, then, the system working in combination with the database administrator should provide as much assistance as possible in the recovery process. Mike Stonebraker, the father of Ingres, has used the phrase "failover reconstruction" to describe when this recovery process occurs automatically under software control. The highest level of fault tolerance will be from a system supporting peer-to-peer replication.

That's because the system considers an update to be successfully completed when it has completed a database update at any peer site.

The site that is updated is like a floating master in this case. The replication server will queue the updates to all other data locations. In a master/slave architecture, if access to the master is denied, then the update is not allowed from the application.

When the master location becomes available it will be updated. After the master has been updated, the replication server attempts to update the slaves. If there is a



failure on the network, the master queues the updates for the slaves for delivery when they become available. This system works as well as a peer-to-peer approach as long as neither the master node nor the network fails.

If either is the case, it's important that your system provide the necessary utilities to allow the rebuilding of remote databases from information on the local log and database information on other remote databases. One such key utility should provide the ability to "difference" replicates–in other words, to look at a master and slave or two peers and determine if inconsistencies exist.

Transparency and richness of function

For a replication server product to be successful, it has to provide enough added function over what customers have developed for themselves, and it should provide that function transparently . There is a significant difference in the amount of replication functionality and in the ease of implementing replication services by various DBMS vendors. Some products require significant programming with database triggers or database calls to implement replication.

Most of the current replication functionality in Oracle 7 and much of the service available through Sybase System 10 Replication Server require programming with RPCs or DBLib calls by the distributed database administrator.

Setting up database replication with CA-OpenIngres is easier, in that a configuration manager is provided that offers a three- step, forms-based approach to defining the replicated environment.

In order to provide transparent replication services to applications, the database administrator in a distributed database environment needs to be very aware of the use of a replication server, and needs to have designed the database in a way that's conducive to a distributed operation. In practice, this issue means that denormalized or aggregated data should not be replicated in TP-R situations. Such derived or aggregated data should be computed at each site from the basic data contained in a transaction.

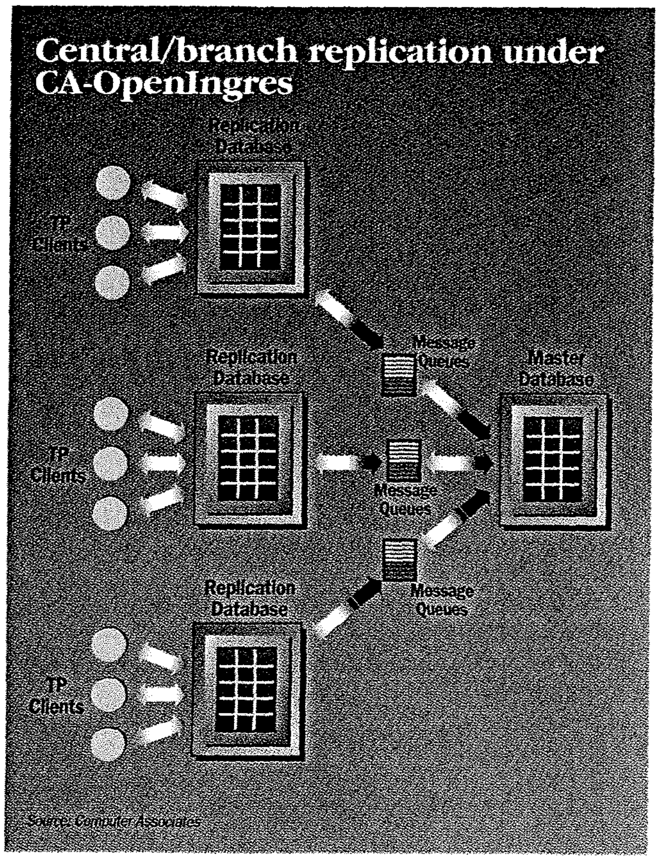
To see this point more clearly, let's look at a simple banking example in which aggregated account balance information is replicated at a bank's three branches–A, B, and C.

Suppose we look at one customer's balance, which is $100 at all three branches. If the network goes down at Branch A and the customer makes a $40 withdrawal at Branch B, then both branches B and C will show that account as having a $60 balance. If the customer then makes a $30 withdrawal at Branch A before the network comes back up and the earlier transactions are replicated at Branch A, then Branch A will show a balance of $70.

Any attempt to reconcile the balances among the three banks at any time thereafter will fail. That is because the account balance field in this example is aggregated and denormalized.

Replicating balance information is going to cause integrity problems with the databases. If the system had simply replicated the transaction amounts–normalized data–each site would be able to recover correctly by using a time order to sequence and compute the balances. In general, a good rule for distributed processing is to use local database triggers to handle computed amounts like account balances.

Furthermore, your application should not need to concern itself with the timing of the asynchronous distribution of data to target sites. Getting this functionality from your replication server also should not require you to do



CA-OpenIngres can also be configured in a replication scheme that is a hybrid of the peer-to-peer and master/slave topologies. In this configuration, updates can be made on any node–just as in a peer-to-peer scheme. Replication, however, is handled by a central server, and client databases do not communicate directly with one another, as in a master/slave scheme.

programming. The nature of system usage will dictate the type of timing used in replication.

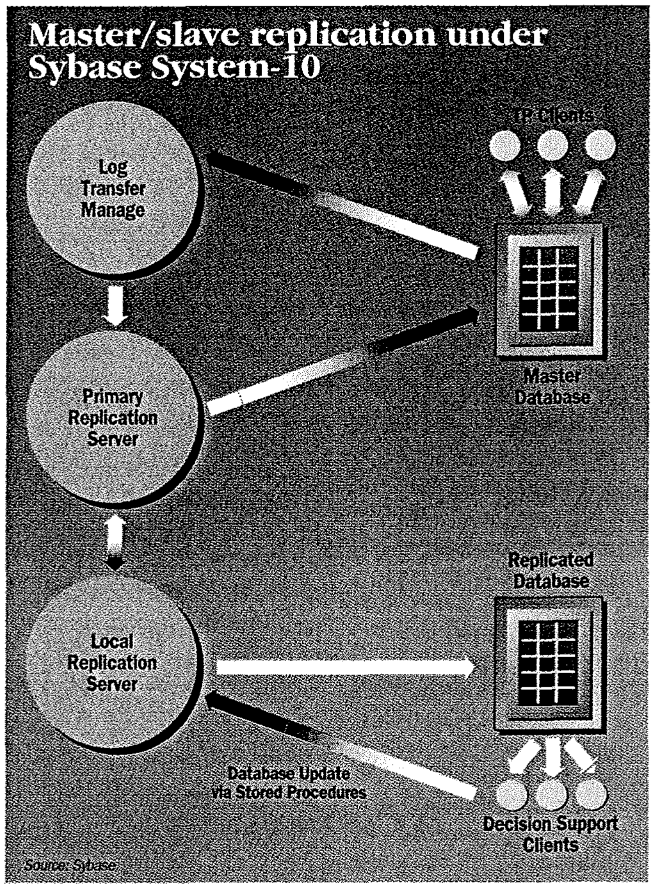
For operational systems that expect to be updated with near realtime transactions, the best approach is likely to be ASAP. There is no additional processing overhead attached to ASAP replication in this case, because the user is likely to be in a situation where the copy distribution is under 2-phase control for each updated site. In such a case, then, there is no processing savings attached to batching the transactions.

For decision-support or period accounting types of systems, a stable database that is consistent throughout may be preferable to having the most current status. In this case, for reasons discussed above, scheduled rather than ASAP replication may be preferable.

A few good DBAs

The benefits of a properly implemented replication scheme can be substantial. A distributed environment's complexity, however, in both a managerial and technical sense, is much greater than that of a local, monolithic environment. This is especially true for TP-R environments.

Data collisions may occur with peer-to-peer approaches; the recovery process that this implies requires both the cooperation of excellent software and of competent administration.



Updates to a replicated database under Sybase System 10 must be processed at the master database node. A stored proceedure is first initiated at the local replication server, which passes the procedure to the master replication server. The latter then initiates the update action on the primary database server. This action triggers the log transfer manager to cause the primary replication server to send an update notice to the local replication server, which finally updates the local database server.

Managing distributed data through replication and copy approaches is non-trivial. It will require competent technical management. Just evaluating the different technologies that are currently available will require an analyst who is of top caliber.

It is wise, therefore, to invest the necessary resources to make sure that the combination of local and global database administrators (DBAs) is adequate for your environment.

Your database administrator will have to create a database design that is correct for replication and test it in the distributed environment. It is important not to shortchange the time that it takes for your database administrator to become an expert in diagnosing and resolving problems in such an environment.

Finally, because implementing distributed systems offers so many combinations of technology and benefits, you will need to do some careful management analysis in order to understand how these approaches can support your business requirements. CST

Consultant George Schussel is founder and chairman of Digital Consulting Inc. (DCI) in Andover, Mass. A CIO, consultant, industry analyst, and lecturer, Schussel has written a new book, coauthored with Steve Guengerich, titled Rightsizing Information Systems *(SAMS Publishing).*

COMPANIES TO CONTACT FOR DATABASE REPLICATION PRODUCTS

COMPUTER ASSOCIATES ▶ CIRCLE 915	IBM ▶ CIRCLE 917	INFORMIX ▶ CIRCLE 919	RED BRICK SYSTEMS ▶ CIRCLE 921
DIGITAL EQUIPMENT ▶ CIRCLE 916	INFORMATION BUILDERS ▶ CIRCLE 918	MICROSOFT ▶ CIRCLE 920	SYBASE ▶ CIRCLE 922