REPLICATION, THE NEXT GENERATION OF DISTRIBUTED DATABASE TECHNOLOGY

by George Schussel
Digital Consulting, Inc.
204 Andover St.
Andover MA, USA 01810

Tel 508/470-3870
Fax 508/470-2002
CompuServe 74407,2472

June 15, 1994

November 1, 1994

REPLICATION, THE NEXT GENERATION OF DISTRIBUTED DATABASE TECHNOLOGY

About the Author

Dr. George Schussel, DCI's founder, is Chairman of Database & Client/Server World and a world-renowned authority on information systems and client/server technology. He is a lecturer, writer and consultant to the computer industry and major user companies on data processing issues.

Introduction

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. New replicator facilities from several vendors are making this technology much more useful and practical than it's been in the past. In this article we will go into enough detail on replication for the reader to understand the importance of replication, its benefits and some of the related technical issues.

Buying trends today clearly indicate that companies want their applications to be open and distributed closer to the line of business. This means that databases supporting those companies have to migrate to this same open, distributed world. As distributed operational applications become more widely used across large enterprises there is going to be a requirement for increasing numbers of data copies to support timely local response. This is because the propagation uncertainties and costs associated with real time networks and/or distributed DBMS solutions are a headache to deal with.

Replication provides users with their own local copies of data. These local, updatable data copies can support increased localized processing, reduced network traffic, easy scalability and cheaper approaches for distributed, non-stop processing.

While replication or data copying can clearly provide users with local and therefore much quicker access to data, the challenge is to provide these copies to users so that the overall systems operate with the same integrity and management capacity that is available with a monolithic, central model. 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 insure that the same product isn't sold to two separate customers.

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. This becomes difficult as the number of participating nodes increases. With 50 or more nodes a tightly coupled 2-phase commit process for updating is probably impractical. A replication approach uncouples the applications from the underlying multiple data copies and allows the applications to proceed while behind the scene the replication server handles the coordination of multiple updates. The difference in approaches between replication and distributed DBMS approaches will be discussed in detail below.

Two Different Approaches: Warehousing versus Replication for Transactions
There are many different approaches to replication, each well suited to solving certain classes of problems. The different types of technologies, in fact, span a

scale of approaches as is illustrated as in Figure 1. On the right 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. On the left side of this scale are approaches that are well suited for supporting decision making, browsing and research on LAN based PC's or other platforms.

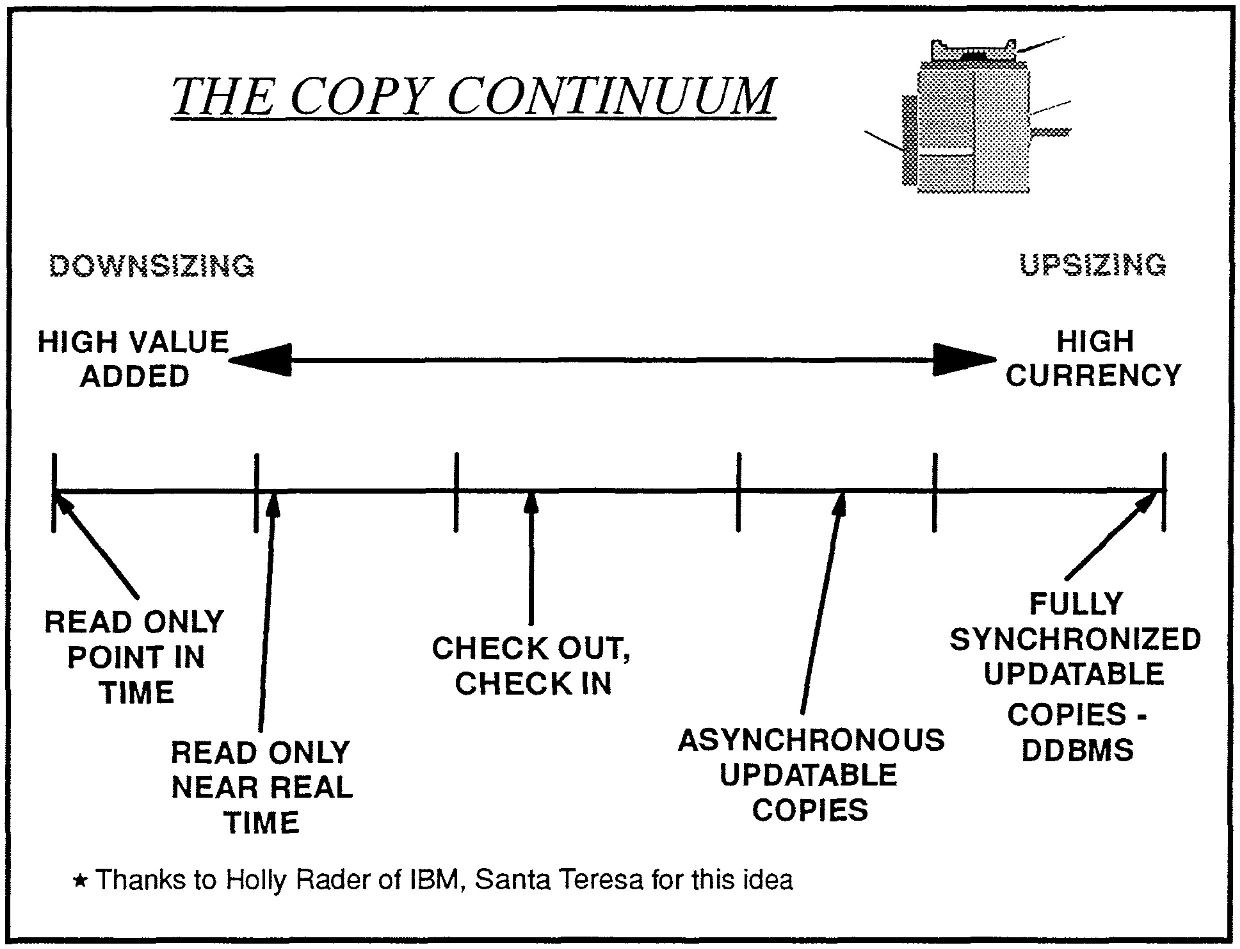


Figure 1 - Different approaches to copying for different purposes

The type of replication strategy that is appropriate is very problem or application dependent. Decision support applications are often well supported by technologies that employ table copying or snapshot technologies. These technologies (the term "warehouse" is often applied here) can support multiple schema's or data views and are normally set up so that the copies are "read only". For simplicity in this paper, these approaches will be referred to as warehouse or (DSS-R, Decision Support Systems - Replication).

Warehousing applications are usually characterized by a need for data copies that are consistent for a single point in time; that point doesn't necessarily have to be the current time and sometimes it's preferable that it's not. In period accounting or trend analysis, for example, a stable data source is essential. In decision support systems (DSS) one usually needs history for a series of data values. Multidimensional or matrix representation of the data with one of the axes being time is frequently used.

The data, when it's presented to the user, cannot be encoded but has to be in a form that is comfortable and familiar to users. GUI forms of presentation are becoming a requirement for warehouse applications. In order to make the data most useful to the end user, it frequently requires the system to perform derivation, aggregation and transformation functions to the raw data.

Database copies that support warehouse applications are usually read only. Updates, as they occur, are performed to the source production system database from which the target warehouse database 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 technology to support data warehousing. DEC, Hewlett Packard and Information Builders are other companies that offer important technology for supporting data warehouse approaches.

At the other end of the replication technology continuum are replication approaches that are designed to replace and improve on distributed on-line or distributed DBMS technologies. These approaches have to offer real time updating against copies of data that may be located in many locations. The basic approach used by replication servers here is to uncouple the distribution of data copies from the originating application. The application is allowed to update its local copy and when that is completed it proceeds to the next transaction. Asynchronously, the replication server, then propagates the changed data to its other locations. This type of replication is appropriate for a production system. It normally requires a single global schema. These systems will be referred to as (TP-R, Transaction Processing - Replication).

TP-R requires a very different technology than warehousing. Production systems need the current state of data, not its history. For efficiency purposes, at input and in processing, the data is frequently encoded. Each production location does not necessarily need access to all of the global data. Subsetting by region, for example, is common. Any node must allow updates to production data. The propagation of update copies to secondary locations should be done as soon as possible. That propagation, then, is done in near real time with a separate 2-phase commit to each target copy location. Such a system can maintain transaction consistency for updates that span multiple tables at one (or more) target sites.

The leaders in TP-R approaches are Sybase and the ASK Group (INGRES Division). Sybase's architecture is built around a master/slave concept. INGRES is based on a peer to peer model. Both of these approaches will be discussed below.

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 considering replication options the user needs to consider requirements for currency, local updates, data enhancement and history maintenance, among other considerations. In the interest of keeping this article of readable length we will concentrate on replication and distributed DBMS issues from the two ends of the continuum scale shown in Figure 1.

Distributed DBMS

It's useful to understand something about distributed DBMS technologies before analyzing replication, because the approaches are very closely related. The concepts behind distributed DBMS were pioneered during the late 1970's 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 DRDA (distributed relational data architecture). DRDA at this time is largely an approach for integrating data sets across the different versions of DB2 that run on AIX, OS/2, OS/400, VM and MVS. DRDA has been published and IBM encourages other DBMS vendors to participate as client or server sites.

The first well-publicized distributed DBMS product was INGRES/Star, announced in 1987. Oracle also announced distributed DBMS capabilities in 1987, but largely as a marketing ploy. The first Oracle product to reasonably support distributed database processing is Oracle 7, which has been in the market since 1993.

A true distributed DBMS, as defined by most industry consultants, requires the system to support updates at any node on the network. A short summary of Chris Date's requirements for the functions that should be supported by a distributed DBMS is provided in Figure 2.

REQUIREMENTS FOR A DISTRIBUTED DBMS

1. LOCAL AUTONOMY; LOCAL DATA ARE MANAGED INDEPENDENTLY OF OTHER SITES
2. LOCATION INDEPENDENCE: USERS AND PROGRAMS DON'T NEED TO KNOW THE LOCATION OR PATH TO THE DATA
3. NO CENTRAL SITE: NO DBMS SITE IS MORE IMPORTANT THAN ANOTHER
4. CONTINUOUS OPERATION: NO PLANNED ACTIVITY SHOULD REQUIRE A SHUT DOWN
5. FRAGMENTATION INDEPENDENCE: A TABLE THAT HAS BEEN FRAGMENTED WILL APPEAR AS A SINGLE TABLE TO USERS
6. REPLICATION INDEPENDENCE: REDUNDANT DATA IS MANAGED, ACCESSED & UPDATED TRANSPARENTLY. FAILOVER RECONSTRUCTION
7. DISTRIBUTED QUERY OPTIMIZATION & PERFORMANCE INDEPENDENCE
8. DISTRIBUTED TRANSACTION MANAGEMENT: INTERLEAVED TRANSACTIONS THAT UPDATE MULTIPLE SITES RUN WITH CONCURRENCY CONTROL AND RECOVERY CONTROL IF THERE'S A FAILURE.
9. INDEPENDENCE FROM: HARDWARE, O/S, NETWORK, DBMS
10. THERE'S DISTRIBUTED ACCESS TO THE DATA DICTIONARY

Figure 2 - Chris Date's requirements & functions of a distributed DBMS

TP-R as Compared with Distributed DBMS

If one compares TP-R to distributed DBMS, the main difference is in the relationship of the application to the various distributed updates. With distributed DBMS there is a single 2-phase commit that synchronously locks all of the data copies until all locations respond with a "committed" message. With TP-R replication this is replaced by "n" separate two phase commits, where "n" is the number of separate data locations.

Replicated database nodes are less synchronized than data copies maintained by a distributed DBMS, of course. Conversely, they offer faster overall system processing, faster local commits of transactions and the potential for significantly reduced network traffic. All of this, however, is at the risk of collisions between different data servers.
A collision is when different physical copies are updated by different applications during the same interval of time. A DBMS replication server needs to provide software to aid in the resolution of such collisions. Software can detect a collision and provide notification. It also can follow any business rules that have been set up to resolve such an occurrence. Collision resolution is discussed later in this article.

The replication approach is more fault tolerant than distributed database and therefore more appropriate for many applications. In a replication approach the timing between the changes at the different nodes is managed through mail or store and forward approaches rather than through locked multi-site transactions. Once the application updates its local data, it is de coupled from the replication engine which has the responsibility for propagating the copies of the changed data to other locations. 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).

Replication cannot be used where absolute data synchronization is required for the application. Examples of such applications would be financial trading and banking funds transfer. Other applications such as order processing, and hotel or airline reservations might be handled with replication approaches. After all, airlines and hotels overbook intentionally. If the application can deal with some inconsistency among the different data nodes for short periods of time, then replication should be considered as an alternative.

The Crucial Role of a 2 Phase Commit

Definition

The essence (and the bane) of distributed database is the 2-phase commit. What the 2-phase commit accomplishes is a synchronized locking of all pieces of a transaction. The result, then is an atomic action when the transaction is spread over multiple locations and processors. A 2-phase commit allows a distributed transaction to be processed with the same data integrity as a transaction that is processed entirely within a single computer database.

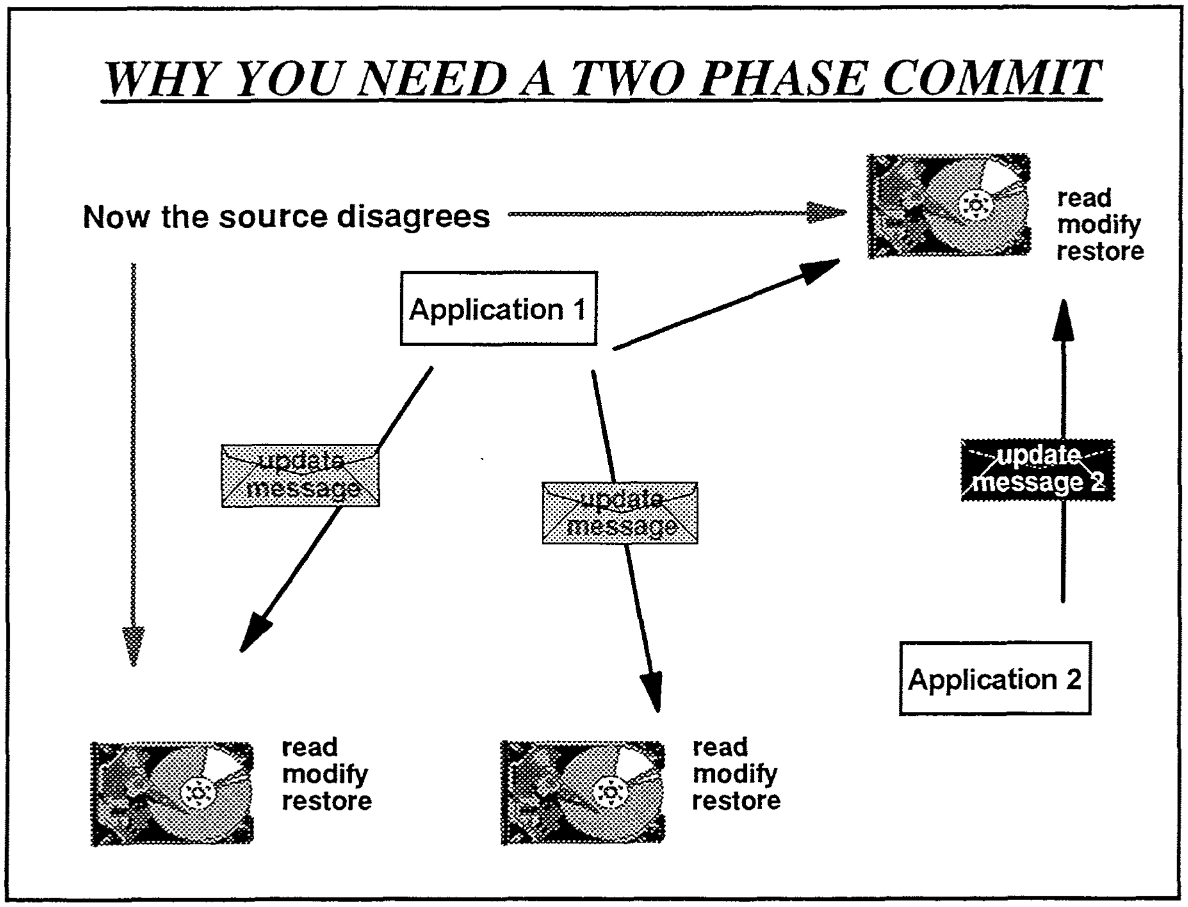
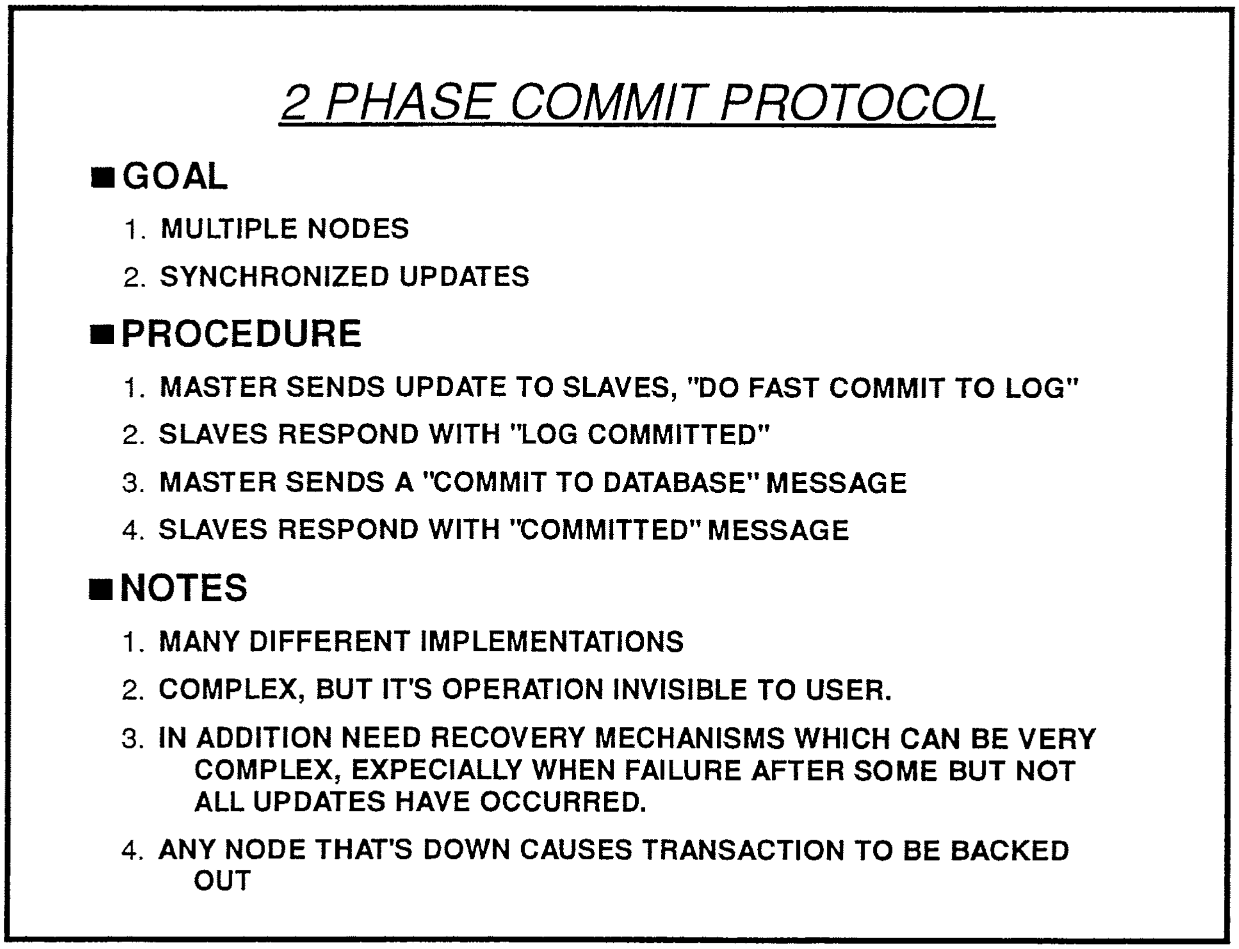


Figure 3 - An example for the 2-phase commit

In Figure 3, an example is provided showing the need for a technology like 2-phase commit. Things start off when application 1 updates the lower left database. It does that by reading the before image of the data to be modified, changing it and holding locks on the data, plus preparing a log entry in the lower left database. That application then goes on to successfully accomplish the same process with the center lower database. But as application 1 tries to complete its updates by updating the database at the top right, it finds out that another application (2) has already modified part of the data that is to be updated by application 1. In other words the data read now doesn't agree with the values that were read in the first two updates. This is a "collision" and the end result for a distributed DBMS is that the first two pending updates have failed, the locks are released and the transaction is rejected.

Figure 4 details the procedure followed by a distributed DBMS in a 2-phase commit. When a synchronized 2-phase commit is combined with data locking, logging and recovery, the necessary ingredients for building a distributed database with absolute data synchronization are in place. However, because any failure in the network or any of the local participating databases causes the entire transaction to fail, this approach to distributed computing is very intolerant to failure.

Because of this intolerance, distributed DBMS are 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.



2 PHASE COMMIT PROTOCOL

■ **GOAL**

1. **MULTIPLE NODES**
2. **SYNCHRONIZED UPDATES**

■ **PROCEDURE**

1. **MASTER SENDS UPDATE TO SLAVES, "DO FAST COMMIT TO LOG"**
2. **SLAVES RESPOND WITH "LOG COMMITTED"**
3. **MASTER SENDS A "COMMIT TO DATABASE" MESSAGE**
4. **SLAVES RESPOND WITH "COMMITTED" MESSAGE**

■ **NOTES**

1. **MANY DIFFERENT IMPLEMENTATIONS**
2. **COMPLEX, BUT IT'S OPERATION INVISIBLE TO USER.**
3. **IN ADDITION NEED RECOVERY MECHANISMS WHICH CAN BE VERY COMPLEX, EXPECIALLY WHEN FAILURE AFTER SOME BUT NOT ALL UPDATES HAVE OCCURRED.**
4. **ANY NODE THAT'S DOWN CAUSES TRANSACTION TO BE BACKED OUT**

Figure 4 - A 2-phase commit approach

All modern distributed DBMS products offer methods for implementing a 2-phase commit. However, the degree of automation support is different from different vendors. For example, IBM, Oracle and INGRES offer high level (transparent to the application) approaches to implementing 2-phase commit. The Sybase replicator takes more programming to implement in that it requires the user to handle some of the "handshaking" issues, by, for example, coding DBLib or RPC calls into the application. If the application environment requires transaction rollback from time to time, this additional programming can be difficult to handle properly.

Distributed DBMS 2-phase commit procedures and implementations are proprietary as there is no standard established for how it should work. 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. IBM and Sybase have published their proprietary protocols and procedures for 2-phase commit. Some other vendors, like INGRES and Digital, have taken advantage of this information by including support for a 2-phase commit process to extend from their own systems to those of IBM and Sybase also.

Replication and 2-Phase Commit
Replication offers an asynchronous approach to updating copies. Asynchronous as defined above means that the distributing of the updates to secondary sites has been uncoupled from the primary update. The process which transmits the updates has to be reliable (insuring that the copies get to the targets) and valid (insuring that the necessary integrity is maintained at the target). DSS-R and TP-R approaches can use the same approach for reliability, but typically use different technologies for validity/integrity.

In both approaches, the process for insuring that no copy information is lost is to use a 2-phase commit protocol for the changed data transmission.

In TP-R environments the integrity of data at the target site must be maintained by applying copy updates one transaction at a time. The changed data from one user transaction can span multiple tables at each update location. At each site, then, all or none of the updates should be applied. This way the data stays consistent across all tables at all times.

In contrast, DSS-R approaches apply updates table by table. All tables that may have been affected by one transaction aren't 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.

In both replication approaches, unlike for a distributed DBMS, it is <u>not</u> necessary for the 2 phase commits that distribute the copies to be part of the original (application driven) transaction. An example of a TP-R implementation approach for this is:

1. The original (application driven) transaction performs a local DBMS transaction with a normal commit. As part of this local transaction the distribution queues and replication logs are updated with a record of the transaction. Once this is complete the application can continue to process other transactions.
2. In near real time fashion the replication server will be notified by the local DBMS that there is a transaction waiting to replicate. The server examines the distribution queues and then schedules multiple sub-transactions to update the target databases. These databases are, typically, remote and therefore the replication server uses a separate 2-phase commit protocol when moving transactions from the distribution queue to each individual target database.
3. For any target databases that are not on-line or available, distribution transactions will remain in the distribution queue until a time when the targets become available. The other (available) target databases can go ahead and

synchronize with the source and, of course, the originating application is not affected by these (behind the scenes) DBMS activities.

It is interesting to consider the above scenario and how it compares to a true distributed DBMS solution based on synchronous updating approaches. The first key point is that the application isn't blocked by a problem that is related to distributed transactions. It performs a local update, which is quick, while the DBMS manages the distribution asynchronously.

The second key point, of course, is that there is a latency between the updates performed at the primary and subsequent copies. This raises application issues which the user needs to be able to live with. These points will be discussed below under the topic of collisions.

A third key point is that in larger and/or less reliable environments a distributed DBMS approach just wouldn't work, while replication's architecture can. Imagine a situation with 100 target database nodes, only 90 of which are available. A replication server would perform 100 separate 2 phase commit transactions, each with two branches. A distributed DBMS would attempt to perform one 2 phase commit transaction with 100 branches. 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 an unreliable WAN scenario (e.g., developing countries) or any situation with many nodes, clearly the distributed DBMS solution just flat doesn't work.

If you use a replication server to support operational systems, the application view of data at the different locations should be 1) each logically consistent within self, but 2) possibly out of phase with each other for some period of time. The differences in data among different nodes are all transitory and get reconciled over time.

Introduction to DSS-R Approaches and Table Copying

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.

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 for updates). Consistent, stable data for a given period is the highest requirement for these types of applications. The decision support systems are tuned for query processing, typically by adding more indexes. In this case, then, continuous propagation of updates would interfere with the ability of the query tool to provide reasonable performance (above and beyond the additional load that is created on the replication server).

The replication server should provide various timing options which can create copies based on timed events (clock or interval), on application events (e.g. end of day reconciliation completed), or on manual request.

Other important requirements for decision support include the ability to access legacy production system data from sources such as IMS, RMS, VSAM and flat files and to provide sophisticated data manipulation/enhancement to that data.

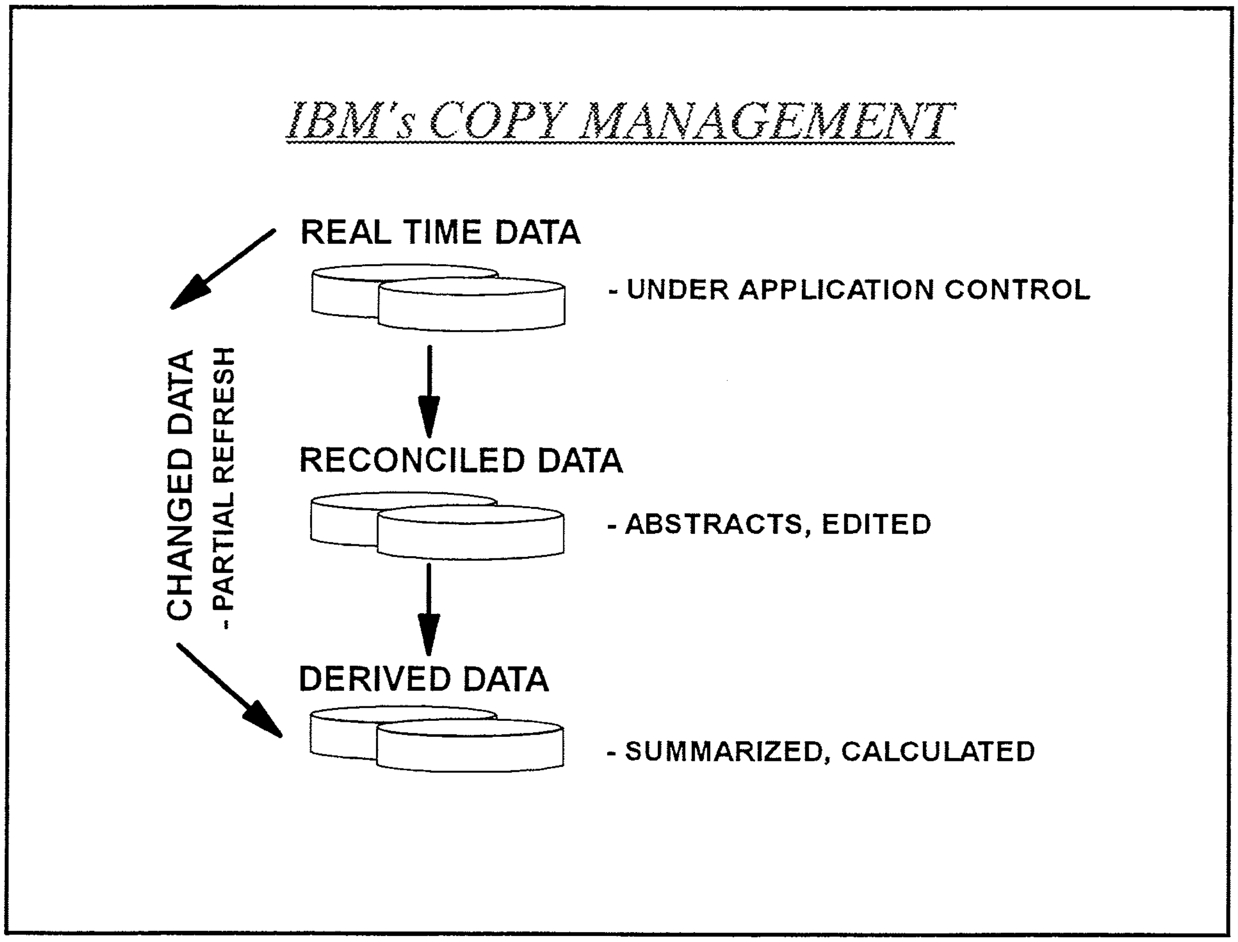


Figure 5 - IBM's Architecture for an Information Warehouse

An example of data enhancement is what IBM has implemented in its Information Warehouse - a sort of three schema architecture for decision support purposes. Recognizing that operational systems frequently aren't correctly structured for supporting queries, IBM offers reconciled copies and derived data which summarize and add calculation value to the copies of data offered for decision support. The copies can be updated at any time and according to criteria established by the d.b.a.

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 which 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 insure consistent transaction data is copied when the source table(s) are not being updated.

Alternatively, DSS-R may be provided through propagation of source table changes to the target. In large database environments (multiple 100s of gigabytes) where full refresh table copy transmission is economically or technically unfeasible on a nightly basis, change propagation is the only solution. In order to insure that consistent data is propagated in this scenario, a 2-phase commit process should be used for the changed data transactions.

DSS-R Schema

The value added to the data by manipulation or enhancement is very important in DSS-R environments. Sources are typically legacy systems and the replication solution should provide the ability to restructure the data from legacy formats into the relational model. Tools should provide support for JOINing data from multiple sources, for calculating new values, for aggregating data and for transforming encoded data into descriptive forms. (See Figure 5). An important side point to keep in mind is that one of the principal benefits of DSS-R, aggregation of data or de normalization, is something that should not be done when the replicate is updatable. This will be discussed further below under TP-R Replication Schema.

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.

Cascading Replicates

A common application model is one where there are 100's or 1,000's of database servers (e.g. in branch offices) fed from a smaller number of distribution nodes, which are in turn fed from a central host source. Efficient distribution requires support for cascading replicates where copies can be made from other copies. For example, the central host distributes to 20 distribution nodes each of which distributes to 20 branch offices. The replication system must distribute consistent data across each of the 400 branches (perhaps at the end of the business day), but at the same time, subset the data for branch related operations.

Optimization - Push & Pull

Where change capture and propagation schemes are used, there is a choice in the distribution model: whether to "push" the changes from the source to the target system(s) as they occur, or whether to "pull" the changes from the source system as the target(s) request. In general, the push model is best for continuous almost real time propagation, whereas the pull model is best for more loose currency requirements. This is because the pull model provides greater flexibility in reducing/combining the data at the source site. The pull model also allows more control and flexibility in the timing of network traffic.

For example, push systems typically distribute every transaction to the target. Target systems must therefore process every transaction. Where only summary data is required, data transformation is an added cost after replication. Pull systems, however, provide the opportunity for aggregation prior to distribution. This is effective both where only summary data is required and where database hot spots (areas within the database which receive the most update activity) can be netted out.

If you are distributing production operational systems, DSS-R technology isn't likely to work for you and a TP-R approach which can maintain near real time transaction integrity at data copy sites is essential. On the other hand, for decision support or other static data the need for real time information may not be important. Here, multiple schema's or data views, efficiency for size or cost reasons, and a consistent stable database for a specific period of time, will argue for copies of an end of period database.

TP-R Replication: Peer to Peer & Master/Slave Approaches

Although many DBMS vendors are talking about replication offerings, it would be a mistake to assume that replication is a commodity. Different architectural approaches to the implementation of replication provide fundamentally different capabilities. Not only are there important replication server differences between DSS-R and TP-R approaches, but within each of these architectures there are important differences.

TP-R approaches have been implemented with two fundamentally different architectures by ASK/INGRES and Sybase. ASK/INGRES has built its replicator on a peer to peer architecture approach. Sybase uses a master/slave approach.

TP-R replication is primarily concerned with creating a single image of a database across distributed autonomous sites and preserving database integrity in near real time processing. The overall integrity of databases is preserved by forwarding data changes resulting from single user transactions.

All data replication, regardless of vendor, copies data from sources to targets. Master/slave approaches replicate data from master to slave, requiring updates to successfully complete at the master before the transaction is considered a success (as far as the application goes). 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.

In the master/slave architecture every table or table fragment is assigned to a primary site. If the primary table's database server fails or access to that server from the network (where a transaction updating that table has occurred) is denied, replication doesn't occur and the transaction is queued. This can present a problem for remotely generated transactions because those processes cannot update their local, or other sites, until they are first routed synchronously through their primary tables.

The master/slave approach to TP-R has the following characteristics:

- It's simpler for a vendor to implement (from the replication server point of view) because it eliminates the potential problem of update collisions (explained below).
- Because its implementation is simpler and more straightforward than peer to peer, in some circumstances applications will run faster because of lower DBMS overhead.
- It introduces a single point of failure that can lower the overall system availability as compared with the peer to peer approach.
- It's a less general solution than peer to peer.

Although the Sybase architecture is master/slave, the vendor states that its Replication Server can be set up to support a peer to peer approach. As is discussed below, collision detection and resolution software should be provided by any system that supports peer to peer transaction replication. Sybase normally requires that updates to slave databases be first routed through the master database. This eliminates the need for collision detection and resolution. However, if you want to build a peer to peer architecture with Sybase technology you'll have to write your own 1) collision identification software, 2) collision resolution logic and 3) logging transfer manager (including recovery). This would be work well beyond the capabilities of the typical DP shop.

The peer to peer architecture, of which INGRES is the only vendor at this point in time, is the most general and 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. And yet, because 1) we're talking about a replication server which uses many individual 2-phase commits to broadcast data changes and 2) those changes are asynchronously distributed from the originating application, peer to peer is more fault tolerant than a distributed DBMS.

A problem that is related to use of a peer to peer replication approach, however, is the possibility of "collisions". Collision occur when two different originating nodes update two different physical copies of the same logical data with two different transactions. When the replication server attempts to broadcast changes from each of those originating sites it will become aware of this conflict in updates and need to begin a process of reconciling the differences.

Collisions with a Peer to Peer Architecture

A collision is when the same record, which is physically replicated at two or multiple sites, is updated during the asynchronous latency period. In other words, after the time a first update has happened, a second update occurs which is processed at one site before the propagation of the first update has been completed. So although 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. First and most important, collision resolution requires that the system provide notification that a collision has occurred.

From the moment any transaction is committed, the replication server has to keep track of all of the processes that further happen in the processing and distribution of that transaction. That's because in the event of a collision, this information has to be available to properly resolve the collision.

The replication server should support multiple options for the d.b.a. to choose from in resolving the conflict. Examples of resolution possibilities include:
1. The initial update has priority. Rollback the conflicting (and later) transaction with necessary messages to designated parties.
2. The last update has priority. Overwrite the conflict and send the necessary notices.
3. Resolve the conflict by firing a user specified trigger.
4. Halt the replication process and send a message to the d.b.a..

In order for a number of these processes to work it's helpful is there is a distributed time service available because current replication servers don't provide this. The replication server depends on the separate operating system clocks. If they aren't 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 your databases have been properly designed for replication, the volume of collisions is likely to be very low. Those conflicts that do occur can be handled 1) by rules in a collision resolution software module with log entries for manual review, or 2) by manual review. Future capabilities for replication servers in this area may include expert systems to help resolve collisions.

Collisions don't happen with a master/slave architecture such as Sybase's. This is because the transaction is simply not accepted unless it can be committed at the master site, or what Sybase calls a "clearing house".

It might be useful to refer back to Figure 3 and re analyze what would have happened had peer to peer replication been used. In that case, the application, would have been accepted and considered successful at the completion of its first database update. That's a powerful performance advantage. Later, however, further processing on the network resulted in a collision. Some further processing and/or manual involvement, then, will be required to recover the multiple database copies in a consistent way.

TP-R and Fault Tolerance

One of the principal benefits 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 d.b.a., should provide as much assistance as possible in the recovery process. (Mike Stonebraker has used the phrase "failover reconstruction" to describe when this recovery process occurs automatically under software control). Necessary steps in the failover reconstruction process should include:

1. understanding what is broken
2. understanding what or how the break occurred
3. determining how to fix the damage and reinstate the broken pieces
4. bringing the broken pieces back on-line
5. making sure that the recovery of the database(s) results in consistent data in those database(s)

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 becomes updated. After the master has been updated and when there is some failure elsewhere, the replication server queues the updates to the slaves until they are available. This system works as well as a peer to peer approach unless it's the master node or network that fails.

In either 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 key utility should be able to "difference" replicates - in other words to look at a master and slave or two peers and determine if inconsistencies exist.

Transparency & 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 to customers. There is a significant difference in the amount of replication function provided by various DBMS vendors and in the ease of implementing replication and its various features. 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 requires programming with RPC's or DBLib calls by the distributed data base administrator (d.d.b.a.). Setting up database replication with INGRES is easier in that a configuration manager is provided that offers a three step forms based approach to defining the replicated environment.

The TP-R Schema
In order to provide transparent replication services to applications, the d.d.b.a. needs to be very much aware of the use of a replication server and needs to have designed the database in a manner that is conducive to distributed operation. In practice this issue means that de-normalized and/or aggregated data should **not** be replicated in TP-R situations. Such derived/aggregated data should be computed at each site from the basic data contained in a transaction.

To see this point more clearly the banking example below may help. It illustrates a process that spans three periods of time (A, B, C) and three branches of a bank (1, 2, 3).

	Bank 1			Bank 2		Bank 3	
Time	Balance	Transact		Balance	Transact	Balance	Transact
A	100			100		100	
B	100		X	60	-40	60	-40
C	70	-30	X	60	-40	60	-40

We're looking at one customer's balances after withdrawals are made during a period of time when the network to one replicated site is down.

- At time A, the network is entirely up and the customer's balance (100) and current transaction (none) are identical at all three bank sites.
- At time B, the network link to 1 is broken. The customer makes a withdrawal at bank 2. That transaction is replicated into Bank 3 and the balance from 2 is also

replicated into 3. Bank 1 still has the old information, since access to it is unavailable.

- At time C, the customer makes a withdrawal at Bank 1.

At any time after this an attempt to reconcile the balances among the three banks is going to fail. That's because the account balance field in this example is aggregated (and denormalized). Replicating balance information is going to cause integrity problems with the data bases.

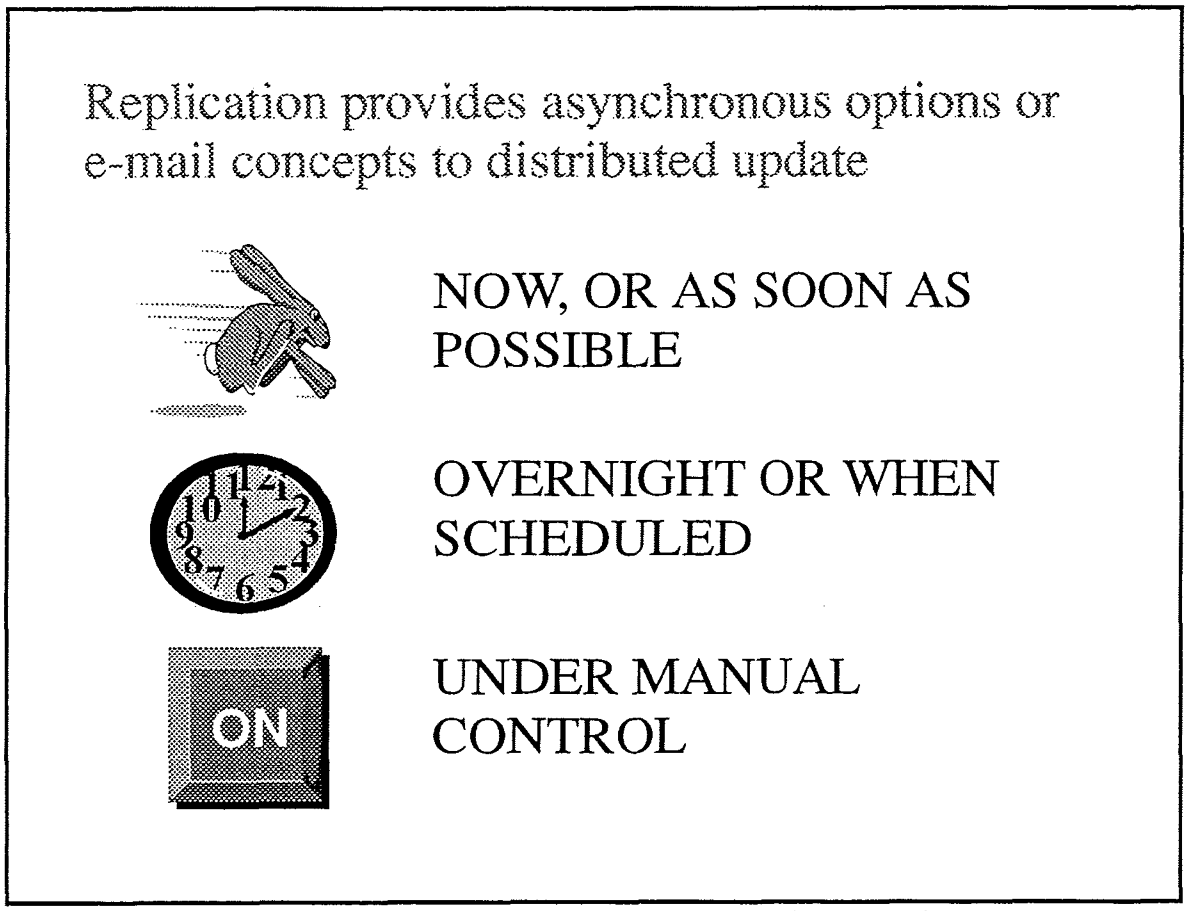
Repeating, then, in the TP-R environment an important rule for replicating data is to not replicate aggregated or denormalized data. If the system had simply replicated the transaction amounts, normalized data, each site would be able to recover correctly from a collision like the one illustrated by using a time order to sequence and process (and compute the balances). In general, a good rule for distributed processing is to use local database triggers to handle computed amounts such as account balances.

Replication Timing

Your application shouldn't need to worry about the timing of the asynchronous distribution of data to target sites. Getting this functionality from your replication server also shouldn't require you to do programming.

The replication server, be it TP-R or DSS-R, should also provide several alternatives for timing. Examples are:

1. immediately, as soon as possible (ASAP). In this case the data is moved through the queues and replication server as fast as possible.
2. scheduled, as determined by the system administrator. In this case, data remains in the replication server until it is scheduled for distribution.
3. triggered, by user defined criteria such as an event happening, the number of records exceeding a limit or time of day. When that trigger is fired, the server moves the data to the distribution queue for remote processing.
4. under manual control



Replication provides asynchronous options or e-mail concepts to distributed update

NOW, OR AS SOON AS POSSIBLE

OVERNIGHT OR WHEN SCHEDULED

UNDER MANUAL CONTROL

Figure 5 - Replication offers various timing options

The nature of the system usage will dictate the type of timing used in replication. For operational systems that expect to be updated with near real time 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 (to preserve transaction integrity). In such a case, then, there is no processing savings attached to batching the transactions (although transmission at night might offer savings).

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 replication may be preferable.

Database Configuration & d.b.a. Utilities

Managing a distributed database is significantly more complicated than running against a monolithic single location database. The distributed d.b.a. has all of the design and implementation issues of a single location *plus* the added complexity of distribution, network latency, time shifts and remote administration.

The distributed d.b.a. (d.d.b.a.)is a new job function in addition to local d.b.a.'s. The following are examples of work the d.d.b.a. will perform:

- Designing and planning the replication system, including how and when data is shared amongst users. It's only after this work has been done that the local d.b.a. can input the necessary information to set up the replication system.
- Coordinating the installation and system configuration amongst its various sites.
- Monitoring the operation, performance and recovery of the system from an enterprise, rather than a local, perspective.

Some ideas to remember as you consider implementing a replicated database environment are:

I. Set up a plan and understand the rules for distribution of data before the implementation begins. Implementing replicated databases is not technology amenable to "let's try it and push it around a bit" approaches. It's necessary to have a good plan in hand before you begin or you will get lost in the middle of building the replicated environment. If your plan is good, the implementation can proceed in incremental fashion, however.

II. Make sure that your d.d.b.a. has good forms based or graphical utilities to assist in the database configuration and in the management of the ongoing network. For example, INGRES comes with forms based management utilities and IBM and Sybase have GUI based management utilities. These facilities should be able to manage all aspects of a replication environment from a single desktop that's moveable and can be anywhere on the network. Some points to carefully consider:

   A. How do you specify enhancements to the data? Do you have to learn a new language for this function?
   B. How is the replication setup handled? How much automated support is provided to the d.d.b.a.?
   C. What is the support provided for failure management? How much recovery is automatically handled and how much d.b.a. intervention is required?

III. Your utilities should be able to answer questions like:

   1. What tables are at what nodes?
   2. What columns are at what locations?
   3. What rows are at what locations?
   4. Where are transactions routed to?

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

V. There should be a mail based error notification system. This allows management of the distributed enterprise from any node on the network.

Replication into Heterogeneous DBMS

Today, there are no standards that apply to replication across diverse products. And there are no standards bodies working on this issue. Issues like utilities and recovery are just handled quite differently in different vendor's products.

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

Both Sybase and INGRES have links **to** non-relational DBMS in their target replication capability. Normally if the vendor supports a gateway to that DBMS, then it can serve as a target for replication. That includes IMS, RMS, VSAM and other environments for both of these vendors. The gateways to non-relational DBMS don't require special coding (such as RPC's) and are valuable in allowing the integration of new distributed systems with older applications.

As a general rule, replication **from** a foreign DBMS into a replication environment such as INGRES or Sybase is only available now if the user is willing to program that functionality. One important exception is an IBM offering which 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 and 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 DBMS. The real world is multi-vendor, multi-department and multi-network. Replication technology that can operate well across heterogeneous DBMS is something that DBMS users will want.

Summary of Replication Benefits

Better Response Time from Local Data

1. A replication server can be instrumental in allowing more efficient usage of a company's computers and network. By shifting data to the local site where it's needed, companies can insure that important applications are available at all times. The response time achievable from local data access can be significantly improved over response that depends on access from a distance. Also, replication is more fault tolerant than distributed DBMS. That fault tolerance results in more consistent processing of transactions with the result that the overall database is up and responsive more than the equivalent configuration would provide if it were a distributed DBMS.

Replication for Hot Standby Backup

2. Replication can provide the architecture for backup that can enhance your system reliability in a local (and/or WAN) environment. Replication, enhanced with hot-standby software, operates by monitoring the performance health of your primary server, while transactions are backed up on the replication server.

When there's a failure on the primary processor the backup is immediately available. The system automatically switches to the backup and designates another machine as the new backup replicate.

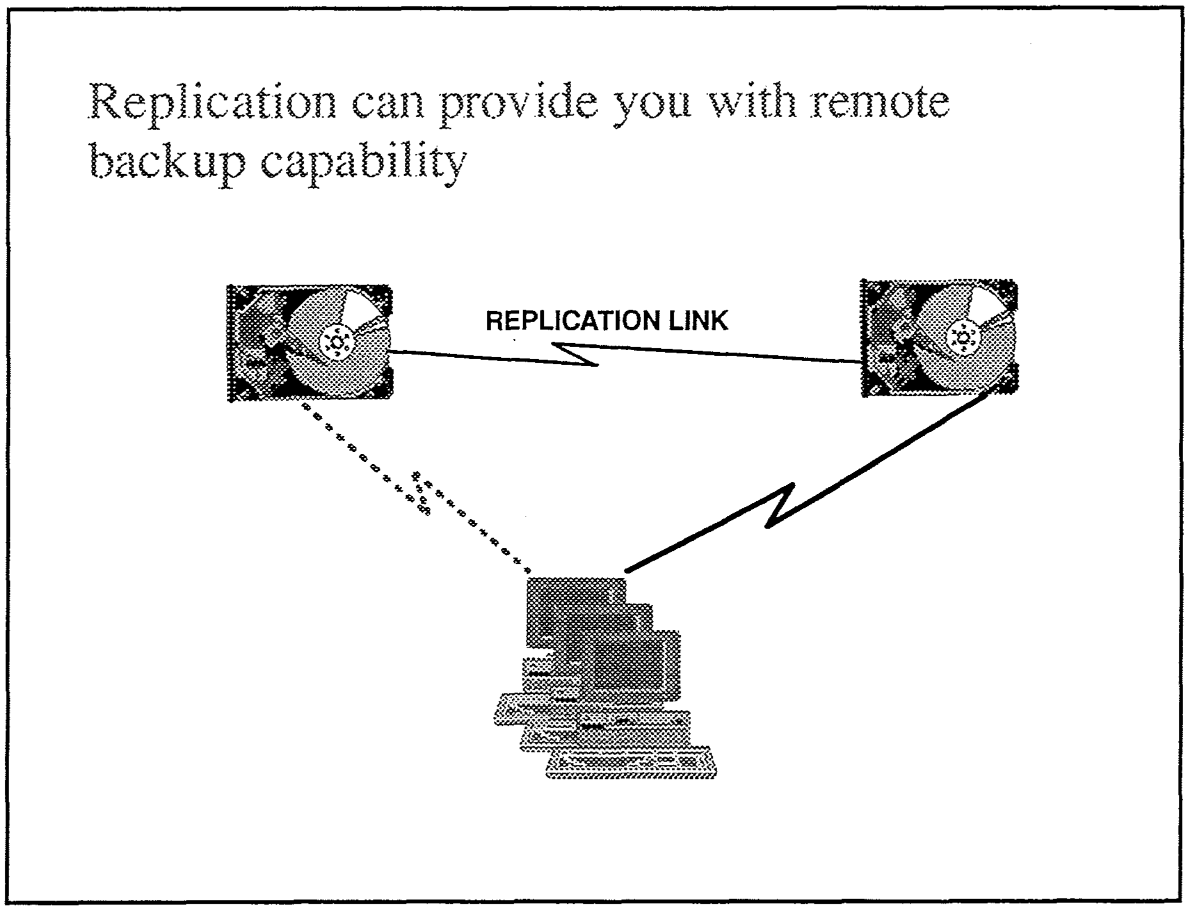


Figure 7 - Replication can provide backup to enhance your system reliability

Data Availability Such as Separate Servers for Separate Functions

3. Individual workgroups can now have their own replicated databases. This means not ever having to say "sorry" for network propagation delays. Replication can enhance performance and provide load balancing locally or over a WAN. As an example of this, two replicate servers could allow queries to be channeled to one machine while updates and production work are channeled to the other. The query server will have accurate information that is exactly current or somewhat dated, depending on the speed of replication chosen by the user. With DSS-R approaches the database copies can be enhanced for decision support. Data can also be replicated from legacy applications and made available now to new styles of processing across the network.

Replication can reduce network traffic, provide better local response, lessen host processing

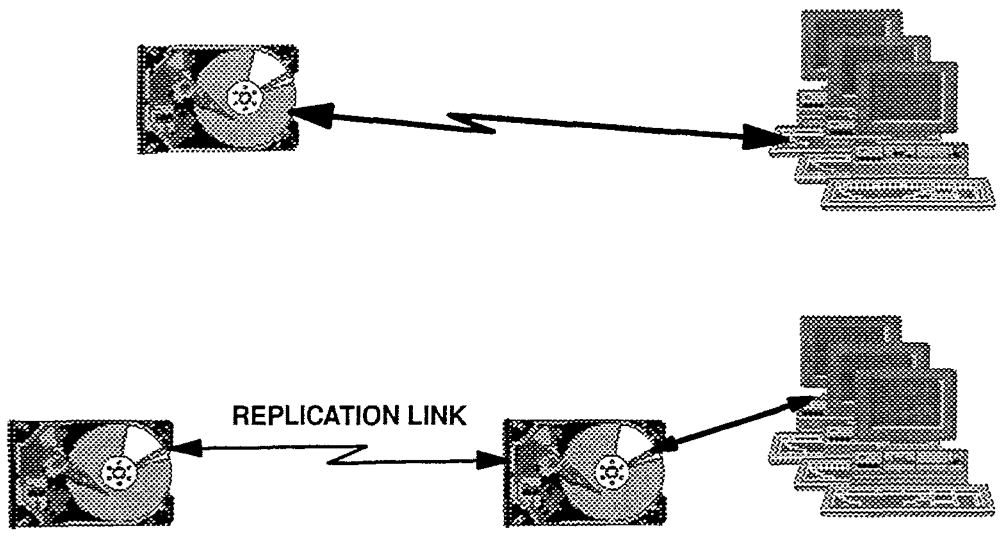


Figure 8 - Replication can enhance performance and provide load balancing

Decision support types of applications are natural replication candidates, because if they're distributed, replication can greatly reduce WAN traffic.

Splitting the Workload for Capacity Relief

4. As companies migrate to decentralized operations, they naturally want their computing support to follow the same form. As the workload is distributed, it is split among multiple servers. 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, then, allows:
    a. a decrease in response times
    b. a reduction in wide area network traffic
    c. the establishment of local autonomy which can take over in case of network or server failure. A key to achieving this advantage is to use a peer to peer type of replication service. This is so that when recovery occurs the completed local updates can be properly propagated to other locations of the same data.

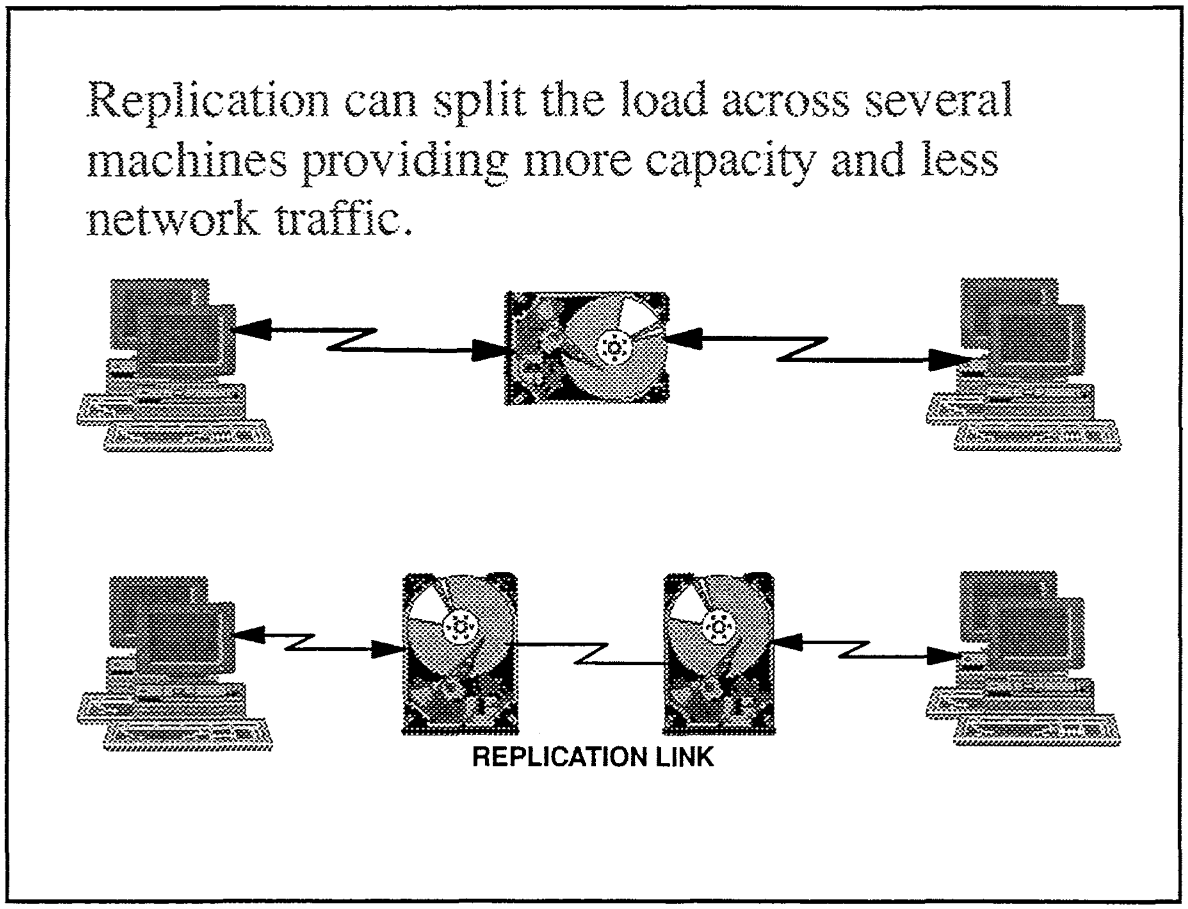

Replication can split the load across several machines providing more capacity and less network traffic.

Figure 9 - Split the database server load across several machines

Non-Stop Processing & System Fault Tolerance

5. Replication is an important technique for increasing the availability or uptime of network based computing. Redundancy is the fundamental engineering approach for increasing reliability and replication can be used exactly for this purpose.

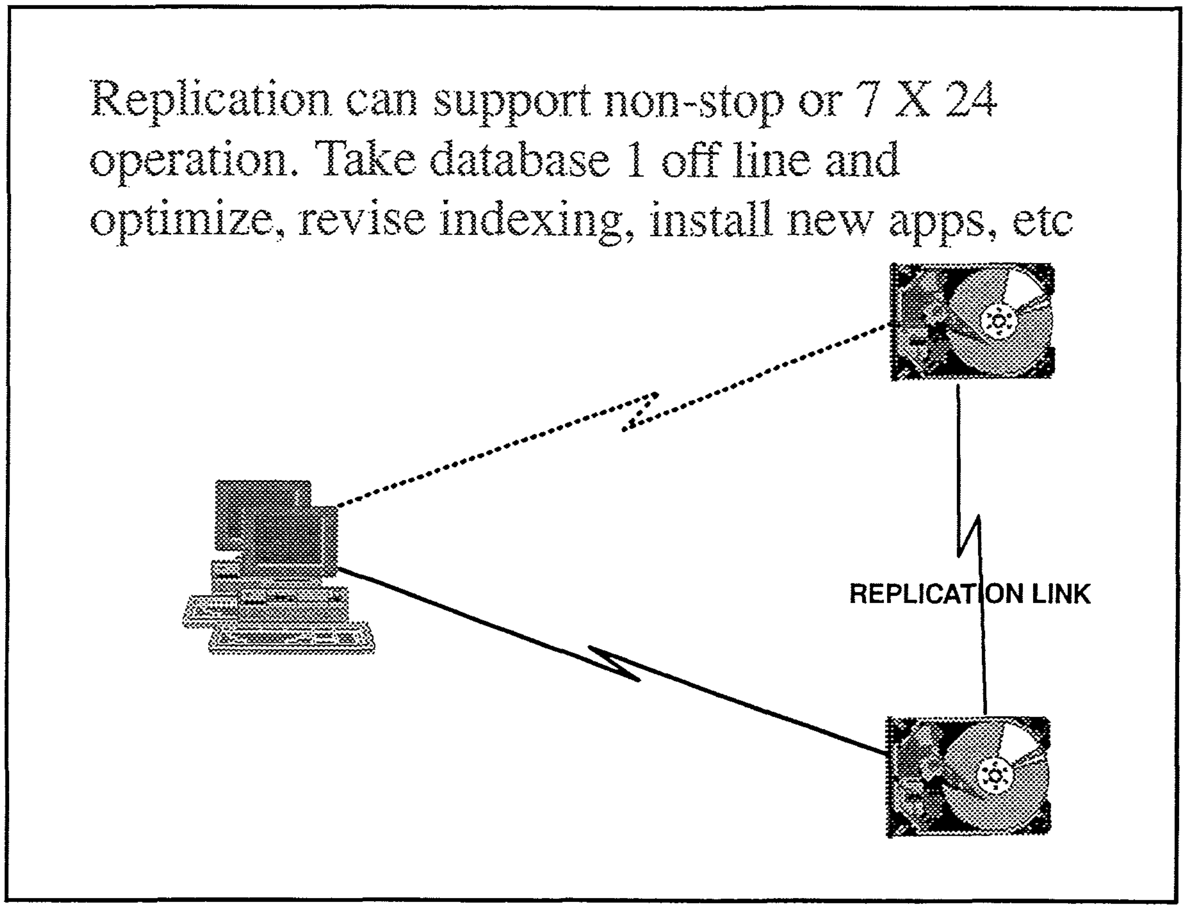

Replication can support non-stop or 7 X 24 operation. Take database 1 off line and optimize, revise indexing, install new apps, etc
REPLICATION LINK

Figure 10 - Replication is a technique for increasing system availability or uptime

Imagine a retail operation where sales offices are widely distributed and inventory is kept at a few major warehouse locations. If the warehouse 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 warehouse 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.

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

Conclusion

It's a Complex Environment

The benefits of a properly implemented replication scheme can be very substantial. The complexity however, in both a managerial and technical sense, of a distributed environment 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 the cooperation of excellent software and competent administration.

Your Database Administrator is a Key Resource

It's wise to invest the necessary resources to make sure that the combination of local and global d.b.a. resources is adequate for your environment. Your d.b.a. will have to create a data base design that is correct for replication and tested in the distributed environment. In an operational sense it's important to not shortchange the time it takes for your d.b.a. to become an expert in diagnosing and resolving problems in this environment. You should seriously consider consultant assistance, probably from your DBMS vendor, as part of the first project.

Your Approach Should be Cost and Benefit Based

Make sure that you understand the architectural, currency, data integrity, and performance implications of a DSS-R or TP-R based approaches. Different approaches from within any one vendor's product line and/or between vendors mean that different technologies have very different cost, performance and integrity results. You should have a DBMS that supports the different requirements of your application environment.

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

Because implementing distributed systems offers so many combinations of technology and benefit you'll need to do some careful management analysis to understand how these approaches can support your business requirements. Those business benefits should be measured against the costs of the software and management necessary.

Keep it Simple, Especially at First

It's wise to begin implementing a distributed database with a single vendor. However, If you have a heterogeneous DBMS environment, be sure to understand how your vendor can support a multiple DBMS approach.