VERSANT Replication: Supporting Fault-Tolerant Object Databases

Yuh-Ming Shyy, H. Stephen Au-Yeung, C.P. Chou
Versant Object Technology
1380 Willow Rd., Menlo Park, CA 94025
email: yms@versant.com

VERSANT is the industry's leading object database management system (ODBMS) for developing applications in multi-user, distributed environments. VERSANT ODBMS has an object-based client-server architecture which is particularly suitable for such complex applications as telecommunications, transportations, and utilities network management systems. Because these applications are usually mission-critical and do not allow any database down time, one major requirement of our customers is 24x7 (24 hours a day, 7 days a week) *high-availability* of object databases, even in the presence of software, hardware, or network failures. While many VERSANT features, such as on-line backup and dynamic schema evolution, have already supported part of this requirement, they do not address the failure cases. Traditional asynchronous replication (e.g., Sybase Replication Server, DEC Data Distributor, and Oracle Symmetric Replication) is not suitable for the following two reasons. First, data integrity may be lost as a result of either delay during propagation or conflicting update requests from different replicates. Second, any failure in the local replica database or central primary database is not transparent to the applications. Special hardware (e.g., mirrored disks) can help but is expensive and it still needs fault-tolerant DBMS support to be transparent to applications. In VERSANT Replication, we take a synchronous database replication approach to support fault-tolerant object databases transparently. As each database can be paired with a hot stand-by replica database located anywhere in the network, VERSANT Replication protects critical data from geological/environment/physical disasters and provides for continuous operation of databases in a distributed environment even in the presence of various failure scenarios.

To use VERSANT Replication, one just has to specify a replica pair "primary_db@node1 replica_db@node2", and use the "createreplica" utility to create the replica database "replica_db" at node2 from an existing database "primary_db" at node1. Once the replica pair has been established, the two databases are maintained *in-sync* automatically, and any failure of either of the two databases is totally transparent to the applications. A pair of replicated databases (X, X') is said to be in-sync if for every object in database X, there is an identical object (with the same object identifier and attribute values) in database X' and vise versa.

VERSANT Replication runs on client-server model. Each application is linked with VERSANT Object Manager and running as a front-end process (client). For every database connected to an application, there is a VERSANT Storage Manager running at the machine where the database is located as a back-end process (server). The internal communication between the client process and server process is done through a network layer by using RPCs (Remote Procedure Calls). Each active database has a process called cleanbe which wakes up periodically to do cleanup work whenever needed, and a shared memory accessible by all the back-end processes of the database. The replica-pair information is stored in a file under the VERSANT root directory where VERSANT ODBMS is installed. When both of a replica pair are available, normal replication is performed as follows.

Database connection. To use VERSANT ODBMS, an application must begin a *session* by connecting to a database (called session database to distinguish from other connections). During a session, an application can make connections to additional databases. The connection request is sent to a daemon process on the machine where the database is located. The daemon process then forks a back-end server process and establishes the connection between the client and server processes. When an application connects to either one of a replica pair, VERSANT Object Manager scans the replica file to find the db@node information of the other database and connects to that database as well automatically. The replica-pair information is then kept in the front-end process memory for fast access.

Object operations. Each object in a VERSANT database is given an immutable and location-independent logical object identifier (LOID) which is unique across the network. All the read, write, delete operations and locking are performed at the object level. Objects requested by the application are read from only one of the replica pair. Object creation, update, deletion, and locking are applied to both databases in the order of primary database followed by replica database to avoid a deadlock situation. We have also extended the standard two-phase commit protocol to send all the transaction begin, commit, and rollback messages to both of a replica pair automatically. If the session database is replicated, then both databases are used to store *coordinator* information for commit/rollback while all the other connected databases are merely *participants*.

When any one of the two databases is down, VERSANT Object Manager automatically switches to the up database (the replica of the failed database) for all the database operations. In the meantime, an asynchronous polling process is forked on the machine where the up database is located and it will re-synchronize the two databases as soon as the failed database is available. After polling finishes, VERSANT Object Manager automatically switches back to the normal replication operation. A more detailed description of fault-tolerant replication is given as follows.

Failure detection. VERSANT Object Manager detects a database failure when it receives a network type error from a connection request or a RPC to a replicated database. It then declares the database down and sends a request for polling process to the up database. When the cleanbe process of the up database sees the request in the shared memory, it will fork a polling process. The polling process begins a session on the up database, sets the status of the up database in the shared memory to SYNC, and keeps trying to connect to the failed database. Note that multiple applications may be running concurrently, and each of them may detect the failure and request a polling process. Because only one polling process is needed for each replica pair, we have implemented a mechanism to ensure that only one polling process is forked.

Deferred list. In order for the polling process to re-synchronize the two databases correctly, we define a system class "o_repobj" to keep all the update information in the up database. One o_repobj instance is created in the up database for each transaction during the period when the failed database is declared down. Each o_repobj instance keeps a list of LOIDs and a list of actions (create, update, and delete) in 1-to-1 correspondence to record all the changes committed to the up database in one transaction. Note that if the database failure is detected in the middle of a transaction before the second-phase commit, VERSANT Object Manager will initialize the new o_repobj with all the changes which have been logged in the up database for this transaction. After the failed database is declared down, VERSANT Object Manager switches to the up database for all the object operations and at the same time adds the (LOID, action) information to the o_repobj of the current transaction. Note that since we can always get the latest state of an object based on its LOID from the up database, we do not have to store the whole object in the deferred list. We have modified the two-phase commit protocol to allow the transaction to commit as long as one of a replica pair is still available. Note that since multiple applications may update the up database concurrently, each o_repobj is assigned a time-stamp to record the commit time of its associated transaction so that the polling process can redo all the actions in the correct commit order.

Synchronization. When the failed database is back on-line again, the polling process is the only one allowed to make the connection. We have implemented a complex mechanism to prevent any new application from accessing the failed database before the polling process finishes. The polling process first does a distributed transaction recovery based on an extended recovery protocol which handles dual copies of coordinator information. It then selects all the o_repobjs from the up database and sorts them in ascending order based on their commit time-stamp. For each o_repobj, polling process re-applies the operations stored in the defer lists to the failed database. For create and update operations, it reads the corresponding objects based on their LOIDs from the up database and writes them to the failed database. For delete operations, it deletes the corresponding objects from the failed database based on their LOIDs. After finishing all the actions in one o_repobj, the polling process deletes this o_repobj and does a commit. While the polling process is doing the synchronization, those running applications may continue to start new transactions and write new o_repobjs to the up database. Therefore, the polling process will select o_repobjs repeatedly until there is no left-over. If the failed database fails again (recursive failure) during the synchronization, the polling process will resume the synchronization when the database is back again. If the only polling process dies unexpectedly, the cleanbe process will detect the exit status and fork a new polling process automatically.

Up detection. When polling process finishes, it sets the status of the up database to SYNC_DONE in the shared memory. At this moment, the polling process has brought the two databases almost in-sync except the changes made by those running transactions which have not committed yet. For each running VERSANT Object Manager, when it writes an o_repobj to the up database during commit time, the back-end server process checks the database status and returns a special status code if it is SYNC_DONE. After receiving this status code, VERSANT Object Manager connects to the failed database and does the re-synchronization by itself for the current o_repobj. If the failed database fails again, a request for polling process is sent to the up database and a new polling process will be forked. If the synchronization finishes successfully, the failed database is declared up and back to normal replication. After all the active applications on the up database detects the SYNC_DONE, the status of the up database is back to UP again.

In the above, we have given an overview of the design and implementation of VERSANT Replication which is the first ODBMS to support fault-tolerant object databases transparently. The target applications of VERSANT Replication are those 24x7 mission-critical applications which allow zero database down time. By maintaining a hot stand-by replica database, VERSANT Replication provides for fault-tolerant operation to any object database which is critical to a 24x7 application. It differs from traditional asynchronous replication mechanisms in that they serve different purposes in different application domains. The trade-off for using VERSANT Replication is performance degradation to some extent. For normal replication case, there is an extra RPC sent to the replica database for every write, delete, locking, and commit RPC sent to the primary database. For failure case, there is a small overhead in the front-end process for adding the (LOID, action) information to the deferred list. When each front-end process detects that the polling process has finished, it has to re-connect to the failed database and do the synchronization for the current transaction. There is no performance overhead for read with no lock (dirty read) operation in both normal replication and failure cases because we read from only one database. There is also no performance overhead for manipulating objects in the front-end process memory. VERSANT Replication is now shipping in beta release.