Virtuoso Replication and Synchronization Services

Abstract

Database Replication and Synchronization are often considered arcane subjects, and the sole province of the DBA (database administrator). However, there are many scenarios that speak for the creative use of Replication services. One question we might ask is, 'if data replication services were not so arcane, would we find more occasion to use these services to solve common business challenges'.
The are several flavors of replication, yet all have in common the objective of maintaining consistent datasets across multiple systems, without the absolute requirement of dedicated connections. In this monograph, we will discuss the various database replication services provided by OpenLink's Virtuoso Universal Server.

Replication and synchronization are terms signifying periodic or continuous copying of selected data between databases. Systems that comprise multiple databases are likely to feature some form of replication and synchronization. Also, capital line systems which evolve (by accident or design) as multi-site databases need replication services in order to decrease point failure dependencies, while enhancing performance. Replication may also be used as a master data distribution mechanism when periodic updates need to be propagated to intermittently connected systems.

Replication is used in the following situations:

- Load Balancing of multiple servers hosting duplicate datasets
- Hot failover for backup server data
- Publishing of selected 'Data Catalog' tables within a distributed system
- Reconciliation of independent systems; e.g. order batch staging via mobile devices and the interposing mobile tier
- Distribution and periodic synchronization of document databases within a content management system
- Off-line analysis or data migration to/from application versions

Virtuoso Replication Techniques

The Virtuoso Virtual Database offers several methods for maintaining data synchronization within a network of heterogeneous servers:

- Two-way synchronization with programmable conflict resolution is provided between Virtuoso and most leading databases. Virtuoso can also periodically push data to other databases based on update tracking triggers. Virtuoso provides trigger based updates between popular databases, including Oracle, MS SQL Server, IBM DB2 and others.
- Transactional Replication is used for keeping relational and object repository data in tight synchrony between Virtuoso servers. Transactional Replication may be unidirectional from central publisher to one or more subscribers, or may be bi-directional with conflict resolution. Transactional replication is executed between database engines, whereas triggered synchronization occurs at the table instance level.
- Virtuoso also supports the Sync/ML protocol for mobile devices. Virtuoso is an ideal host for conversion logic between mobile data and device messaging formats. Sync/ML replication supports DAV collections of vcards, vcalendar, and other XML-based mobile application data.

The Virtual Database (VDB)

Each of Virtuoso's replication modes have a specific suitability for various business scenarios. In a data integration scenario, VDB replication techniques have the advantage of supplying replication services to many popular database servers connected via Virtuoso. There are few downsides to this common topology, but one should be aware of the inherent limitations.

VDB transaction updates are fast, but never real time. This is a limitation shared by all data integration schemes, as updates must be detected and propagated via programmatic control

Inter-databasesynchronization via the VDB applies only to data in rows, columns, and cells. The transaction boundaries, involving procedure calls and updates to state data, is not supported. For this, Transactional Replication is used.
The Virtuoso VDB supports both replication techniques. Her is an explanation:

In Two-Way replication, a Virtuoso server is designated as the publisher of one or more tables. Multiple DBMS servers (including a Virtuoso database instance) may act

as subscribers. Periodic reconciliation runs are executed by the publisher. Each run reads the subscriber change log table, imports modified data, and applies changes to the Publishers copy, detecting conflicts due to post-copy changes. Changes to the Publisher's copy are then pushed to the subscribers.

This technique is readily used for synchronizing data between brand-x servers. Simply create a publication on Virtuoso, initialize it with the data, and subscribe the connected servers involved. Virtuoso creates the change tracking triggers, tables, and will maintain subscriber synchronization on a periodic basis. Configuration of conflict resolution logic is located on Virtuoso, the publisher.

In the following example, a local copy of the Northwind demo database is replicated on Oracle and MS SQL Server remote database.

Figure 1 – Replication SQL Commands to Create Order and Order Details in MS SQL Server and Oracle Databases

```
REPL_CREATE_SNAPSHOT_PUB ('Demo.demo.Orders', 2);

REPL_CREATE_SNAPSHOT_PUB ('Demo.demo.Order_Details', 2);

REPL_SNP_SERVER ('mssql_dsn', 'mssql_uid', 'mssql_pwd');

REPL_CREATE_SNAPSHOT_SUB (REPL_SERVER_NAME ('mssql_dsn'), 'Demo.demo.Orde

REPL_CREATE_SNAPSHOT_SUB (REPL_SERVER_NAME ('mssql_dsn'), 'Demo.demo.Orde

REPL_SNP_SERVER ('oracle_dsn', 'oracle_uid', 'oracle_pwd');

REPL_CREATE_SNAPSHOT_SUB (REPL_SERVER_NAME ('oracle_dsn'), 'Demo.demo.Ord

REPL_CREATE_SNAPSHOT_SUB (REPL_SERVER_NAME ('oracle_dsn'), 'Demo.demo.Ord
```

Having done this, we can call:

```
REPL_INIT_SNAPSHOT (REPL_SERVER_NAME ('mssql_dsn'), 'Demo.demo.Orders', 2

REPL_INIT_SNAPSHOT (REPL_SERVER_NAME ('mssql_dsn'), 'Demo.demo.Order_Deta

REPL_INIT_SNAPSHOT (REPL_SERVER_NAME ('oracle_dsn'), 'Demo.demo.Orders',

REPL_INIT_SNAPSHOT (REPL_SERVER_NAME ('oracle_dsn'), 'Demo.demo.Order_Det
```

to synchronize the local table and its two replicas. Conflict resolution will be executed when appropriate. After the function returns, all three tables will have identical content.

This same process can be accomplished interactively through Virtuoso's web admin interface.

Incremental Push allows pushing a Virtuoso table to a remote database or Virtuoso

instance. One update-tracking table is kept on Virtuoso. Simple snapshot replication is suited for one-time copying of data between Virtuoso and other databases.

Transactional Replication

Transactional Replication is the most efficient and flexible synchronization model in terms of response time and customizability, but is principally used between Virtuoso servers. Transactional Replication has two variants, bi-directional with conflict resolution, and one- way, from publisher to subscribers.

Transactional Replication's basic unit is a publication, comprising schema, tables, stored procedures, and a transaction history log. Modification of data and stored procedure calls are recorded in the publication log, encapsulating the history of the publication. Replaying the log on a subscriber is akin to a miniature database recovery event invoked at regular intervals. Sequence indexing insures that transactions are received by subscribers in the same order as they are committed, and only as complete transactions. Subscribers may require only intermittent connections to the master publisher, depending on the application.

Transactional replication's added advantage is the logging of complete transactions, including procedure calls. Entire logical operations are transferred via a publication, as opposed to a change of data within a table, or row set. This is more efficient, and offers possibilities for distributing application intelligence throughout the replication systems architecture.

Transactional replication is well suited for load balancing. Although two-phase commit offers guaranteed update consistency, transactional replication comes very close, as subscriptions take only milliseconds to complete. In any event, the publisher is the sole arbiter of whether a transaction will commit.

When a subscriber completes a publication cycle, it will remain connected, receiving the publication feed of current transactions until the publisher deems ready to commit; two servers may replicate each other using this method. Virtuoso uses transaction log-shipping for implementing hot failover and load balancing, where transaction logs are transferred for off-line replay.

The examples details in Figures 3, 4 and 5, we make a mock procedure for entering an order into the Northwind database. We then publish the Orders and Order_Details tables and this procedure as a transactional publication.

On Publisher:

Figure 3 – Stored Procedure NEW_ROUTINE_ORDER for Order and Order Detail Data

```
create procedure NEW_ROUTINE_ORDER (in o_quantity integer)
{
  declare o_id integer;

  o_id := null;
  select top 1 OrderID into o_id from Demo..Orders order by OrderID desc;
  o_id := coalesce (o_id, 0) + 1;
  insert into Demo.Demo.Orders (OrderID, CustomerID, EmployeeID, OrderDat
    values (o_id, 'ALFKI', 1, now ());
  insert into Demo.Demo.Order_Details (OrderID, ProductID, UnitPrice, Qua
    values (o_id, 1, 18.00, o_quantity);
}
;
```

Figure 4 – Publisher - SQL Replication Commands

```
REPL_PUBLISH('routine_orders', 'ro.log', 1, 'demo');

REPL_PUB_ADD('routine_orders', 'DB.DBA.NEW_ROUTINE_ORDER', 3, 1, 1);

REPL_PUB_ADD('routine_orders', 'Demo.Demo.Orders', 2, 1, NULL);

REPL_PUB_ADD('routine_orders', 'Demo.Demo.Order_details', 2, 1, NULL);
```

Figure 5 – Subscriber - SQL Replication Commands

On Subscriber:

```
REPL_SERVER('routine_orders_srv','routine_orders_dsn', 'some_host:1111');

REPL_SUBSCRIBE('routine_orders_srv','routine_orders', 'dav', 'dav', 'demo

REPL_SYNC ('routine_orders_srv', 'routine_orders', 'demo', 'demo-pwd');
```

Now, each change to published tables will be replayed on the subscriber in a master/slave relationship . In this case, the definition of the procedure was copied as part of the subscription, but local redefinition by the subscriber is possible.

These operations can also be performed manually via the Virtuoso web based administration interface.

Transactional replication is best defined as a transactional message queue, not limited to solely to table updates. Transaction event messages may be called from any application level event, providing a powerful synchronization medium.

Processing resource allocation transactions in a distributed systems is a good example of the power of transactional replication – especially for intermittently connected clients. An order processing procedure call may invoke client and server specific actions. In this case, the client's order processing procedure may decrement an SKU quantity, while a server procedure may check the inventory delta, update the

central inventory, and other appropriate actions.

Transactional replication is optimized for Virtuoso servers, but there are simple methods to bring this powerful technique to heterogeneous attached systems. One method is to deploy a Virtuoso instance as a front-end to another database. Virtuoso will update the attached system(s) based on data from the replication feed.

Likewise, change-logging triggers on a remote database can initiate a replication event for Virtuoso to include into a transactional publication. In this case, Virtuoso is the glue binding two dissimilar databases into a transactional replication relationship. Virtuoso, in this scenario, will not be required to hold a copy of the database tables.

Conflict Resolution

Replication is not a distributed transaction scheme, and carries with it the potential of data divergence[1]. As replication frequency increases, divergence becomes less of an issue, but remains non-zero. Therefore, for systems that depend on replication, a mechanism for reconciling update conflicts is required.

Virtuoso Server implements conflict resolution by transparently adding a global ID column to publisher and subscriber tables. All row updates create a global ID increment. Primary row keys determine 'sameness' of the row in all attached system instances. A change in the primary key conveys a delete and insert.

A conflict is detected when a row received from a subscriber differs from a corresponding row on the publisher. If the global ID's indicate a conflict, application logic will invoke a conflict resolution mechanism. Conflict resolution is the same for both VDB and transaction based schemes.

There are standardized methods for determining the ultimate owner of a resolution update result; in most cases, application logic will be needed for deciding on the course of action.

The example in Figure 6, we detect a conflicting update of an employee's residence address. The publisher wins but we log the event into a tracking table.

Figure 6 – Stored Procedure to detect update conflict in Employees Address

```
create procedure "Demo"."demo"."replcr_U_Employees_hd" (
  inout "_EmployeeID" INTEGER,
  inout "_LastName" VARCHAR(20),
  inout "_FirstName" VARCHAR(10),
  inout "_Title" VARCHAR(30),
  inout "_TitleOfCourtesy" VARCHAR(25),
  inout "_BirthDate" DATE,
  inout "_HireDate" DATE,
  inout "_Address" VARCHAR(60),
  inout "_City" VARCHAR(15),
  inout "_Region" VARCHAR(15),
  inout "_PostalCode" VARCHAR(10),
  inout "_Country" VARCHAR(15),
  inout "_HomePhone" VARCHAR(24),
  inout "_Extension" VARCHAR(4),
  inout "_Photo" LONG VARBINARY,
  inout "_Notes" LONG VARCHAR,
  inout "_ReportsTo" INTEGER,
  inout "__old_EmployeeID" INTEGER,
  inout "__old_LastName" VARCHAR(20),
  inout "__old_FirstName" VARCHAR(10),
  inout "__old_Title" VARCHAR(30),
  inout "__old_TitleOfCourtesy" VARCHAR(25),
  inout "__old_BirthDate" DATE,
  inout "__old_HireDate" DATE,
  inout "__old_Address" VARCHAR(60),
  inout "__old_City" VARCHAR(15),
  inout "__old_Region" VARCHAR(15),
  inout "__old_PostalCode" VARCHAR(10),
  inout "__old_Country" VARCHAR(15),
  inout "__old_HomePhone" VARCHAR(24),
  inout "__old_Extension" VARCHAR(4),
  inout "__old_Photo" LONG VARBINARY,
  inout "__old_Notes" LONG VARCHAR,
  inout "__old_ReportsTo" INTEGER,
  inout __origin varchar)
 {
  insert into EMPLOYEE_ADDRESS_CONFLICT_LOG (EACL_NEW_ADDRESS, EACL_OLD_A
     ("_Address", "__old_Address", now());
  return 3;  -- publisher wins
 }
;

insert into DB.DBA.SYS_REPL_CR (CR_ID, CR_TABLE_NAME, CR_TYPE, CR_PROC, C
  (0, 'Demo.demo.Employees', 'U', 'Demo.demo.replcr_U_Employees_hd', 10);
```

Notice that the conflict resolution function retains the old and new values of all columns.

Replication and Security

In virtual database replication, security is handled internally as a SQL account privilege. Transactional Replication security depends on the subscriber having been granted rights by the publication.

Document Replication

Object collections in Virtuoso DAV can be added to transactional publications, therefore subscribers may replicate identical DAV collections and paths on a remote server. Update conflicts are non-destructive, as a losing document will be safely archived in a sub-collection, rather than being overwritten or dropped.

Sync/ML

Mobile devices may mutually share data between themselves and a central server using Sync/ML. Virtuoso DAV collections may receive Sync/ML protocol events, allowing management of device ID, login session state, and access permissions. These resources may also be conveyed to other Sync/ML clients and systems.

Virtuoso's powerful data transformation capabilities allow Sync/ML services to be configured for on-the-fly format translation between otherwise incompatible mobile devices[2].

Conclusion

Distributed systems will always have a need to share data between nodes using various expedient methods. Virtuoso's replication capabilities provide a straightforward feature set that is quite flexible, offering a broad range of capabilities. For integration driven applications, Virtuoso offers a platform for complex inter-server data interchange.

Learn More

- **Virtuoso Documentation**
- **Virtuoso Tutorials**

Appendix

Supported Databases

- Bi-directional heterogeneous replication is supported for Oracle 9i onwards,

MS SQL Server 2000 onwards, IBM DB2 7 onwards.

- One-way heterogeneous replication from Virtuoso to other databases is additionally supported for Informix.

Non-incremental snapshot replication is supported for most ODBC data sources, including MS Access and many others.

References

SyncML Links

- http://www.openmobilealliance.org/syncml/technology2.html

Other Links

- http://www.syncml.org./
- http://sourceforge.net/projects/syncml-ctoolkit/
- http://www-106.ibm.com/developerworks/xml/library/x-syncml3.html

[1] Distributed transaction systems may duplicate simultaneous writes to physically separated systems, whereas replication conveys synchronization events to remote systems – in the former, it is the miniscule network latency which causes divergence, whereas in the latter, it is replication cycle time, interval of connection, and latency.

[2] Virtuoso Sync/ML support is separate from Virtuoso's other replication features.