Replication technologies in Informix Dynamic Server
On servers, domains, and clusters

Madison Pruet
Nagaraju Inturi
Nicholas Geib

IBM® Informix® Dynamic Server (IDS) has long been a database with powerful replication technologies. Enterprise Replication (ER) and High-Availability Data Replication (HDR) have been faithfully serving customers for years. Recent versions of IDS, 11.1 and now 11.5, add some new technologies to these stalwarts: the Remote Standalone secondary and the Shared-Disk secondary servers. In this article, get a brief characterization of each replication technology and details on the basics of using each.

Enterprise Replication (ER)

What is ER?

Databases store important data for your business, and business people recognize that storing that data in at least two places is a wise strategy. Wouldn't it be great if you could tell your database to replicate the data you wanted to another server automatically? Well since Version 7, IDS has allowed you to do that. This occurs through our Enterprise Replication (ER) technology, which is both flexible and powerful.

ER works by you first defining the servers among which you would like data to be replicated. This creates a network topology — of root, leaf, and non-root, non-leaf nodes — that will transfer the data. Each ER node may be a single server or a cluster of servers, which this article further discusses below. All the interconnected nodes together are called an ER domain. The domain does not define what data will be transferred, just the paths or routes along which the data will flow.

Next, you decide what data you would like to be replicated. This is defined by an SQL query involving a server, database, table, and columns. The query acts as a filter whose output decides the data to replicate and proves to be a very flexible tool. If you'd like an entire table to be replicated, your query would be SELECT * FROM ... . If you'd like just the first three columns to be replicated, your query would now be SELECT column1, column2, column3 FROM ... . Or, if you'd like only certain rows to be replicated, you just use a WHERE clause in your query. ER can replicate...
data with very broad or very fine filters. To help with replication ER requires that the table have a primary key defined.

After the query is written, determine what nodes should participate in replicating the data. Let's say you want nodes A, B, and C to all have the same data in their Employee table. So A, B, and C are your participants. ER can be configured so that if data changes are made on any server, the other participants become updated too. This is called an update anywhere scenario. What happens if you only want updates to flow from A, into both B and C, but back never to A? ER gives you the flexibility to do these primary-target scenarios as well. In situations where data changes occur on more than one participant, it may happen that a row changed on server B conflicts with a change occurring on server C. ER allows you to define rules for automatically handling such conflicts. These include comparing timestamps, running a stored procedure, and ignoring conflicts.

Now that you know the query, the participants, and the scenario, take all this information and use it to create something called a replicate. Think of a replicate as a faucet that controls the flow of data from the query. The faucet can be turned on or off, and you can have as many faucets as you want. Replicates can be grouped into sets, which make it easier for users to control multiple replicates. Also, you can use templates to help you quickly create replicates for multiple tables. All this work is done through the server utility cdr.

After a replicate has been created and started, how does the data actually get moved? Well, first of all, the replication happens asynchronously. This means there may be a delay between data being committed on one node and it appearing on another node. And only committed data, of course, is replicated. ER works by reading the logical logs, testing if a log record needs to be replicated, and finally sending that information to the appropriate participants.

**A simple example**

Let's set up a simple example of ER. We'll replicate the employees table of the "db1" database in an update-anywhere fashion between server1 and server2. (The following steps assume the informix account is used.)

**Step 1: Prepare the SQLHOSTS files**

The file for each server should contain an entry for both servers as well as two group definitions. Each group represents an ER node:

```
grp1 group - - i=1
server1 onsoctcp host port g=grp1

grp2 group - - i=2
server2 onsoctcp host port g=grp2
```

**Step 2: Prepare the ER disk space**

For each server, perform the following steps:

1. Edit the ONCONFIG to contain a smart blob space:
   ```
   CDR_QDATA_SBSPACE sbs1
   ```
2. Start the server.
3. Create a chunk for that smart blob space, and add the space to the server. For example, on UNIX:

```
touch /ids/chunks/sbs1_chunk1
chmod 660 /ids/chunks/sbs1_chunk1
onspaces -c -S sbs1 -p /ids/chunks/sbs1_chunk1 -s 100000 -o 0
```

**Step 3: Add the nodes to a new ER domain**

On server1, execute the following command:

```
cdr define server --init grp1
```

On server2, execute the following command:

```
cdr define server --sync=grp1 --init grp2
```

These commands create an ER *domain* of two root nodes.

**Step 4: Prepare the table to replicate**

On both server1 and server2, run the following SQL statements:

```
CREATE DATABASE db1 WITH LOG;
CREATE TABLE employees(id int primary key, ...);
```

**Step 5: Create the replicate**

On server1, execute:

```
cdr define replicate --conflict=ignore r_employees 
"db1@grp1:informix.employees" "select * from employees" 
"db1@grp2:informix.employees" "select * from employees"
```

**Step 6: Start the replicate**

On either server, run:

```
cdr start replicate r_employees
```

Now you're replicating data! An insert, update, or delete to a row on either server1 or server2 will soon appear on the other server. This is only the beginning of using ER. To learn more about the technology and useful commands, please see the "Replication" section of the IDS InfoCenter (see Related topics).

**High availability cluster**

Starting in Version 11.1, IDS contains cluster-based replication functionality. In a cluster, you gain scale-out, high availability, disaster recovery, and load-balancing capabilities. This technology
is built right into the database and requires no additional hardware, software, or network infrastructure. In contrast to ER, in a cluster, you cannot control the granularity of data replication; the entire server's data is always replicated. Cluster and ER technologies, as this article shows later, can be used side by side.

A cluster consists of one primary server and a mix of three different types of secondary servers:

1. Shared Disk (SD) secondary server
2. High-Availability Data Replication (HDR) server
3. Remote Standalone (RS) secondary server

Servers must be the same IDS version and run on the same operating system, hardware, and disk architecture. In 11.1, the secondary servers only allowed applications to read data. In 11.5, applicants can now also insert, update, and delete data on secondary servers. Thus, all three secondary server types can be used to increase the capacity of your primary as well as balance the load. Table 1, below, compares the server types. Table 2 lists some common ONCONFIG settings.

### Table 1. Overview of the characteristics of cluster server types

<table>
<thead>
<tr>
<th>Configuration parameter</th>
<th>Primary</th>
<th>SD secondary</th>
<th>HDR</th>
<th>RS secondary</th>
</tr>
</thead>
<tbody>
<tr>
<td>First IDS version</td>
<td>11.1</td>
<td>6.00</td>
<td>11.1</td>
<td></td>
</tr>
<tr>
<td>Primary purpose</td>
<td>Increased capacity/scale-out</td>
<td>High availability/hot standby</td>
<td>Disaster recovery/remote backup</td>
<td></td>
</tr>
<tr>
<td>Maximum number of nodes per cluster</td>
<td>One</td>
<td>No limit</td>
<td>One</td>
<td>No limit</td>
</tr>
<tr>
<td>Possible type changes in one step</td>
<td>SD secondary, HDR, standard</td>
<td>Primary</td>
<td>Primary, RS secondary, standard</td>
<td>HDR, standard</td>
</tr>
<tr>
<td>Supported isolation levels</td>
<td>All</td>
<td>Dirty read, Committed Read, Committed Read Last Committed</td>
<td>Dirty read</td>
<td>Dirty read</td>
</tr>
</tbody>
</table>

### Table 2. ONCONFIG parameters common to cluster server types

<table>
<thead>
<tr>
<th>Configuration parameter</th>
<th>Server type</th>
<th>Supported values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HA_ALIAS</td>
<td>SD and RS secondary servers</td>
<td>DBSERVERNAME or one of the DBSERVERALIAS name. Must be a network protocol.</td>
<td>Used to define the name by which a secondary server is known within a cluster. Gives you flexibility to use something other than DBSERVERNAME.</td>
</tr>
<tr>
<td>LOG_INDEX_BUILDS</td>
<td>Primary, HDR, and RS secondary</td>
<td>0 - Disable index page logging</td>
<td>While creating a new index at primary server, Index Page Logging (IPL) writes index pages to logical log file. Enabling index page logging is required for RSS nodes and optional for HDR secondary server.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - Enable index page logging</td>
<td></td>
</tr>
<tr>
<td>REDIRECTED_WRITES</td>
<td>HDR, SD and RS secondary servers</td>
<td>0 - Disable write functionality at secondary server</td>
<td>Determines the number of network connections used between the primary and secondary server to support data updates.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;=1 - Enable write functionality at secondary server</td>
<td></td>
</tr>
</tbody>
</table>
Let's get to know these server types a little more. Let's explore their characteristics, their ONCONFIG parameters, and, along the way, build a cluster with one of each type.

**Shared Disk (SD) secondary**

SD secondary servers share disk space — except temporary dbspaces — with the primary server. This is typically done through a network-based clustered file system. Adding a new SD secondary to a cluster is very easy and can be done in a few seconds once the shared disk is prepared. Because SD secondary nodes leverage the primary’s disks and can be brought up easily and quickly, they are well-suited for scale-out scenarios. In SD secondary servers checkpoints are synchronized. This means a checkpoint at the primary server completes only after the checkpoint at the SD server completes. From IDS 11.5 onwards, an SD secondary supports committed read and committed read last committed isolation levels as well as dirty read. An SD secondary can be promoted to a primary server with one single command: onmode -d make primary <name of SD server>. Because an SD secondary server is so close to the primary (in other words, it shares the same disk), it is often the best type of server to initially fail over to if the primary should encounter a problem.

**Table 3. Important ONCONFIG parameters for the SD secondary**

<table>
<thead>
<tr>
<th>Configuration parameter</th>
<th>Server type</th>
<th>Supported values</th>
<th>Description</th>
</tr>
</thead>
</table>
| SDS_ENABLE              | Primary, SD secondary     | 0 - Disable SDS functionality  
                                | 1 - Enable SDS functionality | Use this to allow SD secondaries to be added to the cluster.                |
| SDS_PAGING              | SD secondary              | <absolute path for paging file1>,<absolute path for paging file 2> | Two paging files must be configured to bring up SDS node.                   |
| SDS_TEMPDBS             | SD secondary              | <dbspace_name>,<path>,<pagesize in KB>,<offset in KB>,<size in KB> | Temporary dbspace information for SD secondary node.  
You can configure up to 16 SDS_TEMPDBS entries.  
Example: SDS_TEMPDBS sdstmpdbs1, /work/dbspaces/sdstmpdbs1,2,0,16000 |
| SDS_TIMEOUT             | Primary                   | >= 0 seconds     | This configuration parameter is used at the primary to decide how long to wait for an acknowledgement from an SD server.  
If no acknowledgement occurs, the primary acts to shut the SD server down.  
The default value is 20 seconds. |
Adding an SD secondary to a cluster

Let's take a standalone IDS server and turn it into a cluster. First let's add an SD secondary server. (This scenario assumes that the shared disk file system has already been set up and the informix account is used.)

Step 1: Prepare the SQLHOSTS file

Make sure the SQHOSTS file at both the primary and SDS node has entries for the other server:

<table>
<thead>
<tr>
<th>g_mach11</th>
<th>group</th>
<th>-</th>
<th>-</th>
<th>i=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>myprim</td>
<td>ontlitcp</td>
<td>primhost</td>
<td>prim_port</td>
<td>g=g_mach11</td>
</tr>
<tr>
<td>sds1</td>
<td>ontlitcp</td>
<td>sds1host</td>
<td>sds1_port</td>
<td>g=g_mach11</td>
</tr>
</tbody>
</table>

Note that the use of groups here is optional. However, we include it to prepare for an example below.

Step 2: Set the primary as the owner of the shared disk

On the primary, run:

```
onmode -d set SDS primary myprim
```

Step 3: Configure the SD secondary

1. Make sure these match the ONCONFIG of the primary: ROOTNAME, ROOTPATH, ROOTOFFSET, ROOTSIZE, PHYSDBS, PHYSFILE, LOGFILES, and LOGSIZE.
2. Set SDS_ENABLE to 1.
3. Configure SDS_PAGING and SDS_TEMPDBS.

For example:

```
SDS_ENABLE         1
SDS_PAGING         /ids/sds/dbspaces/page_1,/ids/sds/dbspaces/page_2
SDS_TEMPDBS        sdtmpdbs1,/ids/sds/dbspaces/sdtmpdbs1,2,0,16000
REDIRECTED_WRITES  1
TEMPTAB_NOLOG      1
```

Step 4: Start the SD secondary

```
oninit
```

The SD secondary is now up and running! You can check the cluster status using the onstat -g sds command at both the primary and secondary servers. Here's what our cluster looks like:
High-Availability Data Replication (HDR)

High-Availability Data Replication or HDR is the oldest replication technology in IDS. It consists of a pair of servers — the primary and the HDR secondary — and supports both synchronous and asynchronous replication modes. In synchronous mode, transactions on the primary server will not commit until it receives an acknowledgement from the HDR secondary server. Thus the HDR secondary is immediately ready to take the place of the primary server — what is called a "hot" standby. In asynchronous mode, only checkpoints are synchronized between the primary and HDR secondary. One characteristic of HDR is that it uses a half duplex communication protocol and thus is sensitive to network latency. This is not the case for SD and RS secondary servers.

Table 4. Important ONCONFIG parameters for the HDR server

<table>
<thead>
<tr>
<th>Configuration parameter</th>
<th>Server type</th>
<th>Supported values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAUTO</td>
<td>Primary, HDR server</td>
<td>0 - Manual</td>
<td>Controls how the primary and HDR secondary behave in a failure scenario.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - Automatic failover ultimately retaining server types</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 - Automatic failover ultimately reversing server types</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 - Have the Connection Manager Arbiter control the failover</td>
<td></td>
</tr>
<tr>
<td>DRIDXAUTO</td>
<td>Primary, HDR server</td>
<td>0 - Disable automatic index repair</td>
<td>Automatically repair an index, if the HDR secondary server detects a corruption.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - Enable automatic index repair</td>
<td></td>
</tr>
<tr>
<td>DRINTERVAL</td>
<td>Primary</td>
<td>-1 - Operate in synchronous mode</td>
<td>The maximum interval, in seconds, between flushing of the high-availability data-replication buffer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;= 0 - Operate in asynchronous mode</td>
<td></td>
</tr>
<tr>
<td>DRLOSTFOUND</td>
<td>Primary, HDR server</td>
<td>&lt;path for lost and found file&gt;</td>
<td>The path for the HDR lost-and-found transaction file. This file is created during a failover and contains the transactions committed on the primary but not the HDR server.</td>
</tr>
</tbody>
</table>
Adding an HDR server to a cluster

Let's add an HDR secondary node to our cluster.

**Step 1: Prepare the SQLHOSTS files**

Update the SQLHOSTS file at primary and at both SDS and HDR secondary servers:

<table>
<thead>
<tr>
<th>Group</th>
<th>Group Type</th>
<th>Host</th>
<th>Port</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_mach11</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>i=10</td>
</tr>
<tr>
<td>myprim</td>
<td>ontlitcp</td>
<td>primhost</td>
<td>prim_port</td>
<td>g=g_mach11</td>
</tr>
<tr>
<td>hdr</td>
<td>ontlitcp</td>
<td>hdrhost</td>
<td>hdr_port</td>
<td>g=g_mach11</td>
</tr>
<tr>
<td>sds1</td>
<td>ontlitcp</td>
<td>sds1host</td>
<td>sds1_port</td>
<td>g=g_mach11</td>
</tr>
</tbody>
</table>

**Step 2: Configure the ONCONFIG files**

In order for HDR to work, certain `ONCONFIG` parameters must be exactly the same on both the primary and secondary. In many cases, this means they cannot be changed while either server is running. These parameters include `DRAUTO`, `DRINTERVAL`, `DRTIMEOUT`, settings related to the root dbspace, settings related to logical logs, and more. To ensure these settings match, one technique is to copy the onconfig of the primary to the secondary, and then modify the few things that must be different, such as `DBSERVERNAME`. You should plan for these settings before bringing the primary up. This example just uses the defaults.

**Step 3: Backup the primary**

On the primary, take a level zero backup:

```bash
ontape -s -L 0
```

**Step 4: Register the HDR secondary server with the primary**

On the primary, run:

```bash
onmode -d primary hdr
```

**Step 5: Prepare the HDR secondary server's disk**

The storage used on the HDR secondary must match that of the primary (for example, number of dbspaces, number of chunks, chunk sizes, pathnames, and offsets). Since this example uses the backup to restore the HDR secondary server, it is only necessary that the chunk files exist and have the proper permissions.

**Step 6: Restore the backup on the HDR secondary server**

On the HDR server, perform a physical restore of the level 0 backup:
ontape -p

Three questions will be asked. Answer as shown below:
Continue restore? (y/n) y
Do you want to back up the logs? (y/n) n
Restore a level 1 archive (y/n) n

Step 7: Bring the HDR secondary server into online mode

After the restore is complete, the HDR secondary will be in recovery mode. Run the following command:

onmode -d secondary myprim

HDR secondary is up and running! Run onstat -g dri at the primary and HDR secondary to check HDR status. Our cluster now looks like this:

Figure 2. The cluster: Primary, one SD secondary, and one HDR secondary

Remote Standalone (RS) secondary

The primary purpose of the RS secondary server is to provide a disaster recovery solution. As in HDR, the primary server sends all its logical log records to the RS secondary server continuously, except now only in an asynchronous manner. Unlike HDR, the communication occurs using a full duplex protocol. Thus the RS secondary is less sensitive to network latency and can more easily reside in a distant geography. One characteristic of an RS secondary is that unlike in SD and HDR servers, the primary server doesn't synchronize checkpoints with an RS secondary. Thus it is not immediately ready to take the place of the primary; it must be changed into an HDR server first. Should multiple failures in a cluster occur, however, an RS node can prevent your database from being entirely unavailable.

Table 5. Important ONCONFIG parameters for the RS secondary

<table>
<thead>
<tr>
<th>Configuration parameter</th>
<th>Server type</th>
<th>Supported values</th>
<th>Description</th>
</tr>
</thead>
</table>
### LOG_INDEX_BUILDS

<table>
<thead>
<tr>
<th>LOG_INDEX_BUILDS</th>
<th>Primary</th>
<th>0 - Disable index page logging</th>
<th>1 - Enable index page logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary</td>
<td></td>
<td>0 - Disable index page logging</td>
<td>1 - Enable index page logging</td>
</tr>
</tbody>
</table>

While creating a new index at primary server, Index Page Logging(IPL) writes index pages to logical log file. Index page logging must be enabled at primary server to add RS server to a cluster.

### Adding an RS secondary to a cluster

Let's add an RS node to our cluster.

#### Step 1: Prepare the SQLHOSTS files

All servers in the cluster must have SQLHOSTS entries for all others.

```plaintext
<table>
<thead>
<tr>
<th>g_mach11</th>
<th>group</th>
<th>-</th>
<th>i=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>myprim</td>
<td>ontlitcp</td>
<td>primhost</td>
<td>prim_port</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>g=g_mach11</td>
</tr>
<tr>
<td>hdr</td>
<td>ontlitcp</td>
<td>hdrhost</td>
<td>hdr_port</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>g=g_mach11</td>
</tr>
<tr>
<td>sds1</td>
<td>ontlitcp</td>
<td>sds1host</td>
<td>sds1_port</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>g=g_mach11</td>
</tr>
<tr>
<td>rss1</td>
<td>ontlitcp</td>
<td>rss1host</td>
<td>rss1_port</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>g=g_mach11</td>
</tr>
</tbody>
</table>
```

#### Step 2: On the primary, enable index page logging

```
onmode -wf LOG_INDEX_BUILDS=1
```

#### Step 3: On the primary, register the new RS secondary

```
onmode -d add RSS rss1
```

#### Step 4: Take a level 0 backup of the primary

```
ontape -s -L 0
```

#### Step 5: On the RS secondary, restore the backup

```
ontape -p
```

Three questions will be asked. Answer as shown below:

- Continue restore? (y/n) y
- Do you want to back up the logs? (y/n) n
- Restore a level 1 archive (y/n) n

#### Step 6: Bring the RS secondary into online mode

```
onmode -d RSS myprim
```

Now RSS node is up and running! Run the `onstat -g rss` command to check RSS node status at both primary and RSS nodes. Now our cluster looks like this:
Figure 3. The cluster: Primary, one SD secondary, one HDR secondary, and one RS secondary

Conclusion

In this article, you’ve learned about the flexibility of asynchronous replication using ER and the high-availability features of a cluster. What if you want to use both together? Well, you can! Remember you used the "group" notations for the cluster SQLHOSTS files? That same group can become an ER node just like server1 and server2 in the article example. You use the same command to add a cluster as a node to an ER domain as you do a standalone server: `cdr define server --sync=<sync node> --init <name of new group to add>`. When a cluster is an ER node, the primary performs the same responsibilities that a standalone server node does. However, since it’s a cluster, the primary also replicates the data to all of its secondary servers.

Now, there is more to the story — more about each of these individual technologies, more about how they can work together to make your business better, and more about related technologies like the Connection Manager and Continuous Log Restore! We invite you to learn more about IDS!
Related topics

- **IDS 11.5 InfoCenter**: Check out the place for official IDS documentation.
- "**Availability Solutions with Informix Dynamic Server 11**" (IBM, May 2007): Gain an understanding of the availability features of IDS 11 that can be combined together according to business need, and can be quickly and easily reconfigured and expanded. (PDF)
- "**What's new in IDS 11?**" (developerWorks, June 2007): Your reference for the new features in the IDS 11. Explore various aspects of IDS, such as scalability, high availability and performance, integrated solutions, and administration-free zone.
- **developerWorks Informix zone**: Find articles and tutorials, and connect to other resources to expand your Informix skills.

© Copyright IBM Corporation 2008
**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)