Information Management software

DB2 10.1 HADR Multiple Standbys

May 2012

Bruce Jackson (bmj@us.ibm.com)
Dale McInnis (dmcinnis@ca.ibm.com)
Effi Ofer (effio@il.ibm.com)
Nailah Ogeer Bissoon (nbissoon@ca.ibm.com)
Punit Shah (punit@us.ibm.com)
Roger Zheng (rzheng@ca.ibm.com)
Vincent Kulandai Samy (vinci@us.ibm.com)
Yuke Zhuge (zhuge@us.ibm.com)

1 Executive Summary

The DB2 High Availability Disaster Recovery (HADR) feature is a database replication method that provides a high availability solution for both partial and complete site failures. HADR protects against data loss by replicating data changes from a source database, called the primary, to one or more target databases, called the standbys.

The multiple standby HADR feature provides database administrators (DBAs) with a single technology to provide both high availability (HA) as well as disaster recovery (DR). This paper shows how HADR multiple standbys can be set up, configured, and monitored. In addition, various examples demonstrate how HADR multiple standby behaves in different failure scenarios.

2 Business value of HADR

With the introduction of HADR multiple standby support, you can now use a single technology to address both your high availability and disaster recovery needs. Using a single technology to address both critical needs can make the setup, configuration, and maintenance of your DB2 system much easier.

There are a number of other ways in which the HADR standbys can be used beyond their HA or DR purpose:

- **Active use of standby resources by using reads on standby**
  The reads on standby feature can be used to direct a read-only workload to one or more standby databases without affecting the HA or DR responsibility of the standby. This feature can help reduce the workload on the primary without affecting the main responsibility of the standby. Only the primary database is accessible unless the reads on standby feature is enabled on the standby. An application that connects to the standby database does not affect the availability of the standby in the case of a failover.
- **Flexible recovery strategies by using delayed replay**
  Delayed replay can be used to specify that a standby database is to remain a specified number of seconds behind the primary, in terms of log replay. If data is lost or corrupted on the primary, it can be recovered on the time delayed standby.
- **Rolling updates and upgrades**
  Using an HADR setup, you can make various types of upgrades and DB2® fix pack updates to your databases with only a momentary outage during the switchover. With multiple standby mode enabled, you can perform an upgrade while maintaining the protection provided by HADR.

DB2 HADR technology is a flexible architecture that you can use to address several key availability needs in most of your environments.

3 Introduction to multiple standby

You can use DB2 High Availability and Disaster Recovery (HADR) to create a warm copy of a database on a separate site by using the database transaction log shipping method. Beginning in DB2 Version 10.1, the feature is enhanced to support up to three standby databases. Having multiple standbys enables the combination of high availability (HA) and disaster recovery (DR) scenarios for a single database multisite data replication. Additionally, when a multiple standby setup is coupled with the other HADR enhancements for Version 10.1, log spooling and delayed replay, it allows quick recovery from a user error or an errant transaction.

To enable HADR multiple-standby mode, use of the new `hadr_target_list` database configuration parameter. The number of entries specified by this parameter on the primary determines the number of standbys a primary database has. Details about multiple standby configurations follow in subsequent sections.

Traditional HADR features and functionalities work with multiple standbys as well. For example, reads on standby is supported on any standby. Any standby can perform forced or graceful takeover operation. There are important considerations including recovery point objective and automatic reconfiguration regarding the takeover operation for multiple standby. Details follow later. Furthermore, cluster manager automation using IBM Tivoli System Automation for Multiplatforms (SA MP) is supported for multiple standby configurations with a primary database and one of the standby databases. Rolling upgrade is also supported by the multiple standby feature.

You can easily convert a single standby configuration to a multiple standby configuration. Details on conversion are presented in the *Setting up a multiple standby system* section.

3.1 Principal standby versus auxiliary standby

When you deploy the HADR feature in multiple standby mode, you can have up to three standby databases in your setup. You designate one of these databases as the principal HADR standby database; any other standby database is an auxiliary HADR standby database. Both types of HADR standbys are synchronized with the HADR primary database through a direct TCP/IP connection; both types support reads on standby; and you can configure both types for time-delayed log replay. In addition, you can issue a forced or graceful takeover on any standby.

There are some important distinctions between the principal and auxiliary standbys. IBM Tivoli System Automation for Multi-platforms (SA MP) automated failover is supported only for the principal standby. You must manually issue a takeover on one of the auxiliary standbys to make them the primary.

HADR supports a variety of log shipping synchronization modes to balance performance and data protection:

- SYNC: Log write on primary requires replication to the persistent storage on the standby.
- NEARSYNC: Log write on primary requires replication to the memory on the standby.
- ASYNC: Log write on primary requires a successful send to standby (receive is not guaranteed).
- SUPERASYNC: Log write on primary has no dependency on replication to standby

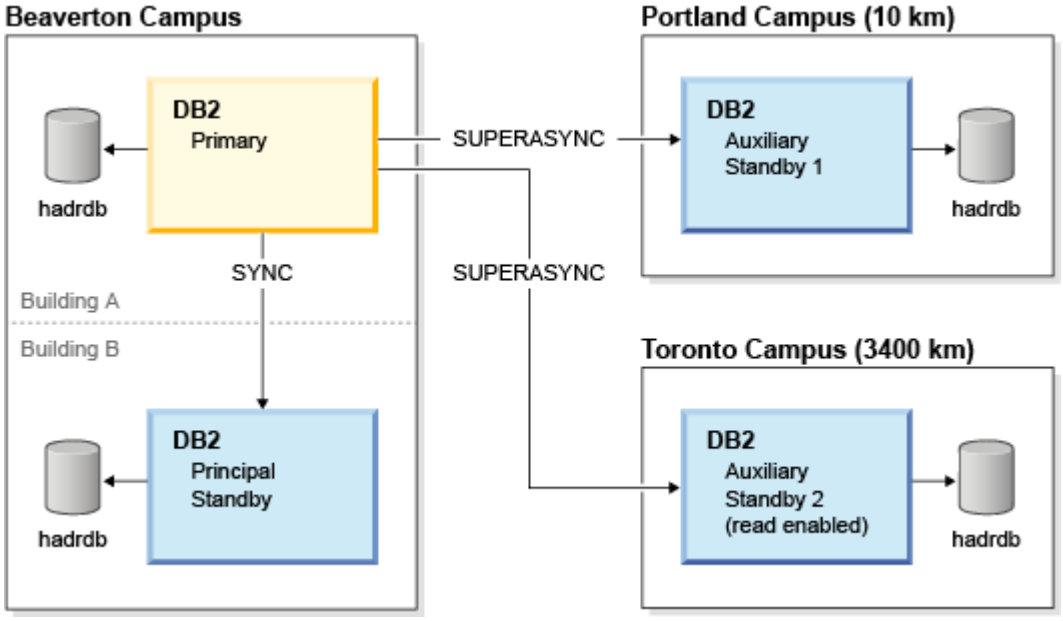
All of the HADR sync modes are supported on the principal standby, but the auxiliary standbys' synchronization mode is always SUPERASYNC mode.

3.2 Multiple standby example

Figure 1 depicts a representative, fully configured (to the three standby maximum) multiple standby setup for a company that has offices in the United States and Canada. The company has its HADR primary database called hadrdb at a site in Building A in Beaverton, Oregon, USA. The principal standby is in Building B of the Beaverton campus and it is configured with a synchronization mode of SYNC. SYNC

mode was chosen because of a business requirement mandating zero data loss in case of a single outage. To sustain high performance with SYNC synchronization mode, a LAN backbone to support fast round-trip time between the buildings is set up. With SYNC mode and the ability to quickly failover to the principal standby, this setup provides high availability capability in case of any outage on the primary site, such as a storage system failure or a power outage in Building A.

Figure 1: Multiple standby configuration



The company has also deployed an auxiliary standby in Portland, Oregon, USA, around 10 km away from the Beaverton buildings to provide a protection against a regional outage on the Beaverton campus, such as fire destroying both the primary and its principal standby. The auxiliary standby uses SUPERASYNC synchronization mode.

In order to enhance disaster recovery capability in case of a wider scope disaster affecting both Beaverton and Portland, the company has deployed another auxiliary standby at a company site in Toronto, Canada. In addition to this disaster recovery role, the company has enabled the reads on standby feature on the auxiliary standby at the Toronto campus to service a reporting workload. Later sections on SA MP automation and delayed replay illustrate how this company might further enhance this type of HADR environment using those features.

4 Planning for multiple standbys

There are several aspects you should consider when planning for multiple standbys. The planning steps help ensure a smooth multiple standby rollout, including a continued level of performance and possibly avoiding any downtime. There are two ways that you can create a multiple standby system:

- Set up a brand new multiple standby system on an existing database.
- Convert an existing HADR single standby system into a multiple standby system.

The following sections discuss how to set up multiple standbys in both scenarios.

4.1 How many standbys?

The first question to answer is how many standby databases you need. Having more than one standby provides both high availability (HA) and disaster recovery (DR). With one standby either HA or DR is possible, but not both. Any additional standby can be used for extra DR protection, reads on standby, or delayed replay. On the other hand, each standby needs its own dedicated hosting environment containing processors, memory, disks, and networking equipment; however, the different standbys do not have to be of equal capacity. In addition, there are maintenance and administrative costs associated with managing each database site.

4.2 Performance impact on primary

There is a marginal processor utilization increase on the primary site when going from no HADR to a single standby setup. Furthermore, there is an additional increase for additional auxiliary standbys. With the increasing number of standbys, network bandwidth requirements on the primary site increase proportionally by a factor of the number of standbys. There is little increase in memory consumption with the increasing number of standbys. Details about performance and scaling are presented in the *Multiple Standby Performance* section.

4.3 Network bandwidth requirement

Ensuring that there is sufficient network bandwidth in an HADR environment is crucial to performance. The general rule is that the maximum network shipping rate between the systems should always be greater that the logging rate. You can use tools such as the HADR simulator (http://www.ibm.com/developerworks/wikis/display/data/HADR_sim) to determine the maximum network shipping rate between systems.

When moving from single to multiple standbys, in addition to log data being shipped from the primary to the principal standby, log data is also shipped from the primary to the auxiliary standbys. As a result, the logging rate is doubled with two standbys and tripled with three standbys. The network must be able to support simultaneous log shipping from the primary to all standbys and the combined maximum network shipping rate should be greater than the combined logging rate to achieve desired performance.

To estimate the maximum network shipping rate in a multiple standby environment, multiple HADR simulator primary-standby pairs should be run simultaneously to simulate multiple log shipping paths. To avoid one pair consuming too many resources, it might be necessary to throttle the pair to an expected logging rate by using the simulator -target option. Because auxiliary standbys must run in SUPERASYNC synchronization mode, use the HADR simulator -rcu option (SUPERASYNC mode is limited to remote catchup state log shipping) to simulate the shipping rate for each primary-auxiliary standby pair.

5 Configuring multiple standby

Configuring a database for multiple standbys is similar to configuring a database for single standby. Some HADR-related database configuration parameters (such as `hadr_syncmode`) have different behavior in multiple standby mode, but the main difference is the use of a new database configuration parameter, `hadr_target_list`, to define a list of standby databases.

All databases that participate in an HADR multiple standby environment must set the `hadr_target_list` parameter. This parameter defines the list of standby databases. The first entry specifies the principal standby. If the database is currently a standby, it defines the databases that it will accept as standbys when it becomes a primary. A primary and a standby must include each other in their `hadr_target_list` parameter value. This setting guarantees that when they switch roles, the old primary remains in the system as a standby of the new primary.

5.1 hadr_target_list

The `hadr_target_list` parameter value on a standby does not have to match the `hadr_target_list` parameter value on the primary, but if the `hadr_target_list` parameter value is set on the primary then it must also be set on the standby. This ensures that if the primary is configured in multiple standby mode, then so is the standby. If the `hadr_target_list` parameter value is set only on the standby or the primary but not on both, then the primary rejects a connection request from the standby database, and the standby shuts down with an invalid configuration error.

If a standby database is not listed in the `hadr_target_list` parameter value on the primary, or the primary is not listed in the `hadr_target_list` parameter value on the standby, the connection is rejected. Include all would-be standbys on `hadr_target_list` parameter value on a standby. This configuration ensures that when the standby becomes a primary it is ready to communicate with all the applicable standbys. In a symmetrical setup, the `hadr_target_list` parameter value on each database lists all other databases. This setup has the advantage of simplicity. But you should configure your system according to your business requirements.

5.2 Syntax and IP version compatibility

The `hadr_target_list` parameter value is a pipe ('|' character) delimited list of remote HADR addresses. The first entry in this list is the principal standby. Each address is in the form of *host*:*port*. *Host* can be either a host name or an IP address, and *port* can be either a service name or a numeric TCP port number. The address is used to match the `hadr_local_host` and `hadr_local_svc` parameter values on the remote database. For matching purposes, host names are converted to IP address and service names converted to port number before the actual comparison is done. Therefore the original form does not need to be the same For example, host names can be specified in the `hadr_target_list` parameter while an IP address is used in the `hadr_local_host` parameter. Listing 1 shows a sample command that updates the `hadr_target_list` parameter with a value that has mixed IP/port and hostname/service name values.

Listing 1. Mixed IP/port and hostname/service name

```
UPDATE DB CFG FOR DB hadrdb USING hadr_target_list \
  "host-1:port1|192.168.10.2:2222|host-3:3333"
```

IPv6 is supported. An IPv6 address can be expressed as a host name where the name resolves to an IPv6 address. It can also be expressed in numerical format. If the numerical IPv6 format is used, the IP address must be enclosed in square brackets ([ ]). This is same convention that is used in URLs. Listing 2 shows a sample command that updates the `hadr_target_list` value with a hexadecimal IPv6 address and two host names that resolve to IPv6 addresses.

Listing 2. Hexadecimal IPv6 and hostnames that resolve to IPv6

```
UPDATE DB CFG FOR DB hadrdb USING hadr_target_list \
  "[FEDC:BA98:7654:3210:FEDC:BA98:7654:3210]:2222|host2:3333|host3:4444"
```

HADR requires that all addresses in the `hadr_local_host`, `hadr_remote_host`, and `hadr_target_list` values for a database be resolved to the same format (either all are IPv4 or all are IPv6). Numerical IP addresses have explicit IPv4/IPv6 designations. Host names are checked against DNS. A hostname could be resolved to both IPv4 and IPv6. If all addresses can be resolved to both IPv6 and IPv4, IPv6 is chosen as the database HADR IP version.

Best practice

When two HADR databases connect, they must have the same HADR IP version, otherwise the connection fails. In the case of ambiguous host names, use a numeric address to force a database onto a specific IP version.

5.3 Configuration example

Table 1 illustrates how to configure the HADR system shown in figure 1. The following hosts and ports are dedicated for each of the databases in figure 1.

Table 1. HADR configuration

HADR hostname	HADR service port	DB2 Instance	Notes
beaverton-a	2222	dbinst1	Primary HADR host located in building A on the Beaverton campus
beaverton-b	3333	dbinst2	Principal HADR standby host located in building B on the Beaverton campus
portland	4444	dbinst3	Auxiliary HADR standby host located 10 km away, on the Portland campus
toronto	5555	dbinst4	Auxiliary HADR standby host located 3400 km away, on the Toronto campus

Configuring the primary database on host beaverton-a

Listing 3 shows sample commands that configure HADR on the primary database.

Listing 3. Configuring HADR on the primary

```
UPDATE DB CFG FOR DB hadrdb USING hadr_target_list \
  "beaverton-b:3333|portland:4444|toronto:5555"

UPDATE DB CFG FOR DB hadrdb USING hadr_local_host  "beaverton-a"
UPDATE DB CFG FOR DB hadrdb USING hadr_local_svc   "2222"

UPDATE DB CFG FOR DB hadrdb USING hadr_syncmode    "SYNC"
UPDATE DB CFG FOR DB hadrdb USING hadr_peer_window "300"

UPDATE DB CFG FOR DB hadrdb USING hadr_remote_host "beaverton-b"
UPDATE DB CFG FOR DB hadrdb USING hadr_remote_svc  "3333"
UPDATE DB CFG FOR DB hadrdb USING hadr_remote_inst "dbinst2"
```

The principal standby here is beaverton-b. The synchronization mode and peer window configured on the primary apply only to the connection between the principal standby and the primary. The remote host, service, and instance on the primary database must point to the principal standby (beaverton-b:3333 in this case) which is always the first element in the `hadr_target_list` value. If not, HADR automatically

reconfigures the `hadr_remote_host`, `hadr_remote_svc`, and `hadr_remote_inst` values accordingly.

Configuring the principal standby database on host beaverton-b

Listing 4 shows sample commands that configure HADR on the principal standby database.

Listing 4. Configuring HADR on the principal standby

```
UPDATE DB CFG FOR DB hadrdb USING hadr_target_list \
          "beaverton-a:2222|portland:4444|toronto:5555"

UPDATE DB CFG FOR DB hadrdb USING hadr_local_host   "beaverton-b"
UPDATE DB CFG FOR DB hadrdb USING hadr_local_svc    "3333"

UPDATE DB CFG FOR DB hadrdb USING hadr_syncmode     "NEARSYNC"
UPDATE DB CFG FOR DB hadrdb USING hadr_peer_window  "100"

UPDATE DB CFG FOR DB hadrdb USING hadr_remote_host  "beaverton-a"
UPDATE DB CFG FOR DB hadrdb USING hadr_remote_svc   "2222"
UPDATE DB CFG FOR DB hadrdb USING hadr_remote_inst  "dbinst1"
```

The synchronization mode (NEARSYNC) and peer window (100) configured on the principal standby apply only when it becomes the primary. The connection mode between the primary (beaverton-a) and principal (beaverton-b) is SYNC with a peer window of 300, but once a takeover is issued on beaverton-b, the synchronization mode between beaverton-b and beaverton-a would become NEARSYNC, with a peer window of 100 seconds. NEARSYNC has been chosen to illustrate that a primary and a standby can have different synchronization modes defined in the `hadr_syncmode` configuration parameter.

Configuring the first auxiliary standby database on host portland

Listing 5 shows sample commands that configure HADR on an auxiliary standby database.

Listing 5. Configuring HADR on an auxiliary standby

```
UPDATE DB CFG FOR DB hadrdb USING hadr_target_list \
          "beaverton-a:2222|beaverton-b:3333|toronto:5555"

UPDATE DB CFG FOR DB hadrdb USING hadr_local_host   "portland"
UPDATE DB CFG FOR DB hadrdb USING hadr_local_svc    "4444"

UPDATE DB CFG FOR DB hadrdb USING hadr_syncmode     "ASYNC"

UPDATE DB CFG FOR DB hadrdb USING hadr_remote_host  "beaverton-a"
UPDATE DB CFG FOR DB hadrdb USING hadr_remote_svc   "2222"
UPDATE DB CFG FOR DB hadrdb USING hadr_remote_inst  "dbinst1"
```

The `hadr_target_list` and `hadr_syncmode` are defined on standby, but are only applicable when host portland becomes the primary. In that case, the principal standby is host beaverton-a (first element in the `hadr_target_list` value defined earlier), and the HADR connection between portland and beaverton-a would be in ASYNC mode.

Configuring the first auxiliary standby database on host toronto

Listing 6 shows sample commands that configure HADR on an auxiliary standby database.

Listing 6. Configuring HADR on an auxiliary standby

```
UPDATE DB CFG FOR DB hadrdb USING hadr_target_list \
          "beaverton-a:2222|beaverton-b:3333|portland:4444"

UPDATE DB CFG FOR DB hadrdb USING hadr_local_host   "toronto"
UPDATE DB CFG FOR DB hadrdb USING hadr_local_svc    "5555"

UPDATE DB CFG FOR DB hadrdb USING hadr_syncmode     "SUPERASYNC"

UPDATE DB CFG FOR DB hadrdb USING hadr_remote_host  "beaverton-a"
UPDATE DB CFG FOR DB hadrdb USING hadr_remote_svc   "2222"
UPDATE DB CFG FOR DB hadrdb USING hadr_remote_inst  "dbinst1"
```

The `hadr_target_list` and `hadr_syncmode` are defined on the standby, but are only applicable when host toronto becomes the primary. In that case, the principal standby would be host beaverton-a, and the HADR connection between toronto and beaverton-a would be in SUPERASYNC mode.

5.4 Effect of configuration change

In previous releases, a database had to be deactivated, and then reactivated in order to apply updated HADR configuration parameters. Starting in Version 10.1, the following HADR configuration parameters are applied whenever HADR is activated, whether by database activation, or by **START HADR** command. These configuration parameters are semidynamic:

- `hadr_target_list`
- `hadr_replay_delay`
- `hadr_spool_limit`
- `hadr_local_host`
- `hadr_local_svc`
- `hadr_peer_window`
- `hadr_remote_host`
- `hadr_remote_inst`
- `hadr_remote_svc`
- `hadr_syncmode`
- `hadr_timeout`

These semidynamic parameters enable a primary database to remain online and available during initial HADR setup and subsequent reconfiguration. For example, issuing a **STOP HADR** command, followed by a **START HADR** command on the primary picks up changes made while the database is online. The **STOP HADR** and **START HADR** commands cannot be issued dynamically on a standby database, so database deactivation and reactivation is still required to pick up HADR configuration changes on a standby.

The `hadr_target_list` parameter is fully dynamic. See the *Adding and dropping standby targets* section for details.

5.5 Effective versus configured synchronization mode

HADR supports a variety of log shipping synchronization modes to balance performance and data protection. The supported modes are:

- SYNC: Log write on primary requires replication to the persistent storage on the standby.

- NEARSYNC: Log write on primary requires replication to the memory on the standby.
- ASYNC: Log write on primary requires a successful send to standby (receive is not guaranteed).
- SUPERASYNC: Log write on primary has no dependency on replication to standby

In an HADR single standby configuration, the primary and standby must have the same `hadr_syncmode` configuration parameter value. However, in multiple standby mode this parameter behaves differently. In multiple standby mode, the `hadr_syncmode` parameter on the primary defines the synchronization mode for the connection between the primary and its principle standby. Auxiliary standbys always use SUPERASYNC mode. On the standbys, the defined `hadr_syncmode` value is the synchronization mode that the database uses for its principal standby when it becomes the primary through a takeover operation.

The primary and standby can have different values configured for the `hadr_syncmode` parameter. In multiple standby mode, this configuration does not result in a configuration error rejection when a standby connects to a primary. Upon connection, the primary instructs the standby what synchronization mode to use. For the principal standby, the primary tells it to use the synchronization mode defined by the `hadr_syncmode` parameter on the primary. For any auxiliary standby, the primary always tells it to use SUPERASYNC. Thus on the standby, its effective synchronization mode can be different from its `hadr_syncmode` configuration parameter.

The configured `hadr_syncmode`, the mode that is used for its principal standby when the database operates as the primary, can be viewed through the example command in listing 7, or any configuration-oriented interface:

Listing 7. Determining the configured synchronization mode

```
GET DB CFG FOR hadrdb
```

The effective HADR synchronization mode, that is the mode that is currently being used for an HADR log shipping channel, can be viewed through the example command in listing 8, or any HADR-oriented interface such as the MON_GET_HADR table function:

Listing 8. Determining the effective synchronization mode

```
db2pd -db hadrdb -hadr
```

5.6 Effective versus configured peer window

The `hadr_peer_window` parameter behaves similar to the `hadr_syncmode` parameter. It defines the peer window for the connection between the primary and its principal standby. Auxiliary standbys are always in SUPERASYNC mode, so peer window does not apply. The configured `hadr_peer_window` parameter on a standby database is not used until it becomes a primary, and the effective peer window for a principal standby is dictated to it by the primary during the HADR connection.

5.7 Automatic reconfiguration for `hadr_remote_host`, `hadr_remote_svc` and `hadr_remote_inst`

In multiple standby mode, on the primary, the `hadr_remote_host`, `hadr_remote_svc`, and `hadr_remote_inst` parameters point to the principal standby. On a standby, these point to the primary database. For ease of use, HADR automatically updates these parameters in the following scenarios:

- **Automatic redirection**
  When a primary is not connected to a standby listed in its `hadr_target_list` value, it periodically tries to reach that standby through the address listed in the `hadr_target_list` value. Both primary and standby databases listen on their localhost and local services address (defined by their `hadr_local_host` and `hadr_local_svc` parameter values) for incoming TCP connections. If the primary is able to reach the misconfigured standby, it sends the standby its `hadr_local_host` value, `hadr_local_svc` value, and instance name. When a standby receives this redirection message, it first authenticates the primary and then automatically updates its `hadr_remote_host`, `hadr_remote_svc`, and `hadr_remote_inst` parameter values to point to the primary, and then connects to the primary using the updated address.
- **During takeover**
  During takeover (forced and not forced), `hadr_remote_host`, `hadr_remote_svc`, and `hadr_remote_inst` on all databases are automatically updated to reflect the new topology and their roles. This greatly simplifies DBA work. A simple takeover command executed on any standby is all that is needed to complete a takeover and have all standbys find and reconnect to the new primary. Automatic redirection is the method used to redirect standbys to the new primary. If a standby is offline at the time of takeover, it is automatically redirected to the new primary when it comes back online. This automatic redirection also applies to primary reintegration, where the old primary is started as standby after a failover. Some concrete examples of automatic reconfiguration are given in the *Takeover* section.
- **During start up of primary**
  When an HADR primary starts up, if the `hadr_remote_host` or `hadr_remote_svc` values do not match the principal standby address defined by the first entry in the `hadr_target_list` value, then the `hadr_remote_host` value, `hadr_remote_svc` value, or both values are automatically updated to match be the principal standby. This automatic update is not applicable to a standby. During standby startup, HADR checks only whether the `hadr_remote_host`:`hadr_remote_svc` value pair is contained in `hadr_target_list`. If the primary is not included in the `hadr_target_list` value then an error is returned.
- **On connection**
  During an HADR connection, the primary and standby databases exchange instance names. If the `hadr_remote_inst` value (on either primary or standby) does not match the actual instance name of the remote database, it is automatically updated. You should still configure the correct instance name during HADR setup because some objects such as Tivoli System Automation for Multiplatforms (SA MP) cluster manager resource names are constructed using remote instance names. Since automatic update of the database configuration parameters does not happen until the primary and standby make a connection, SA MP might use the wrong name and SA MP can behave unexpectedly.

6 Setting up a multiple standby system

This procedure covers two cases:

- A database does not have HADR set up
- A single standby HADR system is already set up

The difference between the two cases is small. For the single standby case, you just need to configure the `hadr_target_list` value for it to become a multiple standbys system.

Before you begin:

- Determine the host name or host IP address (to be used for the `hadr_local_host` setting), service name or port number (to be used for the `hadr_local_svc` setting) of all participating databases.
- Determine the target list for each database.
- Determine the synchronization mode and peer window for the principal standby for each database in the event that the database becomes the primary.
- Determine the setting for the `hadr_timeout` configuration parameter; this parameter must have the same value on all databases.
- Determine if the network path between the primary and each standby can support the log shipping rate. Upgrade if necessary.
- Determine if the network interface on the primary database host can support the aggregated log shipping flow for all standbys. Upgrade if needed.

The main difference between a single standby system and a multiple standby system is that the `hadr_target_list` value is set on all participating databases on a multiple standby system. On a single standby system, it is not set on either the primary or standby. It is "all or nothing". If only one database has its `hadr_target_list` value set during a primary-standby connection, the standby database is shut down because it is incompatible with the primary. The primary is kept online to minimize outages. As a special case, a multiple standby system can have only one standby. But the system still behaves differently from a single standby system. Multiple standby features such as automated configuration will be functional. HADR behavior is determined by whether the `hadr_target_list` parameter is set, not the number of standbys. A multiple standby system with one standby can prove useful because it allows you to easily add and drop standbys.

When converting a single standby system, create and configure the additional standbys. Then reconfigure the existing primary and standby to multiple standby mode. Finally start HADR on all databases to complete the multiple standby setup. By keeping the single standby system functioning until the final steps, you minimize the window where the primary is not under HADR protection.

The procedure consists of the following steps. The primary database is assumed to be online at the start of the procedure and stays online throughout the procedure. There is no service downtime.

1. Create new standby databases from a backup image taken on the primary database or a split mirror of the primary database. This step is identical to creating a standby for single standby systems.
2. Configure each of the new standby databases as follows:
    a. Set the `hadr_local_host` and `hadr_local_svc` configuration parameters to the local HADR address.

    b. Set the `hadr_remote_host`, `hadr_remote_svc`, and `hadr_remote_inst` configuration parameters to point to the primary database.
    c. Set the `hadr_timeout` configuration parameter, with the same setting on all of the databases.
    d. Set the `hadr_target_list` configuration parameter, as previously planned.
    e. Set the `hadr_syncmode` and `hadr_peer_window` configuration parameters for the intended principal standby of the database.
    f. Set any other HADR-specific parameters such as `hadr_spool_limit` or `hadr_replay_delay`, depending on your desired setup.
3. If converting a single standby system, reconfigure the original standby using instructions from step 2. Keep the database online during this step. The new configuration does not take effect until deactivation and reactivation.
4. Configure the primary as follows. Keep the database online during this step. If you are converting a single standby system, update a parameter only if it actually needs to change. The new configuration does not take effect until HADR is stopped then restarted.
    a. Set the `hadr_local_host` and `hadr_local_svc` to the local HADR address.
    b. Set the `hadr_remote_host`, `hadr_remote_svc`, and `hadr_remote_inst` configuration parameters to point to the principal standby database.
    c. Set the `hadr_timeout` configuration, with the same setting as on all of the databases.
    d. Set the `hadr_target_list` configuration parameter, as previously planned.
    e. Set the `hadr_syncmode` and `hadr_peer_window` configuration parameters for the principal standby.
    f. Set any other HADR-specific parameters such as `hadr_spool_limit` or `hadr_replay_delay`, depending on your desired setup.
5. If converting a single standby system, stop HADR on the primary using the **STOP HADR ON DB *dbname*** command. The primary that is still running in single standby mode must be stopped before any standby is started. Otherwise the standby is found to be incompatible and shut down when the standby attempts to connect to the primary.
6. If you are converting a single standby system, deactivate the original standby using the **DEACTIVATE DB *dbname*** command.
7. Start all standbys, including the original standby by using the **START HADR ON DB *dbname* AS STANDBY** command. The standbys should be started before the primary because when the primary starts, it connects to the standbys to make sure that no standby has become a new primary. A situation with two active primaries is sometimes called "split brain" or "dual primary".
8. Start HADR on the primary by using the **START HADR ON DB *dbname* AS PRIMARY** command.

All the standbys should connect to the primary within seconds. On the primary database, issue the **db2pd** command with the -hadr option or query the MON_GET_HADR table function to monitor status of all standbys. On a standby, only the standby itself and the primary are shown in monitoring information. Other standbys are not visible.

All standbys should show "connected" status. Each standby goes through HADR state transition independently. They first scan through locally available logs (local catchup

state), then request more logs to be shipped from the primary through the HADR connection (remote catchup and peer state).

In step 8, if the principal standby cannot connect to the primary, the **START HADR** command fails after trying for the number of seconds specified in the `hadr_timeout` parameter. Should this happen, check the configuration on the primary and standby, check network status, correct any problems, and then retry. The primary database stays online all through the procedure, even if the **START HADR** command fails. Starting in DB2 Version 10.1, the primary database applies the latest HADR configuration each time the **START HADR** command is issued. There is no need to deactivate the database and then reactivate it to pick up HADR configuration change.

7 Adding and dropping standby targets

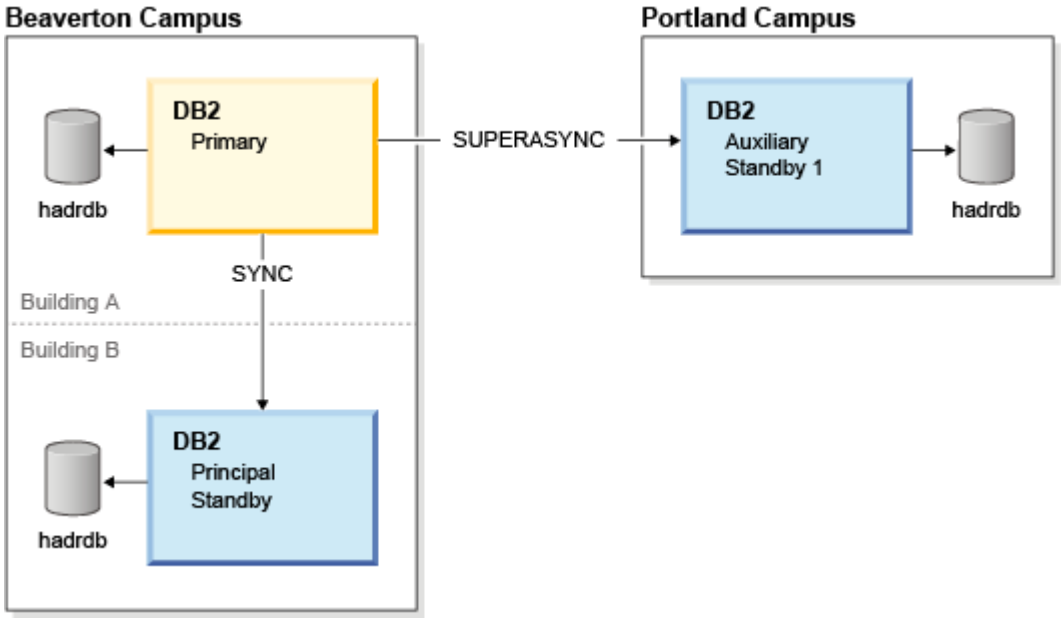
If HADR is set up with multiple standby enabled, then you can add or drop standby targets dynamically. An HADR system is defined as multiple standby enabled when the `hadr_target_list` configuration parameter is set and has at least one standby target defined. For a dynamic update to take place, the database has to be active and must have at least one connection to it. On a standby database, reads on standby must be enabled to allow connections. If the update is not dynamic, it takes effect the next time HADR is activated.

7.1 Adding a standby target dynamically

To add a standby target, simply update the `hadr_target_list` parameter to include the new standby target.

For example, assume that you are starting with a primary located in building A on the Beaverton campus (beaverton-a) and two standby targets in building B at Beaverton and at the Portland campus (beaverton-b and portland) as shown in Figure 2.

Figure 2. Before adding an additional auxiliary standby server



Now suppose that you want to add a third standby at the Toronto campus (toronto). To add the new standby target, create, configure, and activate the new standby

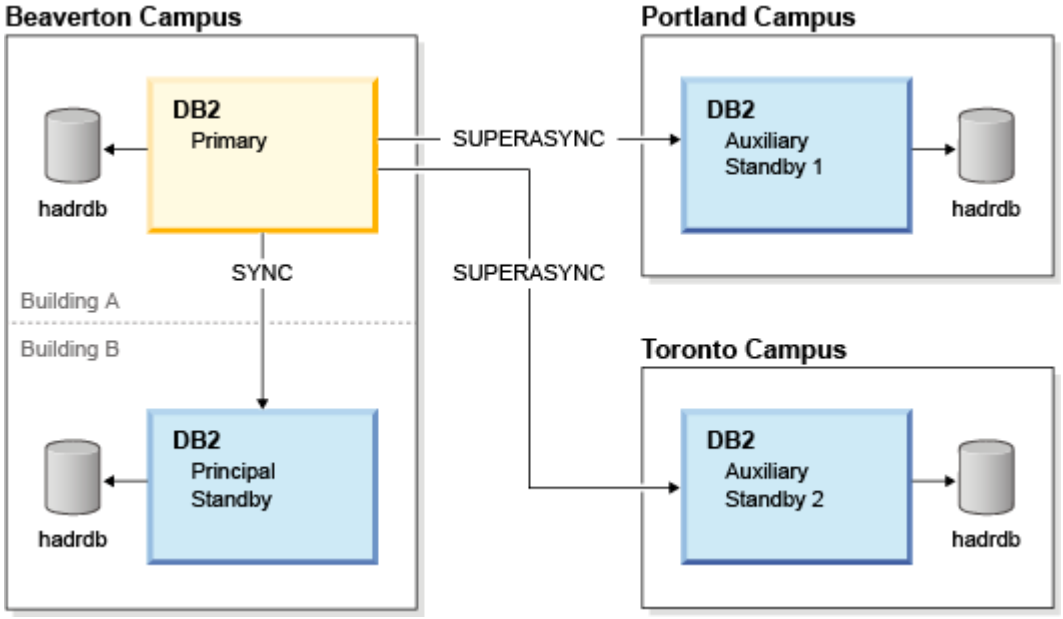
target. Then add it to the `hadr_target_list` parameter on the primary by issuing the example command in listing 9 on the primary database (beaverton-a).

Listing 9. Adding a standby target

```
UPDATE DB CFG FOR hadrdb USING hadr_target_list \
  "beaverton-b:3333|portland:4444|toronto:5555"
```

After the update completes, toronto becomes a valid standby, as shown in figure 3.

Figure 3. After adding an additional auxiliary standby server



Dynamically updating the `hadr_target_list` parameter cannot change the principal standby target. Thus when adding additional standbys, you should keep the original principal standby (the first entry in the list) as the principal (by keeping it as the first entry). During dynamic update, if the principal standby is changed, you get the messages in listing 10 informing you that the on-disk configuration file was updated successfully, but the change is not taking effect immediately. The change takes effect on the next activation of HADR (through the **START HADR** command or **ACTIVATE DB** command).

Listing 10. Expected messages after a dynamic update

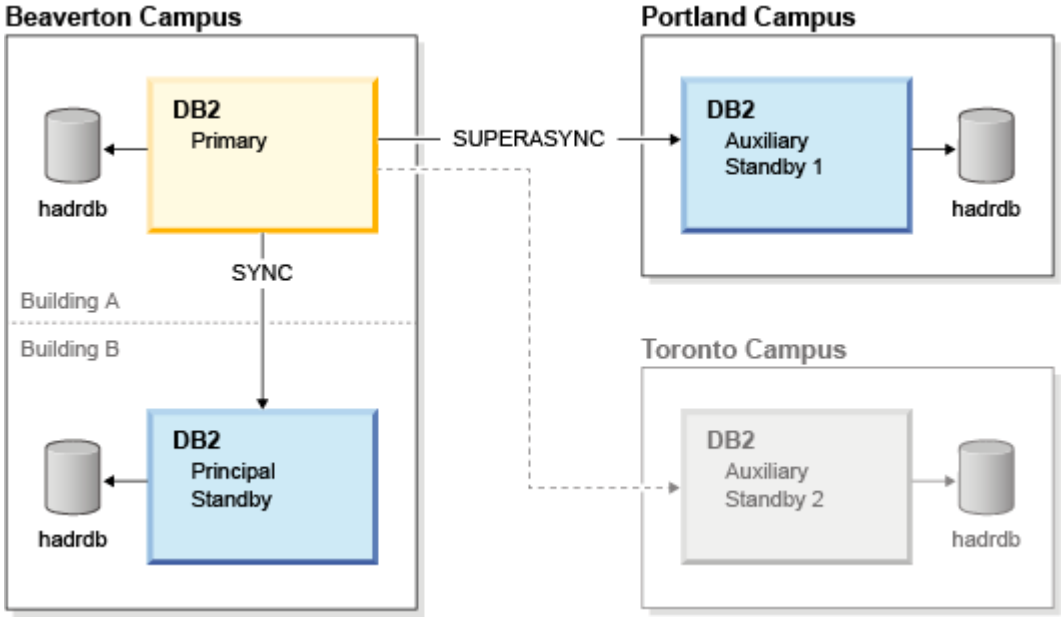
```
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
SQL1363W  Database must be deactivated and reactivated before the changes to one or more
of the configuration parameters will be effective.
```

7.2 Dropping a standby target dynamically

To remove a standby target, simply remove the standby from the `hadr_target_list` parameter value on the primary.

For example, assume that you have the HADR configuration shown in figure 3, with a primary (beaverton-a) and three standby targets (beaverton-b, portland, and toronto), where toronto is currently not connected to the primary. This configuration is shown in figure 4.

Figure 4. Before dropping an auxiliary standby



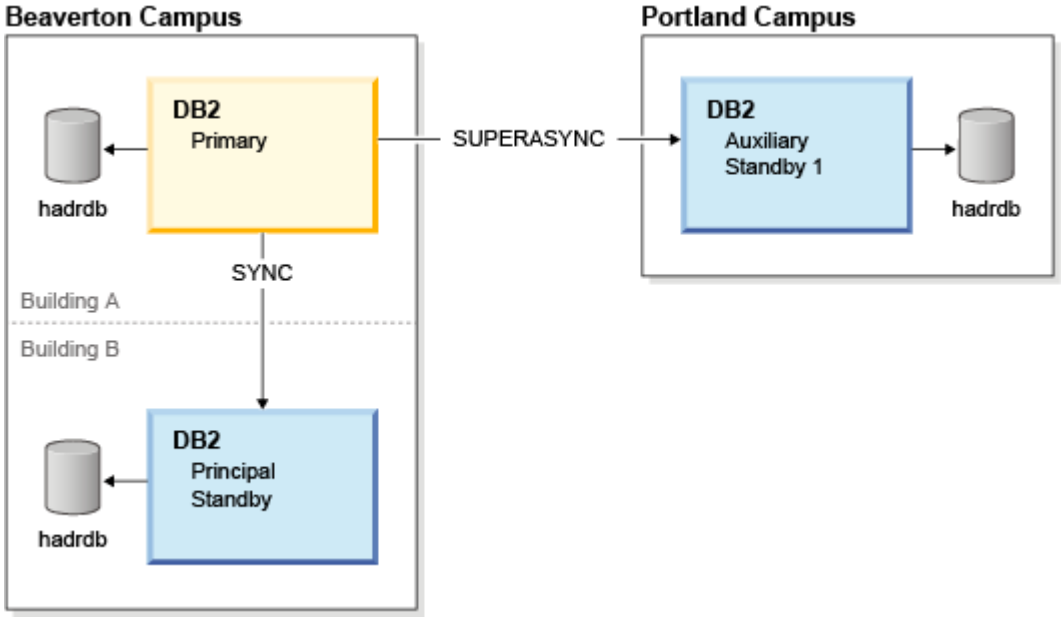
To drop the toronto standby, remove the toronto:5555 entry from the `hadr_target_list` parameter on the primary, as shown in listing 11.

Listing 11. Removing a standby target

```
UPDATE DB CFG FOR hadrdb USING hadr_target_list beaverton-b:3333|portland:4444
```

As a result of the **UPDATE** command, the HADR system changes as shown in figure 5.

Figure 5. After dropping an auxiliary standby



You cannot not remove the principal standby, which is defined as the first entry in the `hadr_target_list` parameter. If you remove the principal standby target, you receive a warning SQL1363W saying the update is not being performed dynamically. Upon the next activation of HADR or the database, the new `hadr_target_list` parameter values are implemented.

You cannot dynamically remove a standby target that is still connected to the primary. If you try to remove such a standby target, you receive a warning SQL1363W stating that the update was not performed dynamically and only the on-disk configuration file was changed. To remove a connected standby target dynamically, deactivate the standby first.

8 Takeover

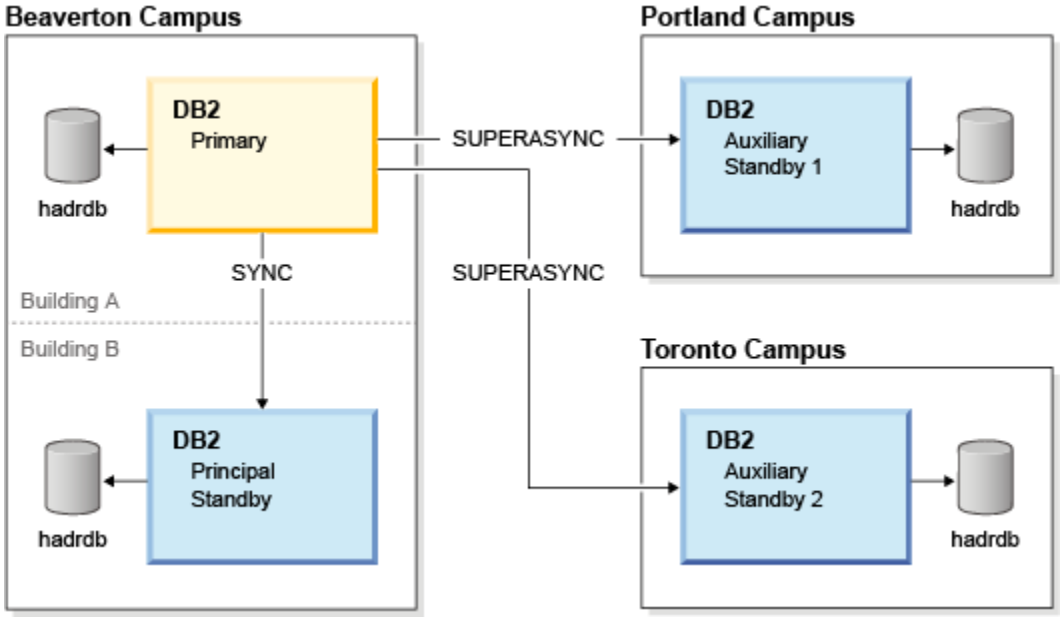
A standby can become the primary through an HADR takeover operation. A takeover can be issued on a principal or an auxiliary standby. After a takeover, the new primary automatically redirects standby targets that are in the target list on the new primary. The `hadr_remote_host`, `hadr_remote_svc`, and `hadr_remote_inst` configuration parameters on the standby are automatically updated to point to the new primary. If a standby is not included in the `hadr_target_list` parameter on the new primary, then that standby is considered to be orphaned and cannot connect to the new primary.

8.1 Automatic redirection of standby targets after takeover

After a takeover (forced or not forced) is completed successfully, the new primary redirects any standby target that is not already connected to it. This redirection is done by the primary sending a redirection message to each standby that is not connected to it. When a standby receives a redirection message, if it is not already connected to another primary, it confirms that the message is coming from a valid primary listed in its `hadr_target_list`. It updates its `hadr_remote_host`, `hadr_remote_svc`, and `hadr_remote_inst` configuration parameters to point to the primary, and then connect to the primary.

For example, assume you have an HADR system with a primary beaverton-a and standby targets beaverton-b, portland, and toronto, as shown in figure 6.

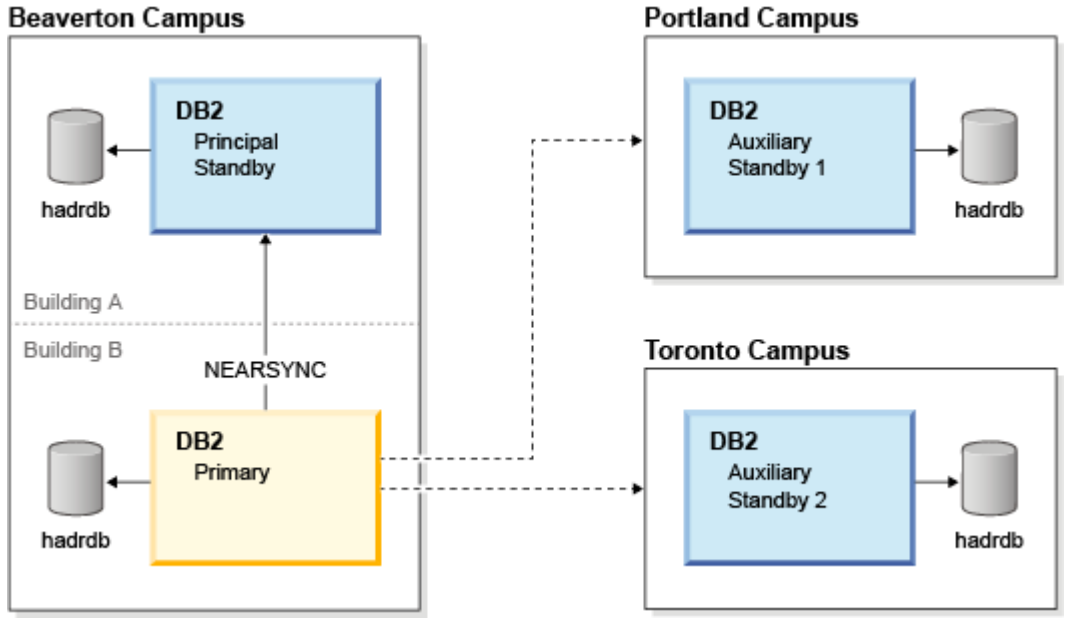
Figure 6. Initial state before takeover



Takeover is issued on principal standby host beaverton-b, resulting in beaverton-b becoming the primary, and beaverton-a, portland, and toronto becoming the standby targets.

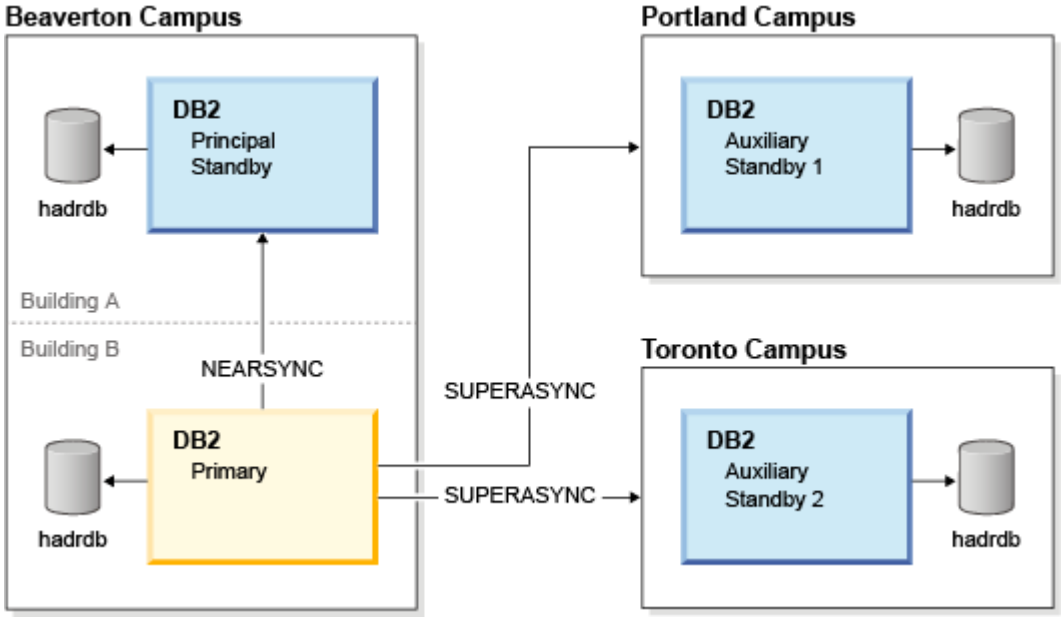
At the end of the takeover, beaverton-a is connected to beaverton-b, but portland and toronto must be informed of the change in primary. This notification is done by beaverton-b sending redirection requests to portland and toronto, as shown in figure 7.

Figure 7. Following takeover by principal standby



After portland and toronto accept and process the redirection requests, their `hadr_remote_host`, `hadr_remote_svc`, and `hadr_remote_inst` configuration parameters are automatically updated to reflect beaverton-b as the primary, and they then connect to the new primary. This configuration is shown in figure 8.

Figure 8. Following automatic redirection of the other standbys.



The synchronization mode between beaverton-a and beaverton-b changed from SYNC to NEARSYNC. The mode changes because the primary determines that the synchronization mode of the standby targets, and the `hadr_syncmode` setting on beaverton-b is NEARSYNC. At connection time, the auxiliary standbys are directed by beaverton-b to use SUPERASYNC, and the principal standby is told to use NEARSYNC. Details about automatic reconfigurations are in the next section.

8.2 Automatic reconfiguration after takeover

In multiple standby mode, the `hadr_remote_host`, `hadr_remote_svc`, and `hadr_remote_inst` configuration parameters on a primary refer to the principal standby. On a standby, the `hadr_remote_host`, `hadr_remote_svc`, and `hadr_remote_inst` refer to the primary.

As a result of a takeover, a new primary is established and the `hadr_remote_host`, `hadr_remote_svc`, and `hadr_remote_inst` configuration parameters are automatically changed on both the standby targets and the new primary. On the new primary, the three parameters change to refer to the new principal standby, as defined by the first entry in the new primary's `hadr_target_list`. On the different standby targets (including the old primary once it is converted into a standby) the parameters change to refer to the new primary.

For example, assume that you have the following setup with a primary database (beaverton-a) and three standby targets (beaverton-b, portland, and toronto), as shown in figure 9.

Figure 9. Initial state before takeover

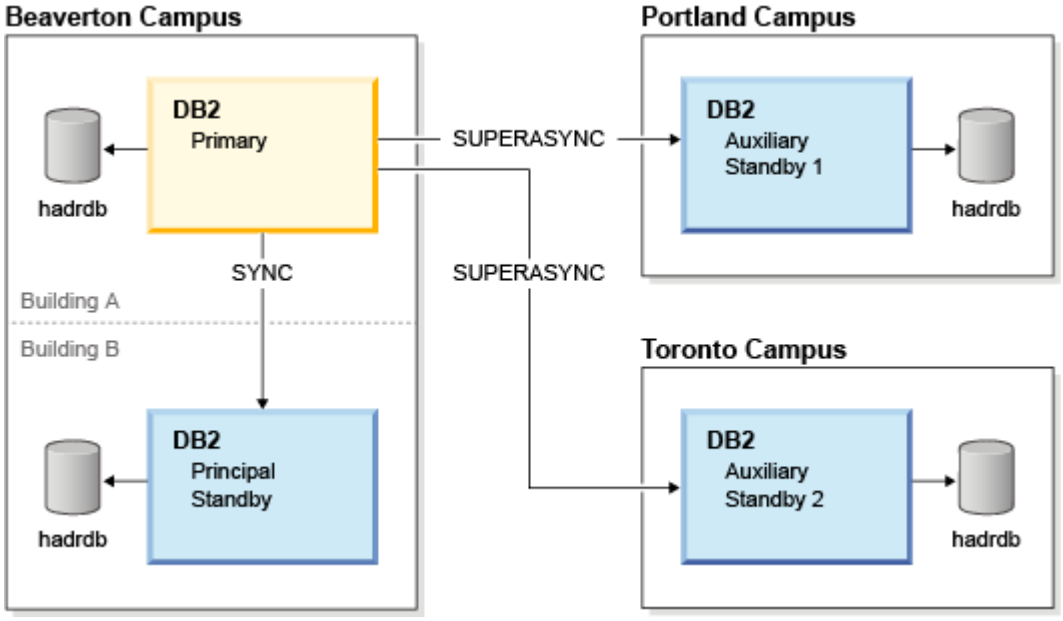


Table 2. HADR configuration before takeover

configuration parameter	host1 (beaverton-a)	host2 (beaverton-b)	host3 (portland)	host4 (toronto)
`hadr_target_list`	beaverton-b:3333 | portland:4444 | toronto:5555	beaverton-a:2222 | portland:4444 | toronto:5555	toronto:5555 | beaverton-a:2222 | beaverton-b:3333	portland:4444 | beaverton-a:2222 | beaverton-b:3333
`hadr_remote_host`	beaverton-b	beaverton-a	beaverton-a	beaverton-a
`hadr_remote_svc`	3333	2222	2222	2222
`hadr_remote_inst`	dbinst2	dbinst1	dbinst1	dbinst1
Effective sync mode	N/A	SYNC	SUPERASYNC	SUPERASYNC

Figure 10 shows the configuration after beaverton-b takes over.

Figure 10. After principal standby takeover and automatic redirection and reconfiguration

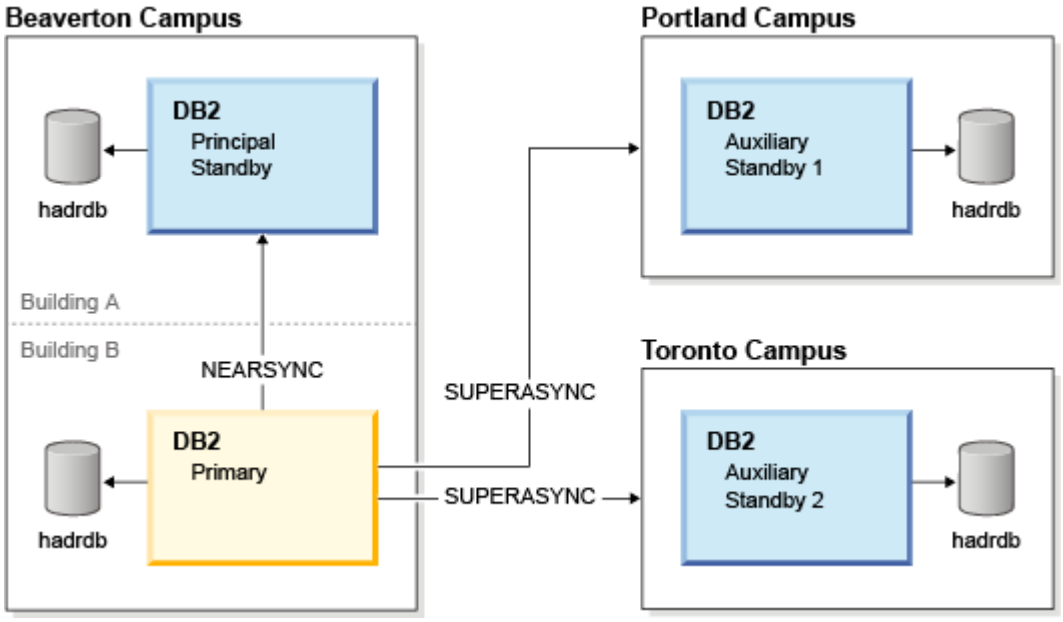


Table 3. Reconfiguration after takeover by beaverton-b

configuration parameter	host1 (beaverton-a)	host2 (beaverton-a)	host3 (portland)	host4 (toronto)
`hadr_target_list`	beaverton-b:3333 | portland:4444 | toronto:5555	beaverton-a:2222 | portland:4444 | toronto:5555	toronto:5555 | beaverton-a:2222 | Beaverton-b:3333	portland:4444 | beaverton-a:2222 | beaverton-b:3333
`hadr_remote_host`	beaverton-b	beaverton-a	beaverton-b	beaverton-b
`hadr_remote_svc`	3333	2222	3333	3333
`hadr_remote_inst`	dbinst2	dbinst1	dbinst2	dbinst2
effective sync mode	NEARSYNC	N/A	SUPERASYNC	SUPERASYNC

Imagine that after beaverton-b takes over, both buildings on the Beaverton campus burn down. This is a true disaster recovery scenario. A forced takeover is issued from the standby host portland at the Portland site. The effective synchronization mode of the principal standby toronto is determined by the `hadr_syncmode` configuration parameter defined on the primary host, portland. The resulting setup is showing in figure 11.

Figure 11. Loss of primary site

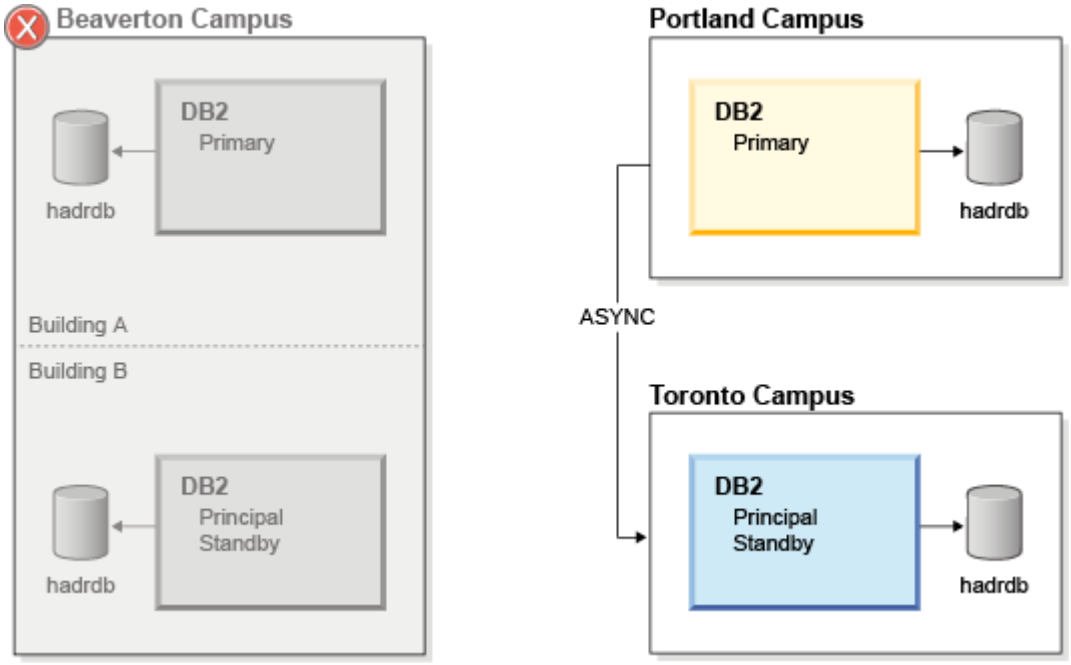


Table 4. Reconfiguration after takeover by portland (host1 and host2 are down)

configuration parameter	host1 (beaverton-a)	host2 (beaverton-b)	host3 (portland)	host4 (toronto)
`hadr_target_list`	beaverton-b:3333 | portland:4444 | toronto:5555	beaverton-a:2222 | portland:4444 | toronto:5555	toronto:5555 | beaverton-a:2222 | beaverton-b:3333	portland:4444 | beaverton-a:2222 | beaverton-b:3333
`hadr_remote_host`	N/A	N/A	Toronto	portland
`hadr_remote_svc`	N/A	N/A	5555	4444
`hadr_remote_inst`	N/A	N/A	dbinst4	dbinst3
effective sync mode	N/A	N/A	N/A	ASYNC

Beaverton-A and Beaverton-B come back online as shown in figure 12.

Figure 12. Beaverton site comes back online

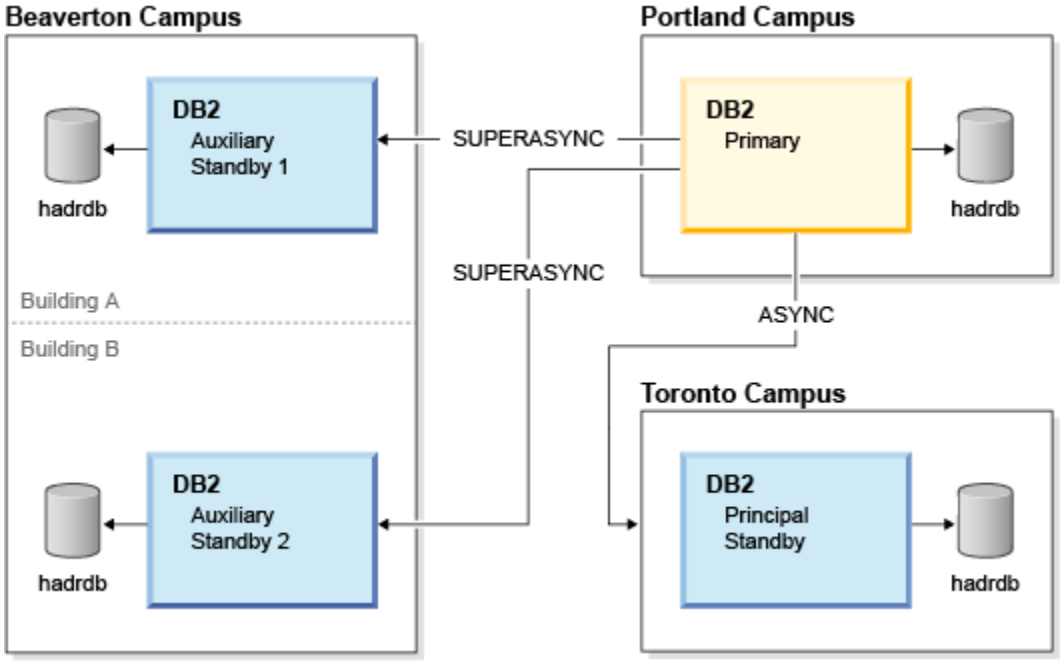


Table 5. Reconfiguration after host1 and host2 come back online

configuration parameter	host1 (beaverton-a)	host2 (beaverton-b)	host3 (portland)	host4 (toronto)
`hadr_target_list`	beaverton-b:3333 | portland:4444 | toronto:5555	beaverton-a:2222 | portland:4444 | toronto:5555	toronto:5555 | beaverton-a:2222 | beaverton-b:3333	portland:4444 | beaverton-a:2222 | beaverton-b:3333
`hadr_remote_host`	portland	Portland	Toronto	portland
`hadr_remote_svc`	4444	4444	5555	4444
`hadr_remote_inst`	dbinst3	dbinst3	dbinst4	dbinst3
effective sync mode	SUPERASYNC	SUPERASYNC	N/A	ASYNC

After takeover, the new primary determines the synchronization mode of all primary-standby pairs. That is, the `hadr_syncmode` setting of the new primary specifies the synchronization mode used for its principal standby, and SUPERASYNC is used for the auxiliary standbys.

If a standby is not online at the time of the takeover, the automated configuration change on that standby is still carried out, once the standby is started. The automatic reconfiguration logic updates the configuration on the standby to redirect it to the new primary. Upon startup, the standby might initially attempt to connect to the old primary using its old configuration. The connection should fail, unless you have a dual primary (or "split brain") scenario, where the old primary is also online. In a dual primary scenario, if a standby is listed in the `hadr_target_list` of old and new primaries, it can get redirected to either of the two primary databases. A split brain

confuses database clients and HADR standbys. To avoid having dual primaries, securely shut down the old primary before failing over to a standby. See the *Dual primary detection* section to see how DB2 can automatically detect a dual primary.

Automatic reconfiguration is only done on a standby that is listed in the `hadr_target_list` value on the new primary. If the new primary does not list a standby, the configuration of the standby is left unchanged (still pointing to the old primary). This standby cannot connect to new primary without a manually adding it to the `hadr_target_list` value of the new primary. If the old primary becomes a primary again at a later time, the orphaned standby can still rejoin the old primary as a standby.

8.3 Dual primary detection

Having more than one primary is known as a dual primary. A dual primary can happen if a takeover by force is used to create a new primary while the old primary continues operating. To reduce the likelihood of a dual primary, in single standby mode, the primary does not start unless it can establish a connection to the standby (as positive confirmation that the standby has not become a primary) or it is started using the **START HADR AS PRIMARY BY FORCE** command. In multiple standby mode, this behavior still applies and a primary does not activate unless it manages to communicate with the principal standby or is started by force.

Additionally, on startup in multiple standby mode, a primary contacts the auxiliary standbys to check if any of them has become a primary. If the starting primary finds another primary database, it fails to activate with SQL1768N reason code 19. Dual primary detection on auxiliary standbys is done only as best effort. If a standby cannot be reached within a short time, the primary starts.

9 Automation

9.1 Client reroute after takeover

There are multiple methods of rerouting client connections to a new primary after an HADR takeover. The method you choose depends upon your business requirements and your specific HADR configuration. One method is to use automatic client reroute (ACR) feature of DB2 databases. Another is to use the db2dsdriver.cfg file on the DB2 client to define client affinities (CA). Finally, you can use a virtual IP address for the primary HADR address.

Automatic client reroute

With the automatic client reroute (ACR) feature, a DB2 client application can recover from a loss of communication to a database. To configure client reroute, specify an alternate server location on the HADR primary database. This alternate database information is cached by the client during the connection process. In the event of a failure, the client can use this information to connect to the alternate location. Because only a single alternate location can be specified, only one standby target may be configured as the alternate database.

Best practice

In a multiple standby environment, configure the primary and the principal standby as the alternate location of each other, because the primary role is more likely to alternate between the two than to move to an auxiliary standby.

For example, if applications are connected to the primary instance beaverton-a you might configure ACR on beaverton-a so that in the event of a failover to beaverton-b clients would transparently re-connect to beaverton-b. To configure ACR for such a scenario, issue the example command in listing 12 on host beaverton-a.

Listing 12. Configuring ACR

```
UPDATE ALTERNATE SERVER FOR DB hadrdb USING HOSTNAME beaverton-b PORT 7777
```

The port here is the DB2 port for host beaverton-b, not the HADR port for beaverton-b.

Client affinity and db2dsdriver.cfg

Rerouting can also be accomplished by configuring client rerouting and affinity information in the db2dsdriver.cfg file on the DB2 client host. To enable client affinities, edit the db2dsdriver.cfg configuration file to define client affinities and to specify primary and alternate servers. A list of alternate servers and affinity lists may be defined. Because client affinity supports multiple alternate servers, all standbys in a multiple standby system can be listed as alternate servers of the primary. If any standby takes over, clients can be rerouted to the new primary. For more details, see the following DB2 Information Center topic for CA information: Client affinities for clients that connect to DB2 Database for Linux, UNIX, and Windows.

Using a virtual IP address for client reroute

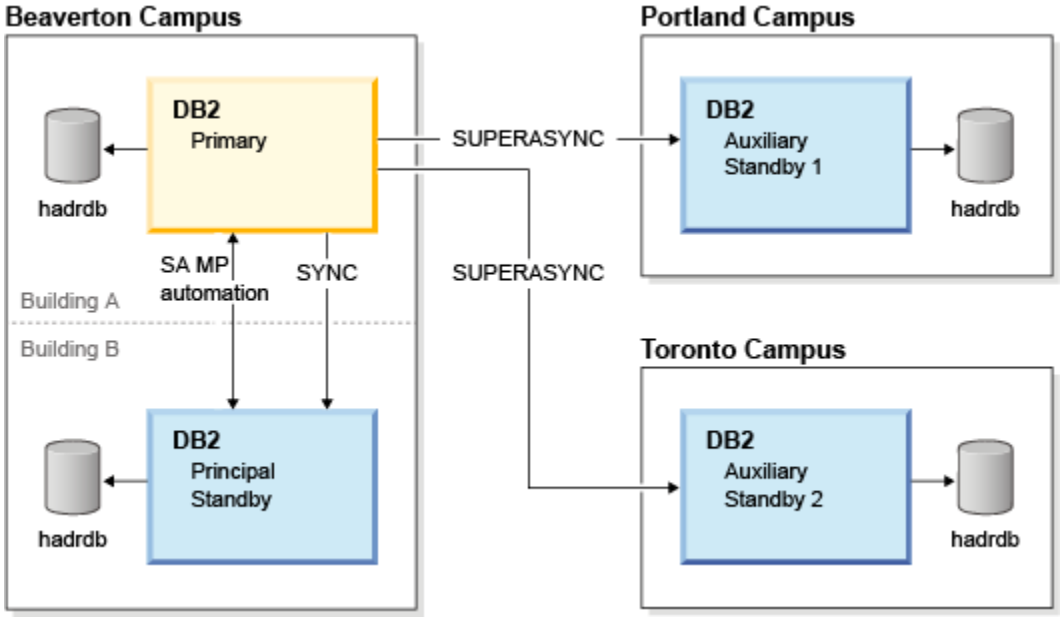
A virtual IP address is an IP address that can be dynamically moved from one physical machine to another. It is common to use a virtual IP address to make client applications transparent to the physical machine a server runs on. When the service a client connects to fails, the virtual IP address is moved to a different physical machine and the service is restarted on that new machine. The client then reconnects to the service through the virtual IP address. Since both the virtual IP address and the service are moved as part of failover, cluster manager software typically manages the movement of both the service and virtual IP address to the new machine. Virtual IP address configuration is supported as part of the setup and configuration of HADR with an integrated cluster manager using the db2haicu tool. Using a virtual IP address to implement client reroute is possible only when the primary and standby are on the same subnet.

9.2 Integrated cluster manager

Integration of HADR with IBM Tivoli System Automation for Multiplatforms (SA MP) cluster manager has been supported since DB2 Version 9.5. SA MP can be used to automatically detect primary database failures and to automate failover by issuing the **TAKEOVER BY FORCE PEER WINDOW ONLY** command on the standby. Integration with SA MP is supported only in SYNC and NEARSYNC modes, because these modes support "PEER WINDOW ONLY" takeover.

In multiple standby mode, SA MP is supported only between the primary and the principal standby. For example, suppose that you are running in the configuration shown in figure 1 and want to add SA MP automation between the HADR primary and principal standby running on hosts beaverton-a and beaverton-b. The resulting configuration is shown in figure 13.

Figure 13. Automation in a multiple standby environment



If the HADR primary fails and SA MP is configured, SA MP automatically performs takeover on the principal standby. After beaverton-b becomes the primary, beaverton-a becomes its principal standby. When the principal standby defines the primary as its own principal standby, no reconfiguration is required after takeover. If however the old primary is not the principal standby for the new primary (for example, if beaverton-b defines portland as its principal standby), then you must manually reconfigure SA MP automation. For example, if you want automatic failure detection between a new primary beaverton-b, and its new principal standby portland, then you need to reconfigure SA MP after beaverton-b perform takeover.

In the event that there is a failure of the principal standby, there is no action from SA MP. This behavior is the same as in single standby mode which takes no action on standby failure. For principal standby failures you must manually start the machine (if it went down), start DB2, and issue the **START HADR AS STANDBY** or **ACTIVATE DATABASE** command to start the principal standby.

In the event that SA MP executes a failover to the principal standby, the other standbys are redirected to the new primary through automatic reconfiguration, which is described in the Takeover section.

Configuring a clustered environment with multiple standby

Configuring cluster controlled automation of HADR with SA MP in a multiple standby environment is similar to a single standby environment. For multiple standbys, use the principal standby as the "standby" database when configuring a clustered environment with the **db2haicu** command. The following resources describe configuring a single standby environment:
"Configuring a clustered environment using DB2 High Availability Instance Configuration Utility (db2haicu)"

"Automated Cluster Controlled HADR (High Availability Disaster Recovery) Configuration Setup using the IBM DB2 High Availability Instance Configuration Utility (db2haicu)"

Manual takeover when SA MP is configured

Before issuing a manual forced or graceful takeover on an auxiliary standby in an SA MP controlled environment, you should disable SA MP on the primary and the principal standby. This can be done using the **db2haicu -disable** command on both the primary and the principal standby hosts. This command prevents SA MP from issuing an automatic failover to the principal standby, causing a dual primary situation. After a new primary-principal standby pair is formed, you can choose to create or re-enable SA MP on the new pair.

To verify that you have disabled SA MP monitoring, issue the **lssam** command on both the principal standby and the primary. The state of the resource group that contains the HADR database should be "Control=SuspendedPropagated" when SA MP is disabled.

Disabling SA MP on the old primary alone is not enough to prevent automatic failover to the old principal standby. However, in certain situations it is not possible to disable SA MP on the principal standby host. For example, if the principal standby has had a power failure SA MP cannot be disabled. In this scenario, the old principal standby host must be kept offline or be isolated from database clients (by means such as disabling the client-server network), until SA MP can be disabled or the database is dropped.

In forced takeover to an auxiliary standby, the old primary host should be kept offline or fenced off because if SA MP remains enabled, it automatically attempts to start the old primary database when the machine restarts. Even if SA MP is disabled, the old primary can still attempt to start upon client connection. In such a scenario dual primary detection might recognize that there are two primary databases, and deactivate the old primary. See the Dual primary detection section for details.

10 Log archiving considerations

10.1 Configuring log archiving on all databases

To use log archiving with DB2 HADR, configure both the primary database and all the standby databases for automatic log retrieval capability from all log archive locations.

10.2 Log file management on standby

The standby database automatically manages log files in its local log path. It writes the logs received from primary to its log path. The standby does not delete a log file from its local log path unless it has been notified by the primary database that the primary database has archived it. This behavior provides added protection against the loss of log files. If the primary database fails before the log file is safely stored in the archive, the standby database would ensure the log file is archived. If both the logarchmeth1 and logarchmeth2 configuration parameters are in use, the standby database does not recycle a log file until the primary database has archived it using both methods.

10.3 Shared versus separate log archive

Only a primary database can archive log files. When the primary and standby databases have different archive locations, logs are archived only to the archive location of the primary database. In the event of a takeover, the archive location changes from the old primary to the archive on the new primary. This situation can result in logs being scattered among different archive locations; with the exception that the new primary database might archive a few log files following a takeover that the original primary database had already archived. A shared archive, which means that one database can access log files archived by another database, is preferred because all the log files are archived to a single location.

Several operations need to retrieve archived log files. These operations include:

- **ROLLFORWARD** command;
- HADR primary database retrieving log files to send to the standby database in remote catch up;
- HADR standby database for local catch-up;
- Replication programs (such as Q Replication) reading logs.

The advantage of using a shared archive is that these operations can to retrieve any log file, no matter which machine generated or archived that log file. Not using a shared archive means that there is a chance that these operations might fail to locate a log file it needs. These operations can only continue after manually copying over of the required log files from one log archive location to another (or to the overflow log path if it is configured).

When using multiple standbys with a shared archive, another standby might be far behind during a takeover causing the new primary to not have the log files needed by the other standby. In this case, you must locate the needed files and copy them to the requesting database.

In addition to the benefits previously listed, a shared log archive device improves the catch-up process by allowing the standby database to directly retrieve older log files from the archive in local catch-up state, instead of retrieving those files indirectly through the primary in remote catch-up state. However, avoid using a serial archive device, such as a tape drive, for HADR databases. With serial devices, you might experience performance degradation on both the primary and standby databases because of mixed read and write operations. The primary writes to the device when it archives log files and the standby reads from the device to replay logs. This performance impact can occur even if the device is not configured as shared.

10.4 How to recognize primary is not able to find log file needed by standby

In the event that the primary cannot locate a log file (for example, S0000100.LOG) needed by a standby database, the message in listing 14 can be found in the db2diag.log of the primary database.

Listing 14. Diagnostic output from the primary

```
2012-03-08-04.50.49.654736-300 I2952993E395          LEVEL: Warning
PID     : 7690                 TID : 46914795989312  PROC : db2sysc
INSTANCE: db2inst1             NODE : 000
HOSTNAME: beaverton-a
EDUID   : 457                  EDUNAME: db2lfr.0 (HADRDB)
FUNCTION: DB2 UDB, recovery manager, sqlplfrFMReadLog, probe:5120
```

```
MESSAGE : Return code for LFR opening file S0000100.LOG was -2146434659

2012-03-08-04.50.49.655067-300 I2953389E600          LEVEL: Error
PID     : 7690                 TID : 46912841443648  PROC : db2sysc
INSTANCE: db2inst1             NODE : 000
HOSTNAME: beaverton-a
EDUID   : 469                  EDUNAME: db2hadrp.0.1 (HADRDB)
FUNCTION: DB2 UDB, High Availability Disaster Recovery, hdrEdu::hdrEduP, probe:20591
MESSAGE : ZRC=0x860F000A=-2045837302=SQLO_FNEX "File not found."
          DIA8411C A file "" could not be found.
DATA #1 : <preformatted>
HADR primary database failed to read log pages for remote catchup. sqlplfrScanNext
scanPages = 0, scanFlagsOut = 0x2

2012-03-08-04.50.49.655699-300 E2953990E433          LEVEL: Event
PID     : 7690                 TID : 46912841443648  PROC : db2sysc
INSTANCE: db2inst1             NODE : 000
HOSTNAME: beaverton-a
EDUID   : 469                  EDUNAME: db2hadrp.0.1 (HADRDB)
FUNCTION: DB2 UDB, High Availability Disaster Recovery, hdrSetHdrState, probe:10000
CHANGE  : HADR state set to HDR_P_REM_CATCHUP_PENDING (was HDR_P_REM_CATCHUP), connId=151
```

Upon encountering this condition, the standby database cannot proceed and is deactivated. The log file (in this case S0000100.LOG and possibly subsequent log files) must be made available, either in the log archive or the overflow log path or the active log path of either the primary or the standby database. DB2 does not automatically delete user copied files in the overflow and active log path. Manually remove the files when they are no longer needed by any HADR standby or any other application.

10.5 Setting up shared log archive on Tivoli Storage Manager

Proxy node function provides capability so that several IBM Tivoli Storage Manager client nodes can perform data protection operations against a centralized name space on the Tivoli Storage Manager server, as opposed to node-specific namespace. The target client node owns the data and agent nodes act on behalf of the target nodes to manage the back-up data.

The proxy node target is the node name defined on the Tivoli Storage Manager server to which back-up versions of distributed data are associated. The data is managed in a single namespace on the Tivoli Storage Manager server as if it is entirely the data for this node. The proxy node target is also referred to as the multi-node as it denotes a node that owns the data of several proxy node agents.

The other production and back-up hosts are designated as the agent node for the proxy node on Tivoli Storage Manager Server, allowing each node to perform tasks on behalf of the target node.

The proxy node target name can be a real node (for example, one of the application hosts) or a virtual node name (that is, with no corresponding physical node). Use a virtual node as the target node.

For example, if HADR is used for a database on machines called beaverton-a and beaverton-b, you can create a virtual proxy node name such as beaverton-v on TSM server with the sample commands in listing 15.

Listing 15. Creating virtual proxy node names

```
Grant proxynode target=beaverton-v agent=beaverton-a
Grant proxynode target=beaverton-v agent=beaverton-b
```

And set the following database configuration parameters on the primary and standby database:

- `vendoropt` to "-asnode=beaverton-v"
- `logarchopt` to "-asnode=beaverton-v"

For multiple standbys, just grant proxynode access to all machines on the Tivoli Storage Manager server and configure the `vendoropt` and `logarchopt` database configuration parameters on all standbys.

11 Reads on standby

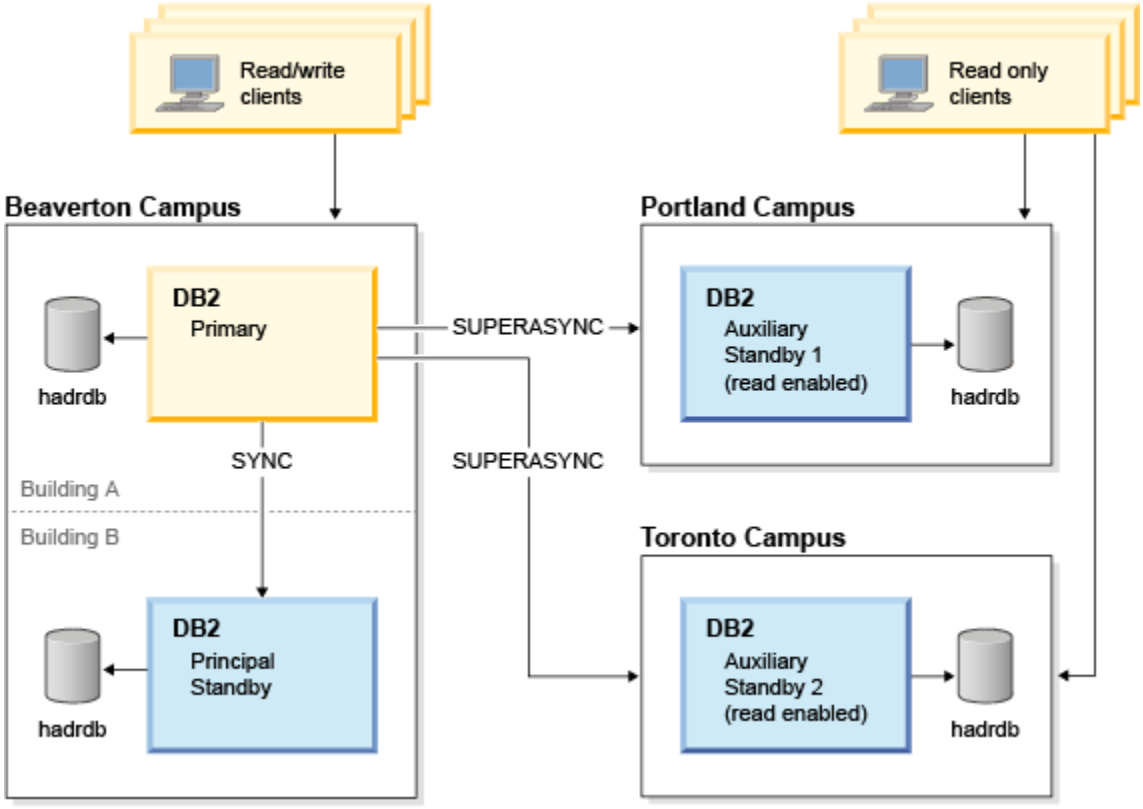
Use the reads on standby feature to execute read-only workloads on the standby database while it continues to provide protection against data loss by replicating data changes from the primary database. In a multiple standby environment, reads on standby is supported on all standbys. Read-only workloads running on standby database do not compromise data loss protection or the ability to takeover in an event of a failover. Since read-enabled standby supports only UR isolation, queries might see transaction changes even if the transaction did not commit.

Best practice

Enable read-only workloads on auxiliary standbys to ensure zero impact on the principal standby.

Multiple standbys offer greater value and flexibility in leveraging reads on standby in a HADR environment. Read-only workloads can be off-loaded to auxiliary standbys, to ensure zero impact to replication performance. In the event of a takeover, all user connections to the standby taking over as the new primary are terminated. Takeover on one standby does not impact read connections to other standbys, thus providing continuous data availability for both read/write and read-only workloads. The client configuration file db2dsdriver.cfg can be used to configure the target servers for the read/write and read-only clients. For more information, see the *Client Affinity and Db2dsdriver.cfg* section. Figure 14 shows a multiple standby setup, where the principal standby is dedicated for high availability and the two auxiliary standbys servicing read-only clients in addition to providing disaster recovery.

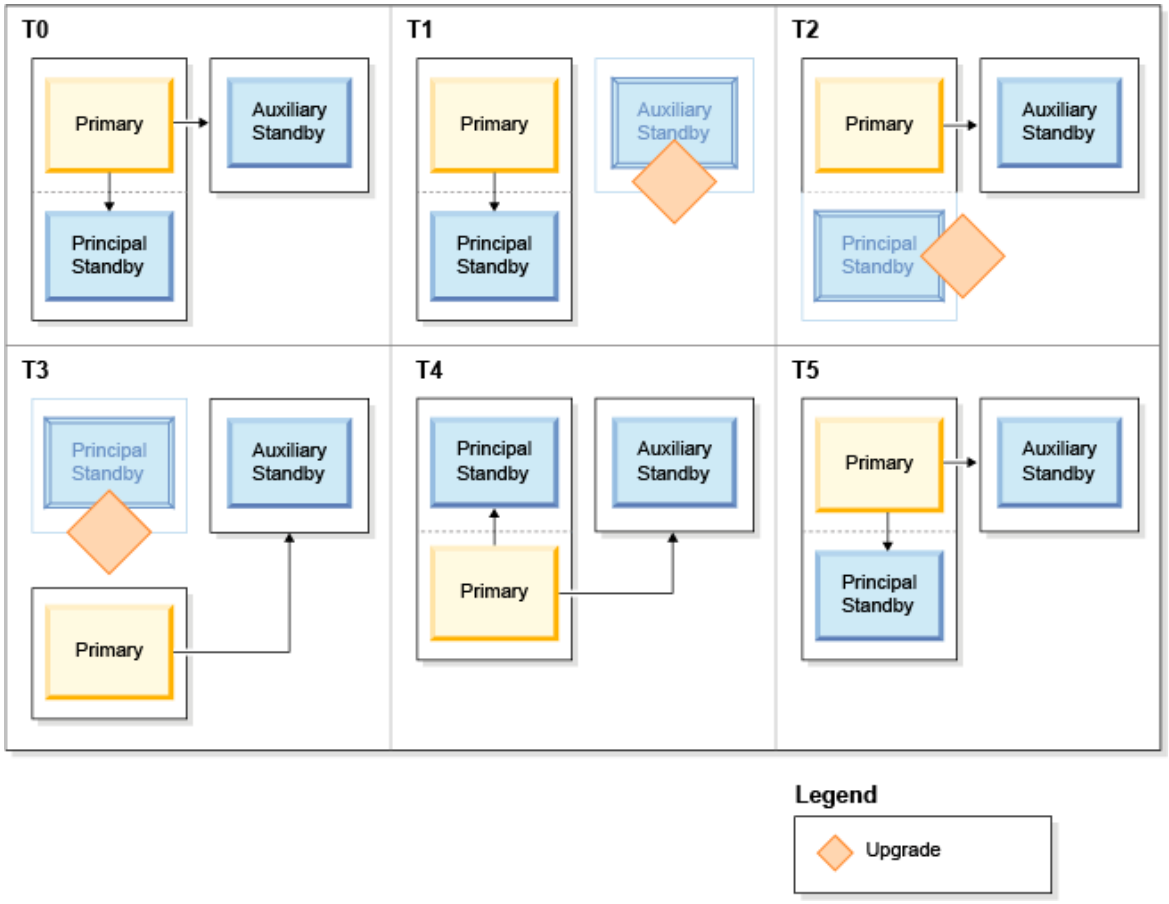
Figure 14. Multiple standby with read enabled standbys



12 Rolling upgrade

DB2 HADR provides high availability while updating software, hardware, database system, and database configurations without any impact to business applications. Business applications experience only a momentary service interruption when they are switched from one database server to the other. Having multiple standbys offers high availability and disaster recovery throughout the rolling upgrade process. The procedure for rolling upgrade is similar to single standby. Figure 15 shows the sequence of steps for upgrading the HADR servers.

Figure 15. Rolling upgrade support



For example, assume that you have a two standby HADR environment where A is primary, B is principal standby, and C is an auxiliary standby. The steps for rolling upgrade are:

T1. Upgrade the auxiliary standby C and then activate C.
T2. Upgrade principal standby B, then activate B.
T3. Initiate a graceful takeover on principal standby B. The database B is the new primary. The new standby A is disconnected because it is still on older release. Standby C is automatically reconfigured to connect to new primary B.
T4. Upgrade A, then activate A. database A connects to the current primary B as a standby. All databases are now upgraded.
T5. Optionally issue a graceful takeover on standby database A to restore A role as primary. Standby C is automatically redirected to new primary A.

See "Performing rolling updates and upgrades in a DB2 High Availability Disaster Recovery (HADR) environment" in the DB2 LUW Information Center for more detail.

13 Log spooling

A buffer is used on the standby database to receive log data sent by the primary. The log data is kept in the buffer to facilitate log replay. If log replay on standby is slow, this buffer can become full. Unless SUPERASYNC is used, new transactions on the

primary can be blocked because it is not able to send log data to the standby if there is no room in the buffer to receive the data.

Log spooling is a new feature in DB2 Version 10.1. Use this feature to have transactions on the primary to make progress without waiting for the log replay on the standby. Log data that is sent by the primary is written, or spooled, to disk on the standby if it falls behind in log replay. The standby can later on read the log data from disk. This configuration allows the system to better tolerate either a spike in transaction volume on the primary, or a slowdown of log replay (due to the replay of particular type of log records) on the standby. This feature can be used in both single and multiple standby environments.

When making use of log spooling, ensure that adequate disk space is provided to the active log path of the standby database to hold the spooled log data, in addition to the disk space required for active logs (which is determined by the `logprimary`, `logsecond`, and `logfilsiz` configuration parameters).

You enable log spooling by setting the `hadr_spool_limit` database configuration parameter. It specifies an upper limit on how much data is written, or spooled, to disk if the log receive buffer fills up. The default value of 0 means no spooling. The special value of -1 means unlimited spooling. Data can be spooled up to the amount of disk space available in the active log path.

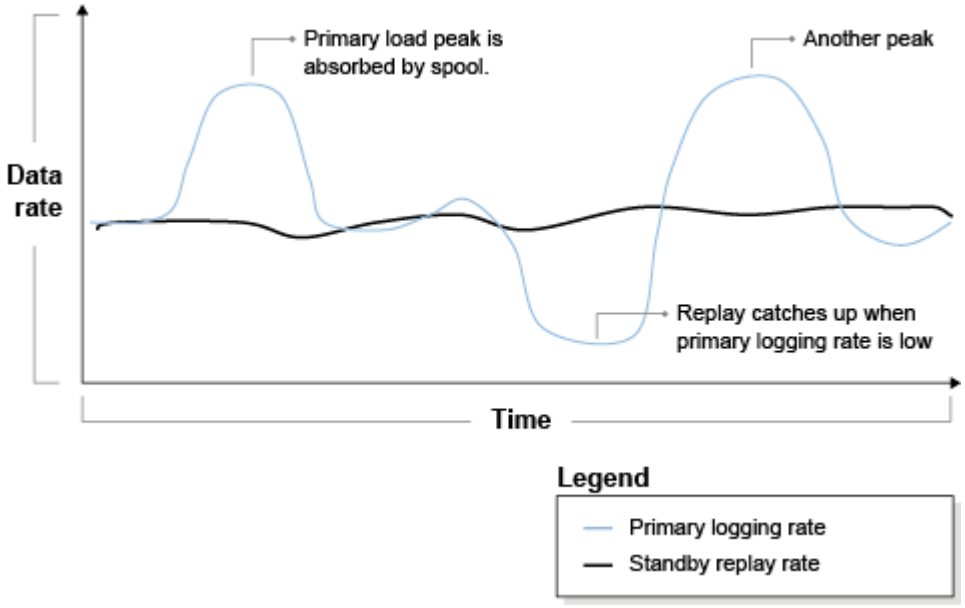
When configuring log spooling, in addition to the extra disk space requirement on standby, you should also consideration that there could be a large gap between the received and replayed log positions on the standby. This large gap can lead to a longer takeover (forced and graceful) time because the standby *must finish replaying all spooled logs* (logs in the gap) before it changes into primary role.

Using log spooling does not compromise the data protection provided by the HADR feature. Data from the primary is still replicated in the form of logs to the standby using the specified synchronization mode; it just takes time to apply (through log replay) the data to the table spaces.

13.1 Why enable log spooling?

Log spooling allows larger variations in primary logging rate. In SYNC, NEARSYNC, and ASYNC modes, it can absorb large load spikes in primary workload. SUPERASYNC mode allows large spikes even without spooling since it uses the primary as a spool. Figure 16 shows an example. During the spike, primary logging rate is much higher than standby replay rate. Standby receives the logs and spools the extra logs for later replay. A larger standby received/replay log gap is expected during the spike. The spooled logs are replayed when primary logging rate slows down. Without spooling, received logs are buffered only in memory and received/replay log gap is limited to the receive buffer size (controlled by registry variable DB2_HADR_BUF_SIZE), which is generally much smaller than spooling device size.

Figure 16. Effect of log spooling on logging and replay rate



13.2 Estimating amount of time to replay through the spool

The average log replay speed on standby can be estimated by doing the following things:

- Shutting down standby for a while.
- Running workload while standby is down.
- Activating the standby database and immediately collect the value of primary log position (value **A**) as well as standby log replay position (value **B**)
- Measuring the time it takes for standby replay position to reach value **A**
- Determining the average log replay speed. Average log replay speed = (**A** – **B**) / (time for standby replay to reach **A**).

After you know the replay speed and maximum toleration for takeover time, you can use the following calculation to determine the value to use for `hadr_spool_limit` configuration parameter:

replay speed (in bytes/sec) * *max toleration for longest takeover time* (in secs) / 4096 (bytes/page)

If a takeover is required, use the **db2pd -hadr** command just before the **TAKEOVER HADR** command to determine how much log data is currently spooled. This determination helps you predict of how much time the takeover will take. During the takeover execution, use the **db2pd -hadr** command to monitor its progress in replaying the spool.

13.3 SUPERASYNC versus log spooling

Using the SUPERASYNC synchronization mode can also avoid the pressure on the primary that can be caused by slow standby log replay. The advantages of using SUPERAYNC are as follows:

- SUPERASYNC can avoid back-pressure on the primary caused by slow or unreliable network.
- SUPERASYNC can be viewed as unlimited spooling. There is no back-pressure regardless of availability of disk space on standby, or other limit imposed by user by using the `hadr_spool_limit` configuration parameter.

The disadvantages of SUPERASYNC are as follows:

- SUPERASYNC provides weaker data protection than other synchronization modes, while log spooling can be used with SYNC, NEARSYNC, and ASYNC modes to provide stronger protection.
- A SUPERASYNC standby can fall far behind the primary because there is no limit at all. This situation can result in very long graceful takeover time.

These two features are not mutually exclusive. In fact, enabling log spooling on standby using SUPERASYNC might provide the best combination of minimal impact to primary performance as well as minimize the data lost in the event of primary outage.

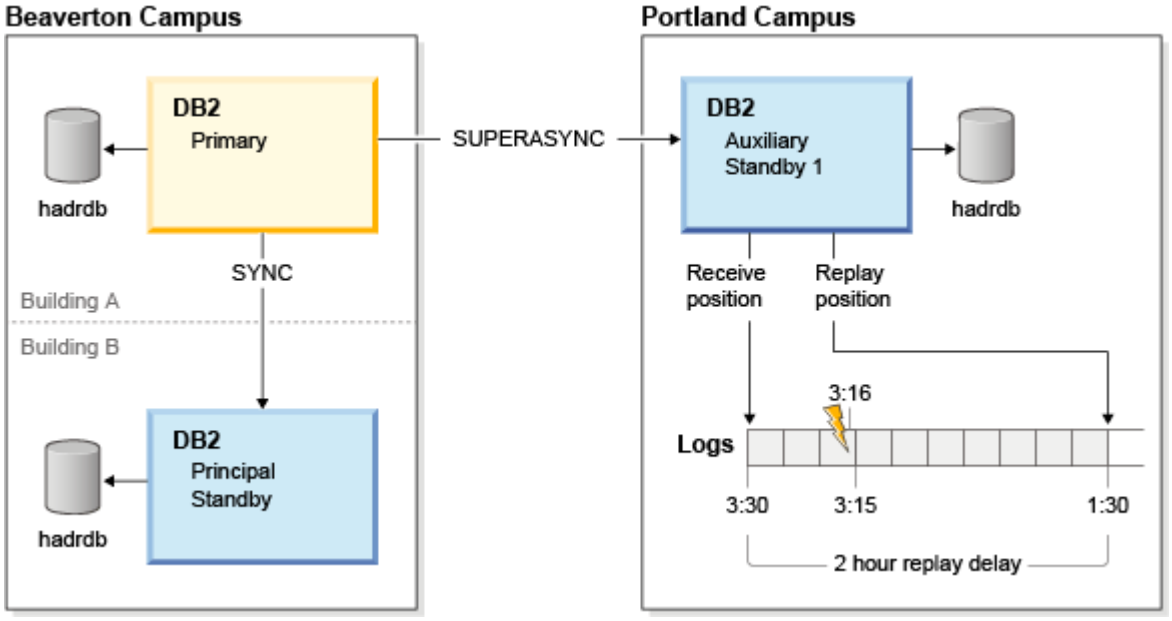
14 Delayed log replay on standby

Normally, in HADR, logs on primary are shipped to the standbys as soon as possible, and logs are replayed on the standbys as soon as possible. An errant transaction on the primary can be shipped and replayed on the standbys within seconds. An example of an errant transaction is "DELETE FROM current_sales", where valuable data is accidentally deleted. Recovering from an errant transaction can be costly. You might have to restore an earlier backup image and roll forward to a point in time right before the errant transaction and experience a large outage.

The delayed log replay feature of HADR was added in DB2 Version 10.1 and was designed to make recovering from errant transactions easy. When delayed log replay is configured, the standby does not commit a transaction until the configured replay delay time has passed since the transaction commit time on the primary. For example, if a transaction is committed on the primary at 5:00, and delayed replay is configured to be 1 hour, the transaction is guaranteed not to be committed on the standby before 6:00. In other words, standby replay is intentionally kept behind the primary. The standby still receives as much log data as possible when delayed replay is enabled to prevent back pressure on the primary. Log receive is blocked only when the log receive buffer and log spool are filled.

Figure 17 shows a hypothetical multiple standby environment with one standby configured on host beaverton-b for high availability. This is the principal standby and it runs in SYNC mode with no delayed replay. An additional standby is configured at the Portland campus. This standby has delayed replay configured to maintain a window of recoverability from errant transactions on the primary, and has also enabled log spooling to accommodate the additional log data that accumulates in the replay delay period.

Figure 17. HADR delayed replay



As shown in figure 17, a two hour delayed replay window is defined for the auxiliary standby at the Portland campus. The standby is currently receiving logs generated at 3:30. An errant transaction was issued on the primary at time 3:16 (the lightning bolt) which has been shipped to the standby, but not replayed. Logs generated at time 1:25 are currently being replayed, and the standby can currently replay only logs older than *current_time* minus the `hadr_replay_delay` value, which is 3:30 - 2 hours (all logs older than 1:30). The logs with commit times in the time delay window (red zone), including the errant transaction, are currently protected by delayed replay. HADR can be stopped on this system and the database can be manually rolled forward to point in time 3:15 to avoid replaying the errant transaction.

The PRIMARY_LOG_TIME, STANDBY_LOG_TIME, and STANDBY_REPLAY_LOG_TIME fields in **db2pd -hadr** command and MON_GET_HADR table function report transaction time. They can be used to monitor delayed replay operation.

Only transaction commit is held back on the standby. The data changes of a transaction are not held back. This is OK because the transaction can be rolled back during the errant transaction recovery process. Because reads on standby sees uncommitted transactions, data returned from reads on standby might not reflect the delay effect. It may see the errant transaction before it is rolled back if the recovery process was invoked to avoid the errant transaction.

Configuring delayed log replay

To configure a standby system for delayed log replay you must set the `hadr_replay_delay` database configuration parameter and restart HADR for the value to take effect. The `hadr_replay_delay` configuration parameter can be set only on a standby that is in SUPERASYNC synchronization mode. You cannot set the parameter on a primary database. The parameter is specified in units of seconds, so for the two hour replay delay shown in figure 17, `hadr_replay_delay` would be set to 7200 seconds, as shown in listing 16.

Listing 16. Enabling delayed replay

```
UPDATE DB CFG FOR DB hadrdb USING hadr_replay_delay 7200
DEACTIVATE DB hadrdb
ACTIVATE DB hadrdb
```

You can disable delayed log replay by setting the delayed replay time to 0, as shown in listing 17.

Listing 17. Disabling delayed replay

```
UPDATE DB CFG FOR DB hadrdb USING hadr_replay_delay 0
DEACTIVATE DB hadrdb
ACTIVATE DB hadrdb
```

Clock Synchronization

Delayed replay works by comparing timestamps in the log records (shipped from the primary, so based on primary clock) and the current system time of the standby. As a result, it is very important to synchronize the clocks of the primary and standby databases. Transaction commit is replayed on the standby according to the following equation:

(*current time on the standby* - `hadr_replay_delay` *time*) >= *timestamp of the committed log record*)

14.1 Log spooling consideration for delayed replay

If you enable delayed replay, also enable log spooling by setting the `hadr_spool_limit` database configuration parameter. Because of the intentional delay of replay, the replay position can be far behind log receive position on the standby. Without log spooling, log receive is bottlenecked by the size of the receive buffer. With spooling enabled the standby can receive many more logs beyond the replay position, providing greater protection against data loss in case of primary failure. Because of the mandatory SUPERASYNC mode, the primary is not blocked by enabling delayed replay.

14.2 Takeover and delayed log replay

Takeover is not allowed on a standby with delayed replay enabled. In order to initiate a takeover, you must first disable delayed log replay as described in the configuration section above. If you attempt to issue a takeover before disabling delayed log replay, you will get a SQL1770N error code with reason code 11.

In addition, a takeover operation must replay through all received logs. Since delayed log replay, by nature, accumulates logs, there can be a large amount of log data to replay during the takeover operation. The time to replay these logs increases with the delay amount, log spool size, and log generation rate on the primary. There is therefore a tradeoff between takeover performance and the added protection of delayed replay.

14.3 Errant transaction recovery process

In the earlier example, an errant transaction is received by the standby at time 3:16 but is not committed on the standby until 5:16. If you catch the error on the primary before 5:16, you can recover using the following process:

- You notice errant transaction on the primary.
- You check that the errant transaction has made it to the standby, but has not yet been replayed. This can be done by looking at the

STANDBY_REPLAY_LOG_TIME and making sure it has not reached the errant transaction commit time, and looking at the standby STANDBY_LOG_TIME (logs received) to ensure that the errant transaction has been received. The times are reported by table function and db2pd (see the earlier monitoring section). If the standby has not received enough log files, you can wait for more logs to be shipped over, but run the risk of replay time reaching errant transaction time. For example, if the delay is 1 hour, you should stop HADR no later than 50 minutes after the errant transaction (10 minutes safety margin), even if log shipping has not reached the desired point in time (PIT). You can stop HADR early and manually copy additional primary log files over if they have not made it to the standby before the errant transaction commit time falls outside the replay window. In a typical scenario, you have the following times listed from most recent to earliest:
    - PRIMARY_LOG_TIME
    - STANDBY_LOG_TIME (logs received)
    - Errant transaction commit time
    - Desired rollforward point in time
    - STANDBY_REPLAY_LOG_TIME
- When you are certain that logs for the desired rollforward point in time have been received by the standby, but not replayed, deactivate the standby and issue the **STOP HADR** command. The database is then in standard role.
- Roll forward the standby to the desired point in time and stop. The database is then a normal database that can be activated and used. It is in the exact same state that the primary database was in at time PIT (prior to the errant transaction).
- At this point there are 2 options for re-establishing HADR:
    - You can use this database as the new primary database. Direct clients to this database and reinitialize the old primary (and other standbys) as a standby by restoring from a backup image taken on this new primary.
    - Copy the affected data from this database back to primary (for example, by exporting and importing the table current_sales in the example). After the data is manually restored on the primary, reinitializ e this standby by restoring from a backup image taken on the primary. This standby must be reinitialized because it is no longer in rollforward mode and might have generated its own log records (undo records) during rollforward stop. It has diverged from the primary log stream. If other standbys exist, the missing data is reapplied to them when it is updated on the primary through the import, so no re-initialization is required on the other standbys.

15 NAT - Network Address Translation

Normally, HADR does a cross-check of local and remote addresses on primary and standby at HADR connection time. The following check is done (the check compares IP address rather than the original string in the `hadr_local_host` parameter and the `hadr_remote_host` parameter):

HADR local address for primary = HADR remote address for standby
and
HADR local address from standby = HADR remote address for primary

When registry variable DB2_HADR_NO_IP_CHECK is set, this cross-check is skipped. This configuration is usually used only in NAT (network address translation) scenarios

where a host might be seen with different IP addresses locally and remotely. For example, host A and B might be known as A and B locally, but to connect to each other, they have to use the address A' and B'.

Table 6. NAT scenarios for two databases

	hadr_local_host	hadr_remote_host	hadr_target_list
Database on A (primary)	A	B’	B
Database on B (standby)	B	A’	A

In NAT scenario, DB2_HADR_NO_IP_CHECK should be set on both primary and standby instances. In multiple-standby set up, it should be set on all databases that might be making a connection to another database across a NAT boundary. If a database never crosses a NAT boundary to connect to another database (no such link is configured), then it does not need to have this parameter set. Setting this parameter bypasses certain checks, making DB2 configuration checking weaker.

The `hadr_target_list` parameter value in NAT

In a NAT multiple standby setup, each standby `hadr_local_host` and `hadr_local_svc` parameter values must be listed in the `hadr_target_list` parameter on the primary. In the above example, A lists B in the `hadr_target_list` parameter, but use B' for its `hadr_remote_host` parameter value. When a standby connects, the primary finds the `hadr_local_host` and `hadr_local_svc` values of the standby in its `hadr_target_list` value and accept the connection.

Normally, on start up in multiple standby mode, a standby checks that its `hadr_remote_host` and `hadr_remote_svc` values are in its `hadr_target_list` value, to ensure that on role switch, the old primary can become a new standby. In NAT scenarios, the check fails (A' is different from A). Thus the registry variable DB2_HADR_NO_IP_CHECK must be set to bypass this check. At connection time, a standby checks that the values of `hadr_local_host` and `hadr_local_svc` from the primary are in its `hadr_target_list`. The check ensures role switch can still succeed on this pair.

HADR Automatic Reconfiguration in NAT

Automatic reconfiguration is disabled when NAT is enabled. A primary does not automatically set its `hadr_remote_host` and `hadr_remote_svc` values to its principle standby, and a standby cannot be automatically redirected to the primary.

16 Monitoring

In DB2 Version 10.1, HADR monitoring has been enhanced to support multiple standby databases, as well as provide better performance monitoring for single and multiple standby systems.

A new table function, MON_GET_HADR has been added. The **db2pd -hadr** command content and format has been updated. The two interfaces return identical information. The table function uses standard SQL interface, so it is very flexible. It can be queried from any SQL client and the output can be further processed by SQL filter, join, aggregate, and so on. In comparison, the **db2pd** command can be issued only on the database host machine. But the **db2pd** command is lightweight, works on databases with reads on standby disabled, and can be incorporated into administration scripts. All old interfaces such as the CLP database snapshot command, SNAPHADR view, SNAP_GET_HADR table function, and snapshot monitor element API have been deprecated for HADR monitoring.

16.1 Monitoring Multiple Standbys

When the MON_GET_HADR table function or the **db2pd -hadr** command is issued to the primary database, information about all standbys is returned. Deprecated interfaces, like snapshot, report only the principal standby.

The MON_GET_HADR table function returns rows, each representing a primary-standby log shipping channel. For the **db2pd** command, repeating sections are used to represent multiple channels.

16.2 Reporting Remote Information

As part of DB2 Version 10.1 enhancement, the primary and standby exchange monitoring information on each heartbeat message. Thus information about remote databases is also available locally, with a small delay of no more than the heartbeat interval. The heartbeat interval is capped at 30 seconds and the heartbeat interval is also reported in the monitoring.

Best practices

Unless reads on standby is enabled, the table function cannot be issued to a standby database. Monitor from the primary, or using the **db2pd** command.

During takeover, use the **db2pd** command over table functions for the following reasons:

- Connections are forced off the old primary.
- Connections are forced off the old standby, if reads on standby is enabled.
- The old primary might turn into a standby that does not support reads on standby.
- There might be a short time when both databases are in standby roles.

On a standby, monitoring returns only information about the standby and the primary. No information about other standbys is reported. Use the primary as the central location for HADR monitoring.

Certain monitoring fields are applicable to either the primary or standby only. For example, PEER_WAIT_LIMIT is applicable only to primary, STANDBY_RECV_BUF_SIZE, READS_ON_STANDBY_ENABLED are applicable only to standby. When this kind of role-specific information is reported, it is on data from the database currently in this role (which may be the remote database), rather than the local database. For example, PEER_WAIT_LIMIT seen on a standby database is the value configured on the primary database, not the standby local configuration (which is used only when the standby turns into a primary).

16.3 Unit of Time

According to monitor table function convention, all MON_GET_HADR time duration fields use milliseconds as their unit. For those fields reflecting a configuration parameter (such as `hadr_timeout` or `hadr_peer_window`) whose unit of configuration is seconds, the number returned by MON_GET_HADR table function is different from

the number used in GET DB CFG or UPDATE DB CFG command, or the number returned by SYSIBMADM.DBCFG admin view, or SYSPROC.DB_GET_CFG() table function. For example, for a `hadr_timeout` value configured to 60 seconds, MON_GET_HADR returns 60000, while the configuration-oriented interfaces return 60.

16.4 Sample MON_GET_HADR output

Example 1

The query in listing 18 is issued on a primary database with three standbys. Three rows are returned. The HADR_ROLE column represents the role of the database to which the query is issued. Therefore it is PRIMARY on all rows.

Listing 18. MON_GET_HADR query and output (primary)

```
db2 "select HADR_ROLE, STANDBY_ID, HADR_STATE, varchar(PRIMARY_MEMBER_HOST,20) as
PRIMARY_MEMBER_HOST, varchar(STANDBY_MEMBER_HOST,20) as STANDBY_MEMBER_HOST from table
(mon_get_hadr(NULL))"

HADR_ROLE    STANDBY_ID    HADR_STATE       PRIMARY_MEMBER_HOST   STANDBY_MEMBER_HOST
-----------  ----------    ---------------  --------------------  --------------------
PRIMARY               1    PEER             beaverton-a            beaverton-b
PRIMARY               2    REMOTE_CATCHUP   beaverton-a            portland
PRIMARY               3    REMOTE_CATCHUP   beaverton-a            toronto

3 record(s) selected.
```

Example 2

The query in listing 19 is issued to a standby database (with reads on standby enabled). The standby knows only about its own primary. Only one row is returned even if the standby is part of a multiple standby system. STANDBY_ID is always zero when a query is issued to a standby.

Listing 19. MON_GET_HADR query and output (standby)

```
db2 "select HADR_ROLE, STANDBY_ID, HADR_STATE, varchar(PRIMARY_MEMBER_HOST,20) as
PRIMARY_MEMBER_HOST, varchar(STANDBY_MEMBER_HOST,20) as STANDBY_MEMBER_HOST from table
(mon_get_hadr(NULL))"

HADR_ROLE    STANDBY_ID    HADR_STATE       PRIMARY_MEMBER_HOST   STANDBY_MEMBER_HOST
-----------  ----------    ---------------  --------------------  --------------------
STANDBY               0    PEER             beaverton-a           beaverton-b

1 record(s) selected.
```

16.5 The db2pd command

The **db2pd -hadr** command returns the same information as the MON_GET_HADR table function. For ease of use, fields not relevant to current status might be omitted in db2pd output. For example, standby-replay-only window fields are shown only when reads on standby is enabled.

Timestamps are printed in the local time zone. The integer in parentheses is the internal representation, typically the number of seconds since Jan 1,1970. This number is included for easy time arithmetic. Subsecond information is shown only in the formatted timestamp.

The output format of the **db2pd -hadr** command has changed in DB2 Version 10.1. It is now using "one field per line" format (similar to db2 CLP snapshot output) for easy reading and parsing. Listing 20 provides sample output from a 2-standby system.

Listing 20. db2pd query and output

```
  Database Member 0 -- Database HADRDB -- Active -- Up 0 days 00:23:17 -- Date 06/08/2011
13:57:23

                            HADR_ROLE = PRIMARY
                          REPLAY_TYPE = PHYSICAL
                        HADR_SYNCMODE = SYNC
                           STANDBY_ID = 1
                        LOG_STREAM_ID = 0
                           HADR_STATE = PEER
                  PRIMARY_MEMBER_HOST = beaverton-a
                     PRIMARY_INSTANCE = dbinst1
                       PRIMARY_MEMBER = 0
                  STANDBY_MEMBER_HOST = beaverton-b
                     STANDBY_INSTANCE = dbinst2
                       STANDBY_MEMBER = 0
                  HADR_CONNECT_STATUS = CONNECTED
             HADR_CONNECT_STATUS_TIME = 06/08/2011 13:38:10.199479 (1307565490)
         HEARTBEAT_INTERVAL(seconds) = 25
               HADR_TIMEOUT(seconds) = 100
       TIME_SINCE_LAST_RECV(seconds) = 3
            PEER_WAIT_LIMIT(seconds) = 0
          LOG_HADR_WAIT_CUR(seconds) = 0.000
    LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
  LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
                  LOG_HADR_WAIT_COUNT = 82
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 50772
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87616
           PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
           STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
                 HADR_LOG_GAP(bytes) = 0
     STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
       STANDBY_RECV_REPLAY_GAP(bytes) = 0
                   PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
                   STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
            STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
        STANDBY_RECV_BUF_SIZE(pages) = 16
            STANDBY_RECV_BUF_PERCENT = 0
          STANDBY_SPOOL_LIMIT(pages) = 0
                PEER_WINDOW(seconds) = 0
           READS_ON_STANDBY_ENABLED = Y
   STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N

                            HADR_ROLE = PRIMARY
                          REPLAY_TYPE = PHYSICAL
                        HADR_SYNCMODE = SUPERASYNC
                           STANDBY_ID = 2
                        LOG_STREAM_ID = 0
                           HADR_STATE = REMOTE_CATCHUP
                  PRIMARY_MEMBER_HOST = beaverton-a
                     PRIMARY_INSTANCE = dbinst1
                       PRIMARY_MEMBER = 0
                  STANDBY_MEMBER_HOST = portland
                     STANDBY_INSTANCE = dbinst3
                       STANDBY_MEMBER = 0
                  HADR_CONNECT_STATUS = CONNECTED
             HADR_CONNECT_STATUS_TIME = 06/08/2011 13:35:51.724447 (1307565351)
         HEARTBEAT_INTERVAL(seconds) = 25
               HADR_TIMEOUT(seconds) = 100
       TIME_SINCE_LAST_RECV(seconds) = 16
            PEER_WAIT_LIMIT(seconds) = 0
          LOG_HADR_WAIT_CUR(seconds) = 0.000
    LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.006298
  LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.516
                  LOG_HADR_WAIT_COUNT = 82
```

```
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 0, 16384
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 0, 87380
            PRIMARY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
            STANDBY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
                  HADR_LOG_GAP(bytes) = 0
     STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0000009.LOG, 1, 49262315
       STANDBY_RECV_REPLAY_GAP(bytes) = 0
                     PRIMARY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
                     STANDBY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
               STANDBY_REPLAY_LOG_TIME = 06/08/2011 13:49:19.000000 (1307566159)
          STANDBY_RECV_BUF_SIZE(pages) = 16
              STANDBY_RECV_BUF_PERCENT = 0
            STANDBY_SPOOL_LIMIT(pages) = 0
                  PEER_WINDOW(seconds) = 0
             READS_ON_STANDBY_ENABLED = N
```

16.6 Common monitoring questions and answers

For complete description of all fields, refer to the following topics in the DB2 Information Center:
"db2pd command"
"MON_GET_HADR table function"

Is the standby and its replay keeping up with the primary?

Check the primary, standby receive, and standby replay log positions. Both byte offset and transaction time are reported. The average gap between the primary position and the standby receive position and the average gap between the standby receive and the replay positions are also reported. The relevant fields are:

- Primary log position and time: PRIMARY_LOG_FILE, PRIMARY_LOG_PAGE, PRIMARY_LOG_POS, PRIMARY_LOG_TIME
- Standby log receive position and time: STANDBY_LOG_FILE, STANDBY_LOG_PAGE, STANDBY_LOG_POS, STANDBY_LOG_TIME
- Standby log replay position and time: STANDBY_REPLAY_LOG_FILE, STANDBY_REPLAY_LOG_PAGE, STANDBY_REPLAY_LOG_POS, STANDBY_REPLAY_LOG_TIME
- Primary position / standby receive position gap: HADR_LOG_GAP
- Standby receive position / standby replay position gap: STANDBY_RECV_REPLAY_GAP

Is primary logging blocked by HADR?

Check the LOG_HADR_WAIT_CUR field. This field shows how long the logger has been waiting on an HADR log shipping request. Zero or a small number (typically no more than a few milliseconds) is good. If the logger is blocked, this field grows in real time. For example, it shows 2 seconds in one query, and 12 seconds in a query issued 10 seconds later.

What is the impact of HADR on database logging?

Check LOG_HADR_WAIT_ACCUMULATED and LOG_HADR_WAIT_COUNT. While LOG_HADR_WAIT_CUR shows the current wait, these two fields show the accumulated wait. Using the accumulated time and the count together, you can compute average wait per log write for any given interval. For your convenience, the db2pd command also reports the recent average wait (averaged over the last a few seconds) as LOG_HADR_WAIT_RECENT_AVG. The recent average is not reported by the table function because the table function is intended as an API to provide raw data for tools to compute their own average in arbitrary interval.

The HADR wait time is more meaningful when compared to disk write time. Disk write time and count is reported by using the LOG_WRITE_TIME and NUM_LOG_WRITE_IO field from table function MON_GET_TRANSACTION_LOG. For example, the data might show that each write takes 4 milliseconds for disk write and 1.5 milliseconds for HADR wait, for a total of 5.5 milliseconds.

For SUPERASYNC synchronization mode, the logger does not enter peer state, so HADR has no direct impact on the logger. No HADR wait time is reported. There might be indirect impact such as adding more read load on the logging disk, therefore slowing down log write.

Why is the primary blocked or slow?

Check STANDBY_RECV_BUF_PERCENT. If it frequently hits 100%, it means that replay is slow and the receive buffer is full. That is, the replay cannot consume received data fast enough. When the receive buffer is full, the standby cannot receive more data and the primary is blocked. If it is usually low, then the network is likely to be the culprit. That is, the primary just cannot push out data fast enough.

When log spooling is enabled, the receive buffer should rarely become full because buffers are released even before they are replayed. But if the spool limit is reached or the spool device is full, then the receive buffer can still fill up and the primary is still blocked.

How much spool space is the standby using?

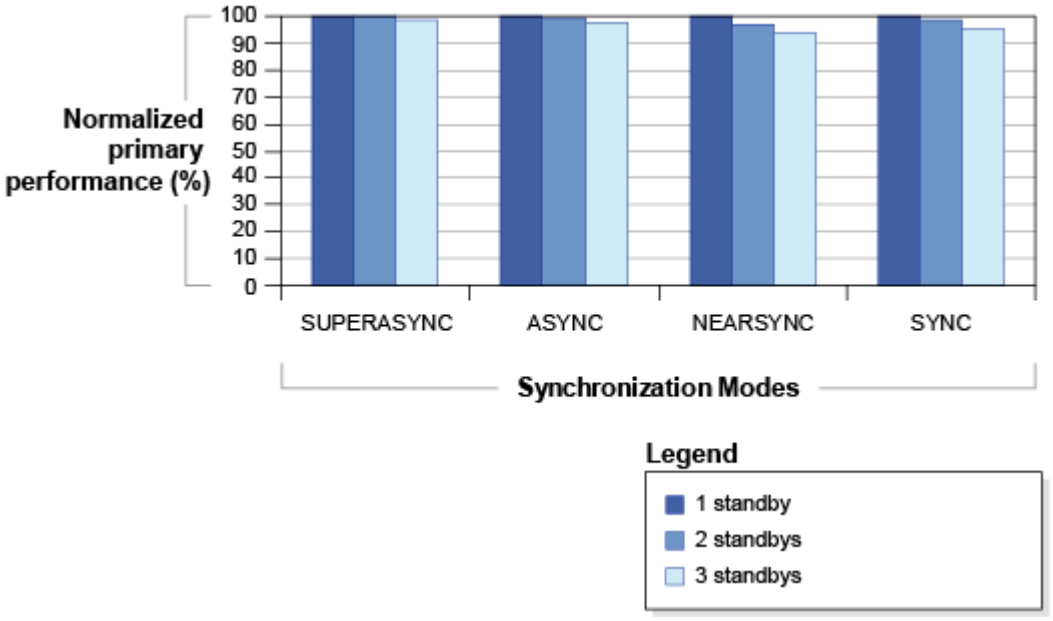
Use this formula to calculate how much spool space is being used:
(STANDBY_LOG_POS - STANDBY_REPLAY_LOG_POS) / 4096 - STANDBY_RECV_BUF_SIZE * STANDBY_RECV_BUF_PERCENT

The LOG_POS fields are in unit of bytes, while STANDBY_RECV_BUF_SIZE is in unit of pages (4096 bytes per page). The result shows the number of pages currently used by spooling. Compare it to STANDBY_SPOOL_LIMIT and you will know how close spooling is to the limit. Because the numbers are not read atomically, it is possible that the formula returns a negative number, which just means little or no spooling space is used.

17 Multiple standby performance

The addition of a second or third standby to a multiple standby HADR configuration should have negligible impact on performance on the primary and no impact on performance on the existing standby. To study the effect on performance, an OLTP workload was executed on the primary, with each synchronization mode, when two and three standbys were configured. The results were normalized to the performance achieved when one standby was configured with each synchronization mode. The performance impact on the primary (with each synchronization mode) after the additional of each standby was approximately 3% as shown in Figure 18. There was also no effect on the existing standby after configuring the additional standbys.

Figure 18. Effect on primary performance with multiple standbys

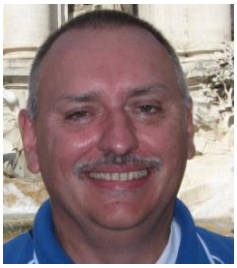


18 Conclusion

The HADR multiple standby feature, available from DB2 Version 10.1, allows you to set up to three standby targets, and offers both a high availability and disaster recovery solution for site failures. Whether you are constructing a brand new multiple standby configuration or converting from single to multiple standbys, proper planning of resources such as processors, memory, disks, and network is crucial for good performance. Adding or dropping additional standbys can be done dynamically as well as automatic client reroute. Reads on standby, delayed replay, and rolling updates are all supported. DB2 Version 10.1 also offers a new feature called log spooling which can be configured on any standby. DB2 Version 10.1 also provides new monitoring facilities that can help diagnose performance problems in single and multiple standby HADR environments. However, in a performance study it was observed that the performance impact on the primary after configuring each additional standby was approximately 3%.

19 Acknowledgements

Bruce Jackson has worked in enterprise software development with Informix Software and IBM since 1999. He has worked on the design and development of multiple high availability products in IBM, including clustered SAN file systems, DB2 pureScale, HACMP, and DB2 HADR. He has also worked on standards based open systems storage management solutions.



Dale McInnis is a Senior Technical Staff Member (STSM) at the IBM Toronto Canada lab. He has a B.Sc.(CS) from the University of New Brunswick and a Masters of Engineering from the University of Toronto. Dale joined IBM in 1988, and has been working on the DB2 development team since 1992. Dale's area of expertise includes DB2 for LUW Kernel development, where he led teams that designed the current backup and recovery architecture and other key high availability and disaster recovery technologies. Dale is a popular speaker at the International DB2 Users Groups (IDUG) conferences worldwide, as well as DB2 Regional users groups and IBM's Information On Demand (IOD) conference. His expertise in the area DB2 availability area is well known in the information technology industry. Dale currently fills the role of DB2 Availability Architect at the IBM Toronto Canada Lab.

Effi Ofer has played key roles in the development of DB2 in areas ranging from monitoring to recovery to indexing and concurrency. Most recently he led the multiple standby team.

Nailah Ogeer Bissoon is currently a performance technical team lead for the DB2 performance quality assurance team. Nailah joined the IBM Toronto Lab in 2004 and has worked on several DB2 performance benchmark publications and used this skill set to diagnose and fix performance related problems in the field. Ensuring that DB2 remains as one of the top performers among DBMSs is vital to her role.

Punit Shah has extensive experience in database architecture, workload optimization, performance and system virtualization. He has written extensively on variety of technology topics. He has been a member of DB2 recovery development team and recently contributed to the shared disk backup and recovery solution in DB2 Version 9.8 and the HADR development for future releases.

Roger Zheng is a master innovator and a world authority on the design and implementation of logging, locking, recovery and transaction management algorithms. Roger currently provides leadership to a number of DB2 Kernel teams, including HADR.

Vincent Kulandai Samy is a DB2 kernel developer in IBM Beaverton Lab, working on DB2 LUW kernel development for the past 10 years. He came to IBM as part of the Informix acquisition. Prior to the Informix acquisition, he was working on Informix IDS and XPS database kernel. His areas of expertise are database kernel, DB2 HADR, Multi-Temperature Warehouse, Linux kernel internals, and kernel debugging. For the past three years, Vincent has also been championing several DB2 HADR adoptions and new sale/deployments through on-site customer visits, consultancy, and customer advocacy.

Yuke Zhuge has been in the database industry for more than 15 years. He worked at Informix Software from 1995 to 2000. He has been working at IBM since 2000. He was one of the initial creators of DB2 HADR and has been the HADR component owner since its initial release. He is now leading the HADR team to add more exciting features.

We would like to thank Lindsay Hemms, Matthew Huras, Rob Causley, and Steven Pearson for their editorial contributions. We also thank Shayan Talaiyan for his help setting up the performance test environments.

20 References

Automated Cluster Controlled HADR (High Availability Disaster Recovery) Configuration Setup using the IBM DB2 High Availability Instance Configuration Utility (db2haicu):
http://download.boulder.ibm.com/ibmdl/pub/software/dw/data/dm-0908hadrdb2haicu/HADR_db2haicu.pdf

Configuring a clustered environment using DB2 High Availability Instance Configuration Utility (db2haicu):
http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.admin.ha.doc/doc/t0052800.html

HADR simulator:
http://www.ibm.com/developerworks/wikis/display/data/HADR_sim

21 For more information

To learn more about the value of renewing your IBM software subscription and support, contact your IBM sales representative or IBM Business Partner, or visit:
ibm.com/software/

To learn more about the IBM Support Portal, visit:
ibm.com/support/

LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

Without limiting the above disclaimers, IBM provides no representations or warranties regarding the accuracy, reliability or serviceability of any information or recommendations provided in this publication, or with respect to any results that may be obtained by the use of the information or observance of any recommendations provided herein. The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any recommendations or techniques herein is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Anyone attempting to adapt these techniques to their own environment do so at their own risk.

This document and the information contained herein may be used solely in connection with the IBM products discussed in this document.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.