IBM® DB2® for Linux®, UNIX®, and Windows®

Best Practices

DB2 High Availability Disaster Recovery

Dale McInnis
Senior Technical Staff Member
DB2 Availability Architect

Yuke Zhuge
DB2 Development

Jessica Rockwood
DB2 Development

Robert Causley
DB2 Information Development

Executive summary

DB2 High Availability Disaster Recovery (HADR) is an easy to use data replication feature that provides a high availability (HA) solution for both partial and complete site failures.

However, given the wide variety of users' requirements, there is no one ideal configuration for HADR. The decisions you make in setting up, tuning, and maintaining your HADR environment are often the result of weighing various pros and cons. For instance, you might need to balance the competing requirements of the availability of your database with protection from data loss. The good news is that this need to find a balance does not necessarily imply that one requirement needs to suffer.

This document provides a number of recommendations for setting up and maintaining your HADR environment in ways that help balance the protection HADR provides with performance and cost. The following specific areas of focus are covered:

- Setting up your system for fast failovers
- Tuning parameters to improve network performance
- Tuning parameters to minimize the impact of HADR-related logging on performance
- Choosing the right table reorganization method and load operation in an HADR environment

Introduction to HADR

HADR replicates data changes from a source database (called the primary) to a target database (called the standby). Each database uses its own storage, as this is a shared-nothing architecture. HADR provides fast failover in case the primary database fails. You can also switch the roles of primary and standby databases easily and quickly for scenarios such as rolling upgrade or rolling maintenance, keeping down time to minimum. HADR is versatile and is fully integrated into DB2 database systems requiring no special hardware or software and using a standard TCP interface to connect the primary and standby databases. Setup requires only a few database configuration parameters.

A central tenet of HADR is that the performance and availability of the database should not be impacted by challenges such as sudden spikes in the workload (which impacts the amount of log replay activity on the standby) or failed servers or networks (which can lead to a failover).

Tuning your HADR solution for optimal performance should follow some basic tenets up front to avoid potential issues over time. As well, an HADR setup has a number of implications for your database maintenance activities that you should know. This best practice paper addresses these points. It focuses on providing guidelines and specific recommendations for designing your HADR infrastructure and configuring your database in order to improve HADR-related performance, specifically logging and failover speed.

Setting up your system

Perform an infrastructure analysis

Before starting your HADR implementation, the critical first step is to analyze the system landscape that will be used. Each component in the infrastructure (such as server, storage, network, and software) needs to be reviewed to identify potential points of failure. Where possible, it is recommended to build in component redundancy (for example, two network adapters, backup server power). As part of the analysis activity, build a chart similar to the example HADR analysis chart in figure 1 that details each component failure on both primary and standby locations (if applicable) and the expected behavior in case of failure.

Location operating as primary	Failure location	Failure description	Expected result	Outage impact
Site A	Site A	For example, AIX® server failure	Tivoli® Systems Automation detects failure of site A within 5 seconds and invokes forced HADR takeover on site B	Any inflight transactions are rolled back; standby server takes over as primary and opens for transactions within 30 seconds

Figure 1. Example of an HADR analysis chart

After the implementation of the HADR setup is complete, use the HADR analysis chart to develop a test plan to validate the implementation and the timing expectations to ensure that high availability and data recovery business requirements can be met (or valid expectations can be set).

Requirements for setting up HADR

There are a number of requirements for setting up HADR, including:

- The operating system and the DB2 version and level must be the same on the primary and standby. The only exception to this is when you are performing rolling upgrades.

- The DB2 software for both the primary and the standby database must have the same bit size.
- A TCP/IP interface must be available between the HADR primary and the standby.
- The primary and standby databases must have the same database name.
- Table spaces must be identical on the primary and standby databases.

Beyond these requirements, there are a number of recommendations for setting up your HADR system.

Use dedicated, high performing disks or file system for the database logs

Because database changes are logged by a single log stream (HADR is not available in partitioned database environments or in DB2 pureScale environments), this stream can become system bottleneck. It is very important to have guaranteed I/O capacity on the logging file system. Do not share devices between the logging file system and table space file systems.

Make the location of archived logs accessible to both the primary and standby databases

Making the location of archived logs available to both databases has several advantages:

- The primary database is the only database that archives logs, but after a takeover, the new primary (original standby) starts archiving logs. Therefore, it is simplest to have all logs archived to the same location. If the archive device is not shared, after a few role switches, some files will be on one device and some on the other. Then you might need to copy the required log files manually before attempting a log retrieval for HADR catchup, when using one of the data replication products such as Q Replication, or other recovery actions.
- A shared archive allows the standby to retrieve log files directly from the archive device during local catchup state, relieving the primary from reading the files and sending them over the network to the standby.

Use a dedicated network for the HADR primary-standby connection

Other traffic (such as client-server communication, database backup, and log archive) on the network used for HADR communication can cause hiccups in HADR performance. The network is the most important part of an HADR landscape, as network connectivity is required to replicate database changes from the primary to the standby, keeping the two databases synchronized.

Whether dedicated or shared, the bandwidth for the HADR network must be greater than the database log generation rate (which can be calculated by monitoring the database and looking at the amount of logs generated in a time period and any other shared network activity.

Consider using multiple network adapters

Using multiple network adapters helps ensure that the failure of a single adapter does not result in the loss of network.

Consider using a virtual IP address for the database server

Configure clients to connect to the database using the virtual IP address. Redirect the virtual IP address from the primary database server to the standby database server upon failover. Because the IP address is shared between the primary and the standby databases, you can prevent both copies of your database from operating independently as primaries (sometimes called "split brain" or "dual primary") by having applications only talk to whichever database owns the virtual IP address.

Consider using automatic client reroute

Automatic client reroute redirects clients from the primary server to the standby server in the event of a failover. Configure your system and your applications to use either a virtual IP address or client reroute, but not both. One advantage of using client reroute instead of a virtual IP is that client reroute is built into the DB2 engine, so it does not require additional hardware and software. For more information on automatic client reroute, see the automatic client reroute roadmap - http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.ha.doc/doc/r0023392.html

Tuning parameters

Before you make any changes to the configuration parameters, try to have identical database manager (DBM) and database (DB) configuration parameters on the primary and the standby system. Changes to DBM and DB configuration parameters on the primary system are not automatically propagated to the standby system, so any updates need to be done on both systems manually.

Updates to non-dynamic database configuration parameters require special consideration on an HADR pair.

None of the HADR configuration parameters can be dynamically updated (configurable online parameters). In addition, the symmetry of the settings of these parameters is enforced between the two HADR databases. The HADR connection will fail if the database manager detects that the settings of the HADR configuration parameters (like HADR_SYNCMODE, HADR_TIMEOUT) are different on the primary and the standby databases. As a result, once the HADR parameters have been updated on both primary

and standby, both databases must be restarted (deactivated and reactivated) for the parameter update to take effect. If only one database is restarted, it cannot be activated due to the mismatch in HADR configuration. This type of update requires a primary outage.

Non-HADR configuration parameter symmetry is not enforced by HADR. As a result, database configuration parameters that are not configurable online parameters can be updated, without a primary outage, through a rolling update procedure. First, the standby can be updated and the database restarted. Then, an HADR takeover can be issued. The new primary uses the new setting of the specified database configuration parameter. The new standby can then be updated and the database restarted.

HADR-specific parameters

Choose the appropriate HADR synchronization mode

HADR provides four synchronization modes to suit a diverse range of operational environments and customer requirements. The synchronization mode is the most important HADR configuration parameter. The database configuration parameter **hadr_syncmode** can be set to one of the following values:

- SYNC: Transactions on the primary database commit only after relevant logs are written to disk on both the primary and the standby
- NEARSYNC: Transactions on the primary database commit only after the relevant logs are written to disk on the primary database and received into memory on the standby database
- ASYNC: Transactions on the primary database commit only after the relevant logs are written to local disk and sent to standby
- SUPERASYNC: Transactions on the primary database commit as soon as the relevant logs are written to local disk

Generally, the choice of HADR synchronization mode depends on network speed. For a WAN, ASYNC or SUPERASYNC modes are the recommended options because transmission latency has no effect on HADR performance in these modes.

For a LAN, the choice is most often between SYNC and NEARSYNC modes. An exception to this might be if the standby is being used for pure disaster recovery and the business requirements can withstand data loss in the case of a failover, or the workload is so heavy that even a LAN cannot handle the workload. Then, you might have to use ASYNC mode. In making the choice between SYNC and NEARSYNC, weigh the possible cost to system performance (resulting from communication overhead) in SYNC mode versus the security against “double failure” in NEARSYNC mode.

SYNC mode

SYNC mode offers the best protection of data. Two on-disk copies of data are required for transaction commit. The cost is the extra time for writing on the standby and sending an acknowledgment message back to the primary.

In SYNC mode, logs are sent to the standby only after they are written to the primary disk. Log write and replication events happen sequentially. The total time for a log write event is the sum of (primary_log_write + log_send + standby_log_write + ack_message). The communication overhead of replication in this mode is significantly higher than that of the other three modes.

NEARSYNC mode

NEARSYNC mode is nearly as good as SYNC, with significantly less communication overhead. In NEARSYNC mode, sending logs to the standby and writing logs to the primary disk are done in parallel, and the standby sends an acknowledgement message as soon as it receives the logs in memory. On a fast network, log replication causes no or little overhead to primary log writing.

In NEARSYNC mode, you will lose data if the primary fails and the standby fails before it has a chance to write the received logs to disk. This is a relatively rare "double failure" scenario. Thus NEARSYNC is a good choice for many applications, providing near synchronization protection at far less performance cost.

A realistic example of NEARSYNC performance analysis is as follows:

> Network throughput: 100Mb/second
>
> Network round trip time: 0.1 millisecond
>
> Assuming each log write on the primary writes 4 log pages (16 KB), log replication time for a write is:
>
> $16 * 1024 * 8 / 100*10^6 + 0.0001 = 0.00141072$ second

This translates to a rate of 11.6 MB/second, which is in the same order of disk write speeds. Because disk write on the primary and log replication to the standby are done in parallel, there is no or little log replication overhead. With the popularity of Giga bit networks today, the 100Mb/second network in the example is a relatively slow network. On a Giga bit network, the log replication rate would be 106MB/second, which is faster than many disks.

ASYNC mode

In ASYNC mode, sending logs to the standby and writing logs to the primary disk are done in parallel, just like in NEARSYNC mode. Because ASYNC mode does not wait for acknowledgment messages from the standby, the primary system throughput is min(log write rate, log send rate). ASYNC mode is well suited for WAN applications. Network transmission delay does not impact performance in this mode, but if the primary

database fails, there is a higher chance that logs in transit will be lost (not replicated to standby).

SUPERASYNC mode

This mode has the shortest transaction response time of all synchronization modes but has also the highest probability of transaction losses if the primary system fails. The primary system throughput is only affected by the time needed to write the transaction to the local disk. This mode is useful when you do not want transactions to be blocked or experience elongated response times due to network interruptions or congestion. SUPERASYNC mode is well suited for WAN applications. Since the transaction commit operations on the primary database are not affected by the relative slowness of the HADR network or the standby HADR server, the log gap between the primary database and the standby database might continue to increase. It is important to monitor the log gap in this mode as it is an indirect measure of the potential number of transactions that might be lost should a true disaster occur on the primary system.

Figure 2 shows when transactions are considered successful based on the synchronization mode chosen.

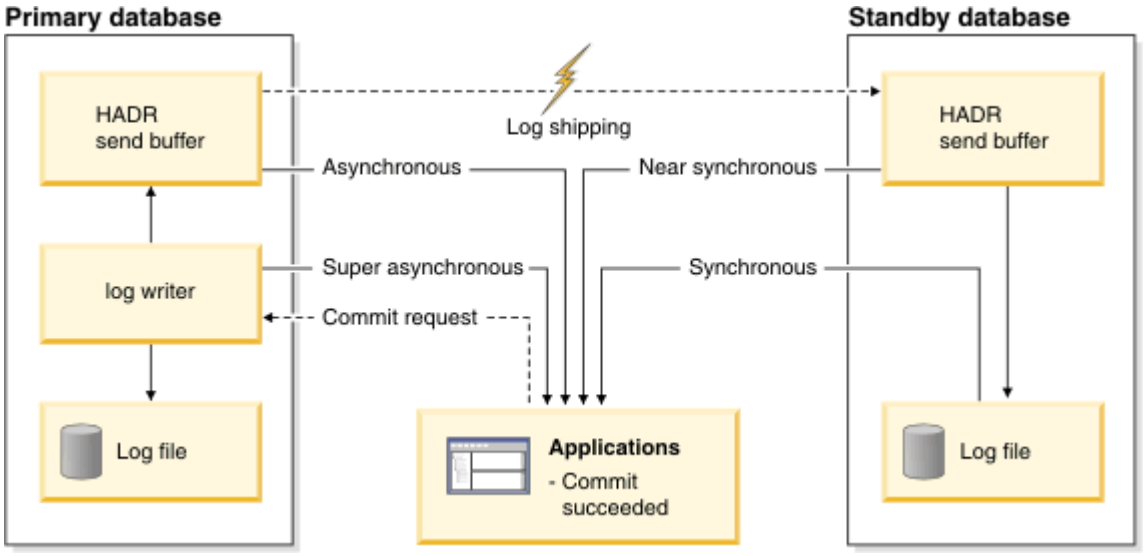

Figure 2. Synchronization modes for high availability and disaster recovery (HADR)

HADR simulator

You can also use the IBM DB2 High Availability Disaster Recovery (HADR) simulator to estimate HADR performance in different synchronization modes. The simulator is a very useful tool in HADR planning and tuning. You can run the simulator before deploying HADR to preview logging performance. If you have already deployed HADR, you can run the simulator to see what performance you would get if you changed the sync mode. All of this can be done within a few seconds, with no impact on the production system. For information on using the HADR simulator, see http://www.ibm.com/developerworks/wikis/display/data/HADR_sim.

Tune DB2_HADR_BUF_SIZE

Setting the **DB2_HADR_BUF_SIZE** registry variable to the size of expected workload spikes configures your HADR standby log receive buffer to absorb spikes in the primary workload.

Log receive on standby writes logs into this receive buffer as a producer. Log replay reads the logs as a consumer. If the consumer is too slow, the buffer fills up, and the standby cannot receive any more logs. For SYNC and NEARSYNC modes, chances are that the primary can still send out one log flush when the standby buffer is full. This flush will be buffered along the network pipeline from the primary to the standby, in various OS or network buffers. But the standby database is not able to receive it into its receive buffer. The primary will then be blocked, waiting for an acknowledgment message. For ASYNC mode, the primary will keep sending logs until the network pipeline fills up, then it will get network congestion error when trying to send more logs. For SYNC and NEARSYNC, if the network pipeline can not absorb a flush, the primary will also see congestion. Congestion is reported using the HADR monitoring mechanisms such as database snapshot or the db2pd command.

To measure workload spikes:

1. Temporarily deactivate the standby.

2. Look at the "primary log position" field in your HADR snapshot or db2pd output to measure primary log rate (Log position is the byte offset of current log position in log stream). If HADR is not enabled on the primary database, look at log write related fields from database snapshot (available via CLP command, snapshot view, or SNAP_GET_DB function) instead.

Monitoring HADR standby receive buffer usage

When tuning the standby buffer size, use the db2pd command with the -hadr option to monitor the HADR standby receive buffer usage. It will report the percentage of buffer used. 100% indicates a full buffer—that is, log receive is faster than log replay—and 0% indicates an empty buffer—that is, there is nothing to replay.

For tuning purposes, sample the buffer usage at a regular interval. If the percentage frequently hits 100%, consider using a larger buffer. If the percentage is always low, consider reducing the buffer size. If the percentage occasionally reaches high, you must balance the benefit of absorbing those spikes with the cost of the memory.

A large standby receive buffer will absorb spikes in primary workload. However, it will not help if the sustained replay rate on standby is slower than the log generation rate on the primary. It is recommended that the standby have the same hardware as the primary, so that the overall standby replay rate will keep up with primary log generation rate. If the standby cannot keep up with the primary, it will slow down the primary.

Tune hadr_timeout

The **hadr_timeout** configuration parameter determines the number of seconds the HADR

process waits before considering a communication attempt to have failed. If a database does not receive a heartbeat message from its partner database within the number of seconds set by the value of the **hadr_timeout** value, the database considers the connection down and closes the TCP connection. Tuning this parameter means balancing the ability of HADR to detect network failure promptly with the possibility of "false alarms." If the **hadr_timeout** configuration parameter is set too long, the HADR process cannot detect network or partner database failure promptly. The primary can end up waiting too long for a log replication attempt, blocking transactions on primary. If it is too short, the HADR process might get too many false alarms, breaking the connection too often. If a peer window is enabled (meaning the **hadr_peer_window** parameter is set to a non-zero value), each disconnection also leaves the primary in disconnected peer state for the configured peer window size duration, leaving transactions blocked on the primary.

An HADR database always disconnects when it detects network errors or remote end closures. **hadr_timeout** is only the last resort of connection failure detection. Usually, when a database crashes, its host machine cleans up its open connections and notifies the remote end of the closure. It is only a machine crash that might leave the remote end hanging until a timeout occurs. Various network layers might have their own heartbeat mechanism too, detecting a connection failure before HADR does. As soon as the failure is reported to the host machine, HADR detects it as HADR continuously monitors the health of the connection via polling.

The recommended **hadr_timeout** parameter setting is at least 60 seconds. When setting the **hadr_timeout** parameter, consider network reliability and machine response time. Use a longer timeout if the network has an irregular or long transmission delay. HADR processes send heartbeat messages at regular intervals. But send and receive are only as prompt as the processes themselves are scheduled to run on the host machines. Events like swapping can prevent a process from sending or receiving messages on time.

Tune hadr_peer_window

The **hadr_peer_window** database configuration parameter enables failover operations with no data loss if the primary failed within a specified time—the peer window. When the HADR primary database is in peer state, logs are sent directly from the in-memory log buffer when the logs are written to the local disk. If the connection is broken, the primary database blocks logging for **hadr_peer_window** seconds and no transaction is committed. If the primary database then fails within this window, we know that there are no committed-but-not-replicated transactions and therefore there is no data loss during the failover to the standby database.

When the peer window is set to non-zero value, the primary sends messages to the standby at regular interval in peer state, indicating a "peer window end" timestamp, which is the time until which the primary blocks logging should it loses connection to the standby. The current peer window end can be retrieved from the primary or standby using the usual HADR monitoring mechanisms such as database snapshot or the db2pd tool. When the primary database fails, you can check the peer window end on the standby database to see if the primary failed within peer window. If it did, you might

perform a failover with no data loss. Otherwise, the primary might or might not have committed transactions after the peer window has ended; you will need to consider the risk of data loss; instead of a failover, you might choose to repair and restart the failed primary.

For easy integration with cluster managers, a PEER WINDOW ONLY option is available for the TAKEOVER BY FORCE command. With this option, the command performs a failover only if the current time is earlier than the peer window end time. Upon detecting primary failure, the cluster manager can issue the TAKEOVER BY FORCE PEER WINDOW ONLY command to initiate a failover. Using this option maximizes the likelihood that an automated failover does not cause data loss. This option is recommended for automated failovers (the DB2 cluster manager uses this option). In such a configuration, the peer window time should be set to at least the amount of time the cluster manager needs to detect primary failure and react to the failure using failover. Use the following equation as an estimate:

> **hadr_peer_window** setting >= response time + safety margin + heartbeat interval
>
> where:
>
> response time = estimated time for automation software to detect failure and invoke HADR takeover
>
> safety margin = 5 sec, safety margin for primary-standby machine clock synchronization
>
> heartbeat interval = MIN(**hadr_timeout** value/4, hadr**_peer_window** value/4, 30 sec)

Figure 3 illustrates the rationale behind this equation. The time between the heartbeat and the primary failure is the *heartbeat interval* (this is the worst case scenario, where the primary fails right before it is going to send out another heartbeat). The time it takes for the database administrator (DBA) or the cluster manager to detect the failure and respond by issuing the TAKEOVER command is known as the *response time*. The "peer window end" timestamp is generated on the primary system and sent to the standby system via the heartbeat message. When it is compared with the current time on the standby during execution of the TAKEOVER BY FORCE PEER WINDOW ONLY command, a five-second *safety margin* is used in the comparison to handle less than perfectly synchronized primary and standby server clocks. (It is recommended that you synchronize the primary and standby server clocks using a method such as network time protocol.) Thus, the peer window size should be at least *heartbeat interval* + *response time* + *safety margin*

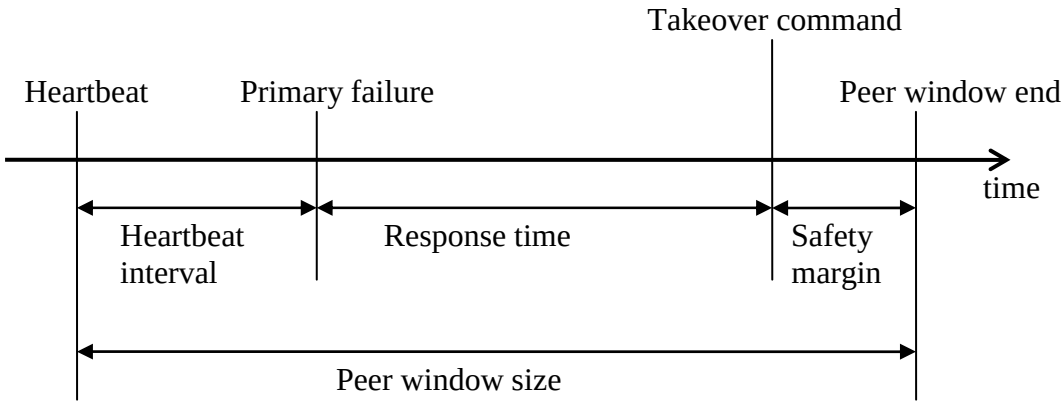

Figure 3. Peer window size

The takeover command should be issued before the peer window end time to ensure no data loss. After the peer window end time, you can still analyze peer window end (reported on primary and standby via the usual HADR monitoring interface such as database snapshot and db2pd) and compare it to primary failure time. As long as the primary failure time is earlier than peer window end, you can issue takeover for lossless failover. You should not use the "peer window only" option if you issue the command after peer window end. The PEER WINDOW ONLY option is intended for a cluster manager to do safe failover operations quickly after primary failures, providing minimal down time.

The “peer window end” message from the primary to the standby is implemented as a heartbeat message. Thus the heartbeat interval on peer-window-enabled systems takes peer window size into account, using the formula above. The peer window end on the standby can be one update interval (heartbeat interval) behind the peer window end on the primary, thus the above peer window setting equation adds a heartbeat interval to the window size.

The above equation only takes into account automatic failovers. Other factors, such as low tolerance to data loss, duration of possible network or standby failure, and time to detect and react to network partition might require a longer peer window. For example, if the network or standby fails, the primary will proceed to process transactions after the peer window expires, with no HA protection. A longer peer window might be needed to block the primary for DBA intervention if business requirements state that having a second copy of data on the standby is very important.

The default value of **hadr_peer_window** is zero, which means that the primary unblocks all waiting transactions and resumes logging as soon as the connection is broken. This provides maximum availability at the risk of losing data during a failover. This parameter can be tuned from zero to as long as years (effectively infinity), allowing the user to fine tune the balance between availability and risk of data loss. An extremely long peer window is essentially enforcing the rule "transaction commit on primary requires a second copy on standby system."

Even when availability is important, an **hadr_peer_window** value of at least a few seconds is recommended as it greatly reduces the chance of losing data in a rolling failure, where the network fails first, then the primary machine. During the interval, if the **hadr_peer_window** value is zero or very short, the primary can commit transactions not replicated to the standby. This data is lost in a failover. The default value of zero is for backward compatibility.

Once peer window is enabled, regardless of how the connection is broken, either from network error, standby failure, or exceeding the **hadr_timeout** value, or exceeding the DB2_HADR_PEER_WAIT_LIMIT registry variable setting, the primary blocks transactions for the configured duration. This ensures that "peer window only" failover on standby is safe in all cases. However, if the connection is closed on normal standby shutdown, the primary bypasses the peer window and resumes logging as soon as the connection is closed. The peer window and **hadr_timeout** are independent and take effect sequentially. Peer window specifies how long to hold primary transactions after a connection is broken, whereas **hadr_timeout** specifies the conditions for breaking the connection. Similarly, peer window and DB2_HADR_PEER_WAIT_LIMIT are independent because DB2_HADR_PEER_WAIT_LIMIT specifies the conditions for ending primary logging wait by breaking the connection.

When using a peer window, lower the heartbeat interval from the default value of 120s to 30s.

Tune DB2_HADR_PEER_WAIT_LIMIT

Setting the DB2_HADR_PEER_WAIT_LIMIT registry variable allows you to prevent primary database logging from blocking because of a slow or blocked standby database. The DB2_HADR_PEER_WAIT_LIMIT value determines how long the HADR primary database waits before breaking the connection, if logging on the primary database is blocked for the specified number of seconds in peer state. The primary can be waiting to send out the logs (waiting for network congestion to end), or waiting for ACK messages to come back from the standby after sending out logs.

Upon disconnection, if a peer window is enabled, the primary database continues to block logging for the peer window size duration; otherwise, it resumes logging immediately. Thus when DB2_HADR_PEER_WAIT_LIMIT is enabled (non-zero), the maximum client blocking time is equal to the DB2_HADR_PEER_WAIT_LIMIT value plus the **hadr_peer_window** value.

The main concern when setting the DB2_HADR_PEER_WAIT_LIMIT value is client response time. For both DB2_HADR_PEER_WAIT_LIMIT and HADR_PEER_WINDOW, a shorter value gives you better client response, but there might be a higher risk of losing data in a failover.

You can use DB2_HADR_PEER_WAIT_LIMIT and **hadr_timeout** together for targeted tuning. You can specify a short **hadr_timeout** to detect network problems and standby crash promptly, while setting a relatively long DB2_HADR_PEER_WAIT_LIMIT to keep replication enabled as long as the primary and standby are connected. Conversely, you can specify a short DB2_HADR_PEER_WAIT_LIMIT for better client response time,

while using a longer **hadr_timeout** to avoid frequent disconnections when the databases are not in peer state.

Tune DB2_HADR_SOSNDBUF and DB2_HADR_SORCVBUF

Use the DB2_HADR_SOSNDBUF (the socket send buffer size) and DB2_HADR_SORCVBUF (the socket receive buffer size) registry variables to optimize HADR TCP connections, without having an impact on other TCP connections on the system. The general rule for setting the TCP window size is TCP window size = send rate * round trip time.

On WAN systems, this likely means a setting that is larger than the system default because of the relatively long round trip time. If the window size is too small, the network cannot fully utilize its bandwidth, and applications like HADR experience throughput lower than the nominal bandwidth. A larger than necessary size usually causes no harm other than consuming more memory.

DB2_HADR_SOSNDBUF and DB2_HADR_SORCVBUF should have the same setting on the primary and standby databases.

The HADR simulator can be used to find out the optimal network setting between the primary and standby machines. Running the simulator takes only seconds, with no impact on the actual database systems. In contrast, updating these parameters on a database requires stopping and restarting the instance to make them effective. It is recommended that you experiment with the simulator first, then apply the values to the database. See http://www.ibm.com/developerworks/wikis/display/data/HADR_sim for more information on the simulator.

Client-server communication parameters

Tune DB2TCP_CLIENT_RCVTIMEOUT

The DB2TCP_CLIENT_RCVTIMEOUT registry variable specifies the number of seconds a client waits for data on a TCP/IP receive operation. When the wait time is exceeded, the client considers the connection down; if client reroute is enabled, client reroute is attempted. Setting DB2TCP_CLIENT_RCVTIMEOUT to a low value can minimize the wait time to detect that the HADR primary database is down. However, you should not set it to a shorter duration than the longest running query; otherwise you get a false alarm.

Database parameters

Set logfilsiz to a moderate size

Use a moderate size for your log files by setting the **logfilsiz** configuration parameter. The size should be usually no more than a few hundred MB. It can take a long time to create, archive, retrieve, or delete a very large log file, causing hiccups in system performance.

You should have an identical **logfilsiz** setting on the primary and standby. In order to maximize the likelihood that all replicated log files on the standby are interchangeable with those on the primary, **logfilsiz** setting on the primary is used on the standby, regardless of what the actual setting on the standby is. However, after a takeover the new primary (old standby) uses its local configuration value upon the first database reactivation after the takeover.

Set logindexbuild to ON

This setting for the **logindexbuild** configuration parameter specifies that your index creation, re-creation, or reorganization operations are to be fully logged. Although this means that index builds might take longer on the primary system and that more log space is required, the indexes are rebuilt on the standby system during HADR log replay and are available when a failover takes place. If **logindexbuild** is off, an index will be marked invalid when standby replays a build or reorg event because the log contains no data for the build event. If index builds on the primary system are not logged and a failover occurs, any invalid indexes that remain after the takeover is complete will have to be rebuilt before they can be accessed on the new primary.

You might not want index activity to be logged and replicated if any of the following statements apply:

- You are using the indexes temporarily (for example, on staging tables).
- The index data is very large and the table is not accessed often (in which case, you should also have the **indexrec** configuration parameter set to RESTART).
- You do not have enough active log space to support logging of the index builds.

You can override the database-level setting for **logindexbuild** at the individual table level by using the LOG INDEX BUILD table attribute.

Set indexrec to RESTART

When indexrec is set to RESTART, the new primary database searches for all invalid indexes and rebuilds them in the background at end of a takeover operation, Rebuilding the indexes does not prevent clients from connecting to the new primary database.

Tune softmax

The **softmax** configuration parameter setting determines the frequency of soft checkpoints and the recovery range, which help out in the crash recovery process. The setting for **softmax** should be a value that balances roll forward performance on the standby database with the crash recovery speed. Lower values reduce both the number of log records that need to be processed and the number of redundant log records that are processed during crash recovery, but lower values reduce roll forward performance as more frequent buffer pool flushing is required.

Database parameters related to client reroute

Set DB2_MAX_CLIENT_CONNRETRIES and DB2_CONNRETRIES_INTERVAL

Setting the DB2_MAX_CLIENT_CONNRETRIES and DB2_CONNRETRIES_INTERVAL registry variables allow you to configure the retry behavior for automatic client reroute. The first variable specifies the number of times to retry the connection, while the second specifies the sleep time between consecutive retries. If both of these variables are not set, the automatic client reroute feature retries the connection to a database repeatedly for up to 10 minutes.

Database administration and maintenance

Choosing the appropriate reorganization method

Both online and offline reorganizations are replicated operations. However, the method in which they are replicated differs and can influence the choice of operation. Choose a reorganization method based on whether you want to keep the affected tables and indexes available during the operation.

Perform an online reorganization to maintain availability of the affected tables and indexes

During an online (or inplace) reorganization, all operations are logged at a fine granularity. Changes to the table or index are logged and replicated as the reorganization progresses. As a result, HADR can replicate the operation at a more even pace than offline reorganization. However, there is a potentially large impact to the system from the amount of logs generated by the operation.

If there is a failover during an online reorganization, the reorganization cannot be resumed on the new primary database, as the status file is stored outside of the database and is not replicated. However, the reorganization can be restarted on the new primary database.

Refer to the "Minimizing Planned Outages" Best Practice paper for a detailed discussion on reorganization operations and availability. This paper and others are available at the DB2 Best Practices website at http://www.ibm.com/developerworks/db2/bestpractices/.

Perform an offline reorganization if the affected tables and indexes can be unavailable

An offline (or classic) reorganization has much less granularity of replication than online reorganization; however, the impact depends on whether the operation is reclustering using an index.

In a clustering reorganization, operations are typically logged per hundreds or thousands of affected rows. If your database is in peer state, this can cause intermittent blocking of primary transactions, as the standby must replay many updates at once for a single log record. Standby replay can be blocked long enough to fill up standby receive buffer, causing primary blockage.

A non-clustering reorganization generates a single log record after the reorganization is completed on the primary. This method has the greatest impact on the HADR pair, as the standby performs the entire reorganization from scratch after receiving the critical log record from the primary. The operation is effectively performed twice in a serial fashion, first on the primary, then on the standby.

Because the standby must replay the reorganization record, its log receiving buffer can fill up and it will stop receiving more logs from the primary, causing primary logging to be blocked if the HADR pair is in peer state.

If you have to perform large offline reorganizations, consider one of the following recommendations:

- **Temporarily disable HADR in order to avoid blocking the primary when the standby replays the reorganization**. Do this by deactivating the standby database before reorganization on primary starts. After the reorganization completes on the primary, reactivate the standby database and allow it to catch up to the primary. The reorganization records will be replayed in catchup states, which does not block primary logging. However, deactivating the standby results in a time window in which there is no high availability or disaster recovery available. Alternatively, you can run “stop HADR” and “start HADR” on the primary instead of deactivating the standby and reactivation standby to keep the primary out of peer state during reorg.

- **Set the HADR_PEER_WAIT_LIMIT value to automatically break the primary out of peer state.** The advantage is that this does not require any manual intervention on your part. However, the disadvantage is that the auto break is triggered only after transactions on the primary have been blocked for the period specified by the HADR_PEER_WAIT_LIMIT value.

Performing load operations in an HADR environment

Load operations require special consideration in an HADR environment, because only a successful LOAD COPY YES is replicated to the standby. All other types of LOAD operations might result in the standby database no longer being a viable candidate for failover because data is inconsistent between the HADR pair. To prevent this from happening, implement the recommendations in the sections that follow.

Perform a nonrecoverable load only if the load operation does not need to be replicated on the standby

In some cases, you might not want the table replicated to the standby (for example, if the table is a staging table that will be dropped). Nonrecoverable loads result in the target table being marked bad and the standby skips future log records regarding this table. To monitor this, look for an ADM5571W message in the db2diag.log on the standby.

If a nonrecoverable or failed LOAD … COPY YES operation marks a table space bad, identify potential issues affecting the viability of the standby database for failover operations. Issue the db2pd –db <dbname> -tablespaces command to identify table spaces in non-normal table space states. Look in the "State" column in the Tablespace Statistics report generated for non-zero values. Focus on states that would impact HADR replay (e.g., 0x100 – restore pending, 0x80 – roll forward pending, or 0x4000 – offline, and not accessible).

Ensure that the load copy is available to the standby when it replays the load

After a successful LOAD ... COPY YES operation completes, the data is replicated on the standby only if the standby can access the copy through the path or device specified in the load command. Consider using a shared copy device such as NFS for this. If you transfer the copy by means such as physically transferring a tape, it is recommended that the standby be stopped before the primary starts the load. Once the primary finishes the load and the copy is available to the standby, restart the standby. The standby will then reconnect to the primary and replay the load.

If the load copy cannot be accessed on the standby, the table space in which the table is stored is marked bad on the standby and the standby skips future log records regarding this table space. In this situation, the standby database has to be rebuilt from a backup image taken after the load operation was issued on the primary. To monitor for this condition, scan the db2diag.log on the standby for messages that identify the table for which LOAD was run, the inaccessible path where the copy was placed, and a message for each table space marked bad. If a failover occurs before you detect this condition, scan the new db2diag.log on the primary for messages that indicate which table spaces are in non-normal states.

Set the DB2_LOAD_COPY_NO_OVERRIDE registry variable to COPY YES if there will be frequent load operations

When the registry variable DB2_LOAD_COPY_NO_OVERRIDE is set to "copy yes <copy destination>", all LOAD … COPY NO operations are converted to LOAD … COPY YES, using the copy destination specified in the registry variable.

Best Practices

- **Perform an infrastructure analysis**
- Use separate, high performing disks or file systems for the database logs
- Make the location of archived logs accessible by both the primary and standby databases
- Use a dedicated network for the HADR primary-standby connection
- Use multiple network adapters
- Consider using a virtual IP address for the database server or using automatic client reroute
- Choose the appropriate **hadr_syncmode**
- Tune DB2_HADR_BUF_SIZE, **hadr_timeout**, **hadr_peer_window**, and DB2_HADR_PEER_WAIT_LIMIT
- Tune DB2TCP_CLIENT_RCVTIMEOUT, DB2_HADR_SOSNDBUF, and DB2_HADR_SORCVBUF
- Tune **logfilsiz** and **softmax**
- Set **logindexbuild** to ON
- Set **indexrec** to RESTART
- Set DB2_MAX_CLIENT_CONNRETRIES and DB2_CONNRETRIES_INTERVAL for client reroute
- Perform an online reorganization if you want to maintain availability of the affected tables and indexes

- Perform an offline reorganization if it is not necessary to maintain the availability of the affected tables and indexes
- Perform a nonrecoverable load only if the load operation does not need to be replicated on the standby
- Ensure that the load copy is available to the standby when it replays the load
- Set the DB2_LOAD_COPY_NO_OVERRIDE to COPY YES if there will be frequent load operations

Conclusion

While this set of best practices is not exhaustive, it focuses on two of the key aspects of the HADR feature: the network and the logging. We have provided specific recommendations, but as was noted earlier, there is no one-size-fits-all setup for HADR. Instead, we have given guidance for how to tune various parameters in order to find what works best for your particular environment.

Further reading

- DB2 Best Practices
  http://www.ibm.com/developerworks/db2/bestpractices/
- DB2 9 for Linux, UNIX and Windows manuals
  http://www.ibm.com/support/docview.wss?rs=71&uid=swg27009474
- DB2 9.5 for Linux, UNIX, and Windows Information Center
  http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp
- DB2 9.7 for Linux, UNIX, and Windows Information Center
  http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp

Contributors

Steven R. Pearson
DB2 Development

Tim Vincent
Chief Architect DB2 for Linux, UNIX, and Windows

Alexandria Burkleaux
DB2 Development

Ilker Ender
DB2 Development

Louise McNicoll
DB2 Information Development

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites 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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample 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.