IBM solidDB
Version 7.0

High Availability User Guide

SC27-3843-05

Note
Before using this information and the product it supports, read the information in "Notices" on page 203.

First edition, fifth revision

This edition applies to V7.0 Fix Pack 8 of IBM solidDB (product number 5724-V17) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

Summary of changes

Changes for revision 05

- New parameter **HotStandby.NetcopyRpcCompress** added in section HotStandby section.
- New error message `30794,HSB,Message,Invalid HotStandby.Connect option -z. -z option is not supported.` added in section HotStandby errors and status codes.

Changes for revision 04

- Examples for the parameter values of **HotStandby.Connect** parameter corrected in sections HotStandby quick start procedure and Defining primary and secondary node HotStandby configuration. The **HotStandby.Connect** parameter must include the node name or IP address of the other computer in the HSB pair.
- New High Availability Controller (HAC) parameters added in section High Availability Controller (HAC) parameters
  - **HAController.NetcopyErrorLevel**
  - **HAController.NetcopyWarningLevel**

Changes for revision 03

- Previously undocumented High Availability Controller (HAC) commands added in section High Availability Controller commands (ADMIN COMMAND):
  - **hac shutdowndb**
  - **hac restartdb**

Changes for revision 02

- New parameter **HotStandby.NetcopyReceiveBufferSize** added in section HotStandby section.
- New simplified hot migration procedure added in section Hot migration procedure. As of 7.0 Interim Fix 2 / Fix Pack 3, the hot migration procedure no longer requires a netcopy to synchronize the servers during the upgrade.

Changes for revision 01

- Sections Configuring SMA TC with HotStandby and Syntax of Transparent Connectivity Info – ODBC updated with information about specifying load balancing method when using SMA with Transparent Connectivity. When using SMA with TC, the load balancing method must be set to LOCAL_READ. If READ_MOSTLY or WRITE_MOSTLY is specified, a network connection is used instead of the SMA connection.
- Section Hot migration procedure updated: instead of `-xmigratetohsbg2`, the option `–xconvert` or `–xautoconvert` is used in hot migration.
- Section Syntax of Transparent Connectivity Info – ODBC updated: encryption attribute `[USE_GSKIT={YES|NO}]` added.

About this manual

The IBM® solidDB® High Availability (HotStandby) component increases the reliability of your database system, reducing downtime. HotStandby uses a "hot standby" approach, in which a second database server runs in parallel with the primary server and keeps an exact up-to-date copy of the data. If the primary database server fails, the High Availability Controller (HAC) makes a switch over to the secondary, transparently to applications and with no loss of committed transactions, and with minimal performance impact. Switchover times can be quite fast — as short as a couple of hundred milliseconds, depending upon the characteristics of your hardware and software environment.

This guide contains information specific to the HotStandby component only. For general administration and maintenance information about solidDB databases, see *IBM solidDB Administrator Guide*.

This guide assumes the reader has general database management system (DBMS) knowledge, and familiarity with SQL and solidDB.

Illustration conventions

This document contains several server diagrams depicting different scenarios in the HotStandby environment.

The table below provides an illustration key for the server diagrams:.

Table 1. HotStandby illustration conventions

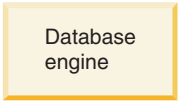
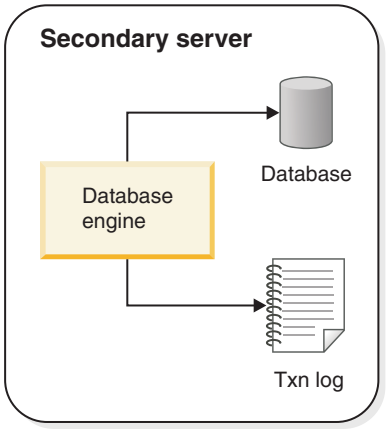
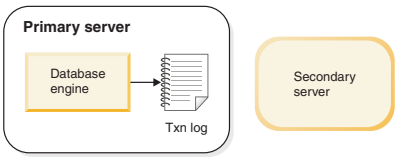
Symbol	Description
	The rectangle represents the executing program, that is, the database server (engine) itself.
	The cylinder represents the data, generally stored on disk. Alternatively, some or all may be stored in memory.
	This symbol represents the transaction log (Txn Log), which is used in both database recovery and HotStandby.

Table 1. HotStandby illustration conventions (continued)

Symbol	Description
	A rounded rectangle represents a complete server with data and Txn Log. If the phrase **Secondary server** or **Primary server** is inside the rounded rectangle, then the server is a HotStandby server.
	For simplicity, in some cases the cylinder that represents the data in the database is omitted. In some cases, the symbol is simplified even further to show just the rounded rectangle. Both icons are simplified representations of a server.

Typographic conventions

solidDB documentation uses the following typographic conventions:

Table 2. Typographic conventions

Format	Used for
Database table	This font is used for all ordinary text.
NOT NULL	Uppercase letters on this font indicate SQL keywords and macro names.
`solid.ini`	These fonts indicate file names and path expressions.
`SET SYNC MASTER YES;` `COMMIT WORK;`	This font is used for program code and program output. Example SQL statements also use this font.
`run.sh`	This font is used for sample command lines.
TRIG_COUNT()	This font is used for function names.
java.sql.Connection	This font is used for interface names.
`LockHashSize`	This font is used for parameter names, function arguments, and Windows registry entries.
argument	Words emphasized like this indicate information that the user or the application must provide.

Table 2. Typographic conventions (continued)

Format	Used for
Administrator Guide	This style is used for references to other documents, or chapters in the same document. New terms and emphasized issues are also written like this.
File path presentation	Unless otherwise indicated, file paths are presented in the UNIX format. The slash (/) character represents the installation root directory.
Operating systems	If documentation contains differences between operating systems, the UNIX format is mentioned first. The Microsoft Windows format is mentioned in parentheses after the UNIX format. Other operating systems are separately mentioned. There may also be different chapters for different operating systems.

Syntax notation conventions

solidDB documentation uses the following syntax notation conventions:

Table 3. Syntax notation conventions

Format	Used for
`INSERT INTO` *`table_name`*	Syntax descriptions are on this font. Replaceable sections are on *this* font.
`solid.ini`	This font indicates file names and path expressions.
[ ]	Square brackets indicate optional items; if in bold text, brackets must be included in the syntax.
|	A vertical bar separates two mutually exclusive choices in a syntax line.
{ }	Curly brackets delimit a set of mutually exclusive choices in a syntax line; if in bold text, braces must be included in the syntax.
...	An ellipsis indicates that arguments can be repeated several times.
. . .	A column of three dots indicates continuation of previous lines of code.

1 Introduction to IBM solidDB HotStandby

The goal of high availability (HA) systems is to make system failures tolerable. To implement high availability, the solidDB HotStandby component enables a secondary server (a *standby server*) to run in parallel with the primary server (an *active server*) and keep an up-to-date copy of the data in the primary server.

The HotStandby component implements an internal state machine that makes the server aware of the HA state. The HA state machine makes it possible to preserve database consistency. For example, when the server is in the Secondary state (that is, receiving a transaction stream from the Primary), updates to the Secondary database are disabled.

The availability or states of the servers can be controlled with a *watchdog* program. The solidDB server provides a watchdog implementation called the solidDB High Availability Controller (HAC).

Internally, HAC uses a set of HotStandby commands (HotStandby API) to control the server states. A solution such as this allows implementation of systems that have increased reliability. A failed database server no longer brings your site to a complete halt. In as little as a few hundred milliseconds, in any engine configuration supported by solidDB (such as solidDB advance replication with master and replica servers), HotStandby allows the secondary database to replace the failed one.

HotStandby architecture

HotStandby includes the following components:

- Watchdog application (for example, HAC)
- HotStandby API (HSB admin commands)
- Primary and Secondary solidDB servers

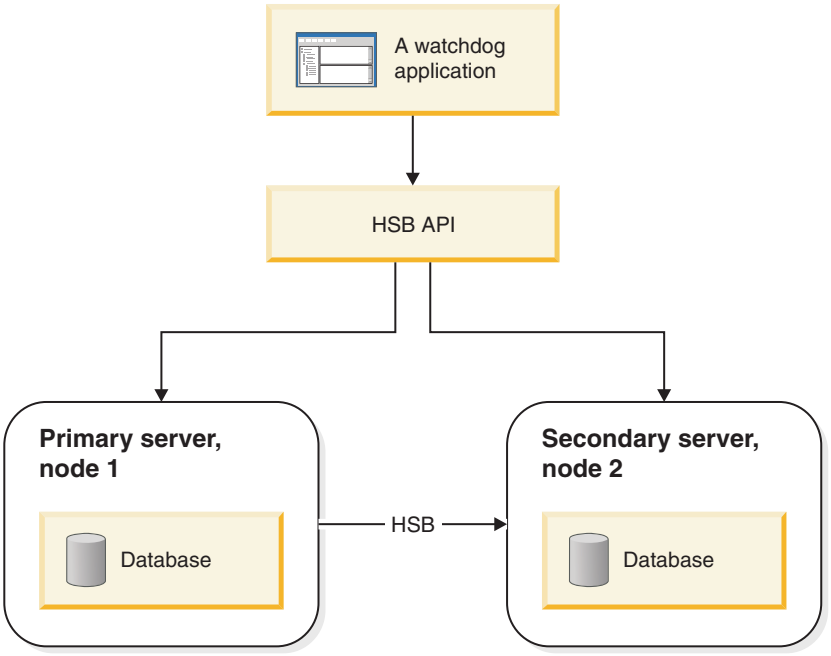


Figure 1. HotStandby architecture

Principles of operation

HotStandby (HSB) performs synchronous transaction replication between two nodes: a primary server and a secondary server.

The primary server node (*Primary*) contains the active database. The secondary server node (*Secondary*) contains an exact, up-to-date copy of the active database, and it can replace the Primary if the Primary fails.

The Secondary receives updates from the primary server, and is ready to take over as the Primary if the original Primary fails. An additional benefit of having the Secondary is that the Secondary can also respond to read-only requests (for example, SELECT statements) from clients. This allows you to spread some of your workload over two servers rather than one.

Note: The term "hot standby" (two words, all lower case) refers to the general technique of having a second server ready to take over if the first server fails. "HotStandby" (one word, capitalized as shown) refers to solidDB's specific implementation of this general technique. The abbreviation for HotStandby is HSB.

Similarly, a watchdog refers to a technology that supervises the state of two databases and can switch the states, if necessary. HAC is the solidDB watchdog implementation. You can also use the HSB API to program your own watchdog application. The solidDB package also includes a watchdog sample that you can use as a starting point.

1.1 Key features of HotStandby

The HotStandby key features include failover support, transaction logging, and load balancing.

1.1.1 HotStandby API (HSB admin commands)

The high availability behavior is controlled with an API that is based on a subset of solidDB admin commands.

This subset of commands is identified by the command prefix **hotstandby**, abbreviated as **hsb**. These commands can be issued by using any SQL-capable tool (like **solsql**) or programmatic interfaces like ODBC or JDBC. The HotStandby admin commands have the following syntax:

```
admin command 'hotstandby hsb-command options';
```

or

```
admin command 'hsb hsb-command options';
```

The **hotstandby** admin commands control the high availability state of a solidDB HSB server, or retrieve state information. The commands can be issued manually or programmatically. The solidDB HA management tools, High Availability Controller (HAC), and the Watchdog sample use the commands programmatically.

1.1.2 Basic HotStandby server scheme

In the basic HotStandby server scheme, there are two database servers, the *Primary* and the *Secondary* server. Both servers have their own disk drives on which they stores the database, and each of which have their own transaction logs (*Txn Log*).

Figure 2 illustrates the basic HotStandby server scheme. The Primary writes to its transaction log and forwards it to the Secondary so that the Secondary can make the same changes to its copy of the database. The transaction log on the Secondary is not actively involved in HSB, but it is maintained so that the Secondary can recover data that was committed but not yet written to the main data tables.

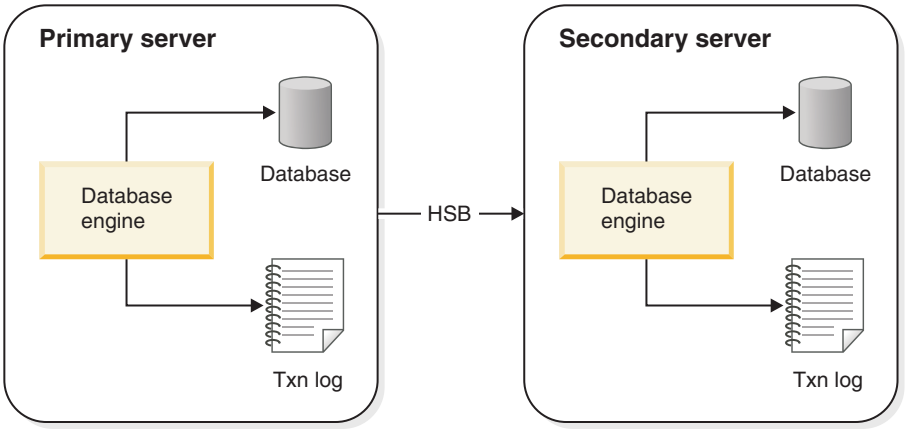


Figure 2. HotStandby server scheme

Heartbeat

Internally, solidDB HSB uses a technique, which is referred to as *heartbeat*, to monitor the connection between servers.

Sequences of keepalive messages are sent between active and standby servers. Both servers continuously send the unidirectional "I am here" messages to the other server. The messages are sent on a fixed time interval. A message from the other server is expected to arrive within a predefined time window. In solidDB, the heartbeat technique is called *ping*.

Important: In solidDB the heartbeat technique is called *ping*, although there are no ping requests sent. The heartbeat technique is different from the Ping protocol used in Internet Protocol networks.

The transaction log and HotStandby

HotStandby uses the Primary server transaction log, which contains a copy of the transactions that are committed on the server. In a non-HotStandby server, this transaction log is used to recover data if the server shuts down abnormally.

In a HotStandby Primary server, the log data is also sent to the Secondary server so that the Secondary knows what data to update. The Secondary database runs a continuous roll-forward process that receives the log data and keeps the copy of the data on the Secondary synchronized with the Primary.

If the Primary server fails, a watchdog application tells the secondary to become the Primary. When the new Primary is in operation, the clients can connect to it and continue working. Clients continue to see all data that was committed before the Primary went down. (Clients must restart any transactions that were started but not finished when the original Primary server went down.)

A special type of client connectivity called *Transparent Connectivity (TC)* is available for clients in the HSB environment that calls for handling of failovers and switchovers. For more information, see 4, "Using HotStandby with applications," on page 73.

If the Secondary server fails, the Primary can continue to operate. It continues writing data to the transaction log and keeps that transaction log until the Primary and Secondary are reconnected to each other and the Primary has sent the log to the Secondary. The exact length of time that the Primary keeps the log depends upon the settings of the `solid.ini` configuration parameters **`General.CheckpointDeleteLog`** and **`General.BackupDeleteLog`**.

1. If **`General.CheckpointDeleteLog`**=`Y`, the Primary keeps all transaction logs since the time that the Secondary went down or since the most recent checkpoint, whichever is less recent.
2. If **`General.CheckpointDeleteLog`**=`N` and **`General.BackupDeleteLog`**=`Y`, the Primary keeps all transaction logs since the time that the Secondary went down or since the most recent backup, whichever is less recent.
3. If **`General.CheckpointDeleteLog`**=`N` and **`General.BackupDeleteLog`**=`N`, the server keeps the logs indefinitely.

When the failed database server becomes available again, it can be configured to become the new Secondary database server (the server that did not fail is already acting as the current Primary).

If the Primary server is the server that fails, then the servers will reverse their responsibilities, with the original Secondary taking over as the Primary, and the original Primary coming back into the system as the new Secondary after it is repaired. These reversals can happen each time there is a failure. The fact that either server can be the Primary allows the system to survive multiple failures over time, and continue operating indefinitely.

Note: If the Primary server is unable to contact the Secondary server for a long time, the transaction log can fill all the available disk space. You can avoid running

out of disk space because of large log files by applying appropriate configuration parameter settings. For more information, see 3.6.3, “Running out of space for transaction logs,” on page 67.

You can use HSB to reduce downtime during hardware and software upgrades. You can leave one server to run as Primary while you upgrade the other.

HotStandby can also be used to help choose a customized balance of speed and safety. The HSB parameters **HotStandby.SafenessLevel** and **HotStandby.2SafeAckPolicy** control the way the Secondary server acknowledges the transactions. Together with the logging-related **Logging.DurabilityLevel** parameter, these parameters let you specify a combination of speed and safety. Some parameter settings can increase performance over non-HSB servers. (For more information, see the discussion of durability level and safeness parameters in 3.5, “Performance tuning,” on page 65.)

You can also configure the safeness level to change dynamically in relation to the durability level by using the **HotStandby.SafenessLevel=auto** setting.

Failover

In a failover, the secondary is switched to be the new primary.

There are several reasons for switching the secondary to new primary:

1. The primary fails-
2. You want to administer the primary.
3. You must choose a primary when there is no existing primary on the system.

The secondary is switched to be the new primary by issuing the following command on the Secondary server:

```
ADMIN COMMAND 'hotstandby set primary alone';
```

In the case of a failover, the new Primary contains the up-to-date committed data from the old Primary database. Everything that was committed in the Primary database, can be found from the Secondary database. If Transparent Connectivity (TC) is used, connections are not lost on the failover. However, the ongoing transactions are aborted and must be re-executed. For more information, see 4.2.2, “Failure transparency in Transparent Connectivity,” on page 87. The new Primary can operate alone and continue to write transactions and data to its database and transaction log.

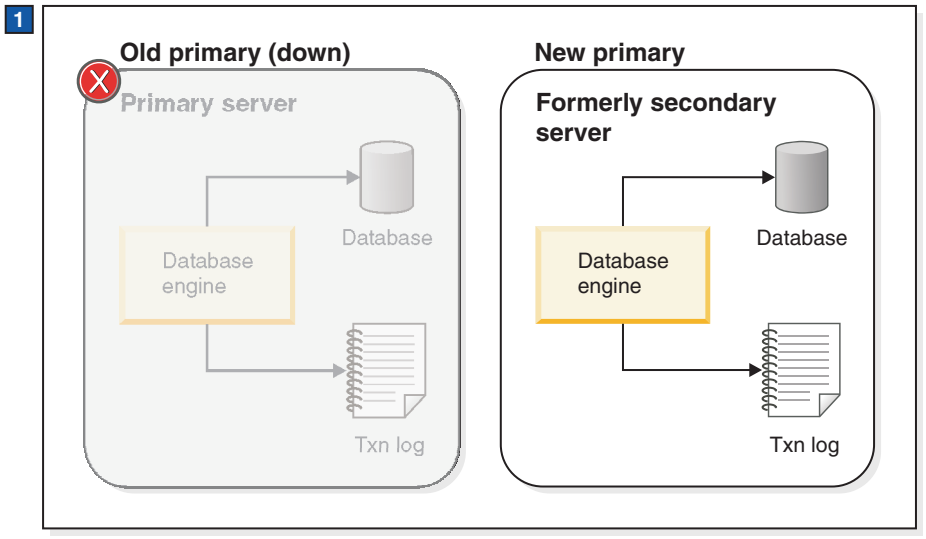


Figure 3. HotStandby switchover to new Primary (old Secondary)

1. The server that was originally the Secondary becomes the new Primary after the old Primary server fails.

Server Catchup

When the old Primary is back online, assuming that there is an existing Primary, it becomes the new Secondary. At this stage, the information in the new Secondary lags behind that of the new Primary as new transactions are committed to the new Primary database. To bring the new Secondary up to date, the transaction log data of the new Primary is sent to the new Secondary automatically after the servers are connected. All pending changes are written from the transaction log to the new Secondary so that the Secondary can keep in sync with the Primary. Server catchup is illustrated in Figure 4 on page 7.

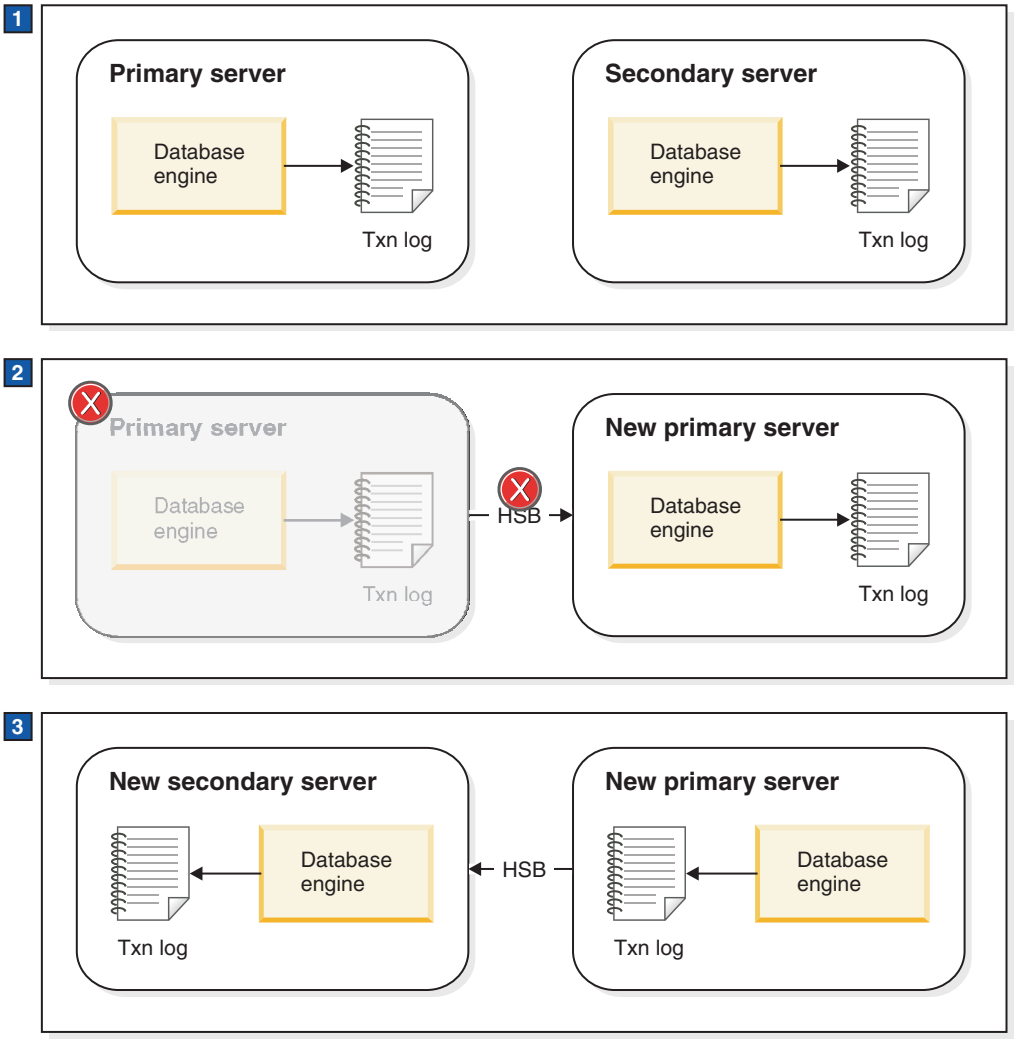


Figure 4. Server failover and catchup example

1. Normal operation: Primary server sends data to Secondary server.
2. When Primary server fails, Secondary server takes over as the new Primary. New Primary server saves transaction information in its transaction log so that it can send the data to the new Secondary server later.
3. After the old Primary server is brought up as the new Secondary server, the information in the transaction log of the new Primary is sent to the new Secondary so that it can catch up.

1.1.3 Server HotStandby states

In a HotStandby system, each server is in one of several possible states that describes the current behavior of the server.

For example, when the Primary and Secondary are communicating and synchronizing, they are in the PRIMARY ACTIVE and SECONDARY ACTIVE states.

Alternatively, if the Primary loses contact with the Secondary, the Primary switches to the PRIMARY UNCERTAIN state automatically. In that state, it does not accept new transactions. The user or, more typically, the HAC can switch the server to the

PRIMARY ALONE state, in which the server acts as an independent server–it accepts new transactions and stores them to send to the Secondary later.

Description of server states

Both servers in an HotStandby (HSB) pair have *states* that can be queried and manipulated.

Table 4. Description of server states

STATE	DESCRIPTION
PRIMARY ACTIVE	The servers are connected, and this server (the Primary server) is accepting read/write transactions and sending the data to the Secondary server. The Secondary server must be in SECONDARY ACTIVE state.
PRIMARY ALONE	The peer servers are not interconnected. The peer might be up, but it is not connected and therefore is not accepting any transactions. The peer might be in the SECONDARY ALONE state). This server (the Primary) is actively accepting and executing read/write transactions and collecting them to be sent to the Secondary later.

Table 4. Description of server states (continued)

STATE	DESCRIPTION
PRIMARY UNCERTAIN	The servers have disconnected abnormally and the **AutoPrimaryAlone** configuration parameter is set to No. In the PRIMARY UNCERTAIN state, any unacknowledged transactions remain in a pending status, which means that the server does not commit or roll back the transaction until HAC changes the server to another state. The operator has three possible actions: reconnect the Primary to the Secondary, set the Primary server to PRIMARY ALONE state, or set the Primary server to SECONDARY ALONE state. 1. If the server is reconnected to the Secondary, then the transactions are committed on the Primary. 2. If the state is changed to PRIMARY ALONE, then the open transactions are committed on the Primary. 3. If the state is changed to SECONDARY ALONE, then the open transactions remain pending. They are finally resolved after the server changes to another state. For example, if the server is moved to the SECONDARY ACTIVE state, the blocked transactions are aborted or committed, depending on the catchup outcome. If the server state is changed to STANDALONE or PRIMARY ALONE, then the blocked transactions are committed. If you want the Primary server to automatically go to PRIMARY ALONE rather than PRIMARY UNCERTAIN when it loses contact with the Secondary, then read the description of the **AutoPrimaryAlone** configuration parameter. **Note:** HAC can maximize safety by always switching the server from PRIMARY UNCERTAIN to SECONDARY ALONE. This prevents the possibility of dual primaries. However, it also prevents users from updating data on the server. For more information, see 3.6.2, “Network partitions and dual primaries,” on page 66.

Table 4. Description of server states (continued)

STATE	DESCRIPTION
SECONDARY ACTIVE	The peer servers are interconnected, and this server is accepting incoming transaction log data from the Primary. The transactions are executed on the Secondary so that it has the same data as the Primary. The transactions are also written to the transaction log of the Secondary so that the Secondary itself can recover the data if the Secondary fails. Additionally, clients can perform read-only transactions on a server in the SECONDARY ACTIVE state. When a server is in the SECONDARY ACTIVE state, the server's peer must be in PRIMARY ACTIVE state.
SECONDARY ALONE	The Secondary is disconnected from its peer server. Only read requests are accepted. The server can be connected to the peer by issuing the command **HotStandby connect** on either the Secondary or the Primary.
STANDALONE	The server has no HSB state (Primary or Secondary) and operates in the way a regular stand-alone server operates. Transaction logs are processed and removed in the normal way, too; they are not saved for the Secondary. To resume HSB operation, the server must be set to either PRIMARY ALONE or SECONDARY ALONE, and the Primary has to do a **netcopy** or **copy** operation to send a complete copy of the database to the Secondary.
OFFLINE	The server was started in "netcopy listen mode" (also called "backupserver mode"). In this mode, the server is waiting for an incoming **netcopy** from a server that is in PRIMARY ALONE state. When the server successfully completes **netcopy**, the server moves to the state SECONDARY ALONE. You cannot directly observe the OFFLINE state because a server in OFFLINE state does not accept client connections. If you attempt to connect to a server in the OFFLINE state, error code `14552 Server is in backup server mode, no connections are allowed` is returned. A server in the OFFLINE state responds only to a **netcopy** operation.

1.1.4 Replication modes in HotStandby

The purpose of a HotStandby replication protocol is to carry the transaction results safely from the primary server to the secondary server. solidDB offers both synchronous (*2-safe*) and asynchronous (*1-safe*) replication protocols. Together with various durability (logging) levels, the replication protocol t can be used to tune the HotStandby system to the required balance between performance and endurance.

- `1-safe`: the transaction is first committed at Primary and then transmitted to Secondary
- `2-safe`: the transaction is not committed before it has been acknowledged by Secondary (default).

The safeness level can be controlled at three levels:

- Global – **`HotStandby.SafenessLevel`** parameter
- Session – SET SAFENESS statement
- Transaction – SET TRANSACTION SAFENESS statement

Synchronous HotStandby with 2-safe replication

To ensure that the Primary and Secondary have the same data, solidDB uses, primarily, a *synchronous HotStandby* model. It is called a *2–safe* replication method; the data is written in two places before the user is told that the data has been committed.

Before committing changes to a transaction in the Primary database, the Primary server sends the transaction data to the Secondary server. The Secondary server must send acknowledgement to the Primary that it has committed or at least received the data. Otherwise, the Primary server times out and changes its state from PRIMARY ACTIVE to PRIMARY UNCERTAIN. In this case, the Primary server cannot roll back or commit the transaction. The HAC can set the Primary server to PRIMARY ALONE state, which allows the Primary to continue to receive transactions and operate independently of the Secondary. It commits the pending transactions that were sent to the Secondary and resumes accepting new transactions.

Note: The Secondary server sends an acknowledgement as soon as it has committed (or at least received) the transaction log entries. This configuration prevents lost transactions when there is a single failure. Additionally, a file-based transaction log is optionally retained to facilitate database recovery in case a total system failure occurs.

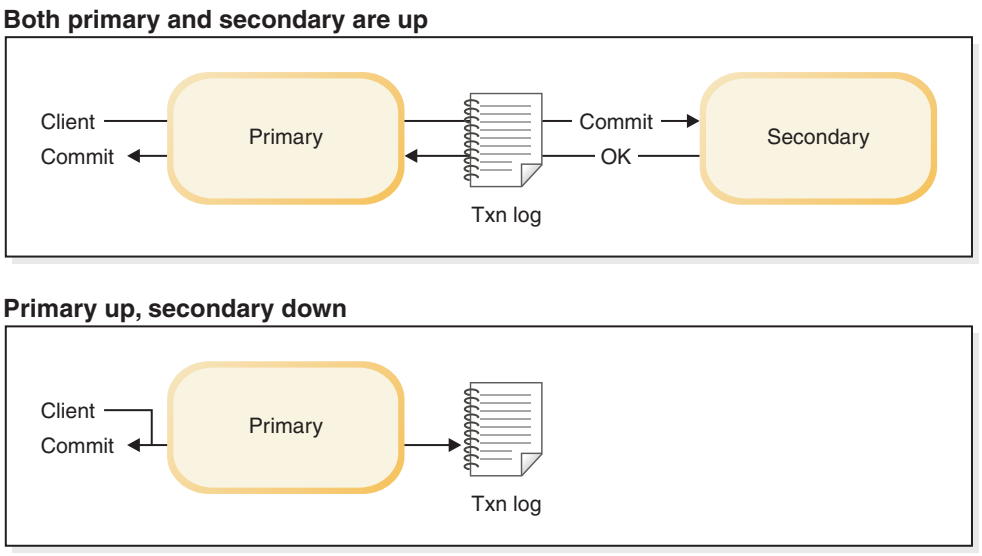


Figure 5. Synchronous HotStandby configuration

Basic steps in sending data

Sending data with synchronous replication includes the following steps:

1. The Primary server writes data (in record level format) to the transaction log at the Primary node.
2. When the Primary server encounters a commit statement, all changed data is sent to the Secondary server.

   Note: If the Secondary server fails after the transaction starts and before the Primary sends the data, then the Primary will roll back the transaction.
3. The Secondary acknowledges the commit message. The timing of the acknowledgement depends upon the setting of the **HotStandby.2SafeAckPolicy** configuration parameter. With the fastest alternative, *2-safe received*, the Secondary sends acknowledgement to the Primary immediately upon receiving the commit message. With the safest alternative, *2-safe durable*, the Secondary sends acknowledgement after it has executed and written the transaction durably to its own transaction log.

   When the Primary receives the acknowledgement, the Primary notifies the user that the data has been committed.
4. If the Primary does not receive acknowledgement from the Secondary (for example, because of a network failure or node failure), the Primary server times out and switches to the PRIMARY UNCERTAIN state. The Primary is unable to roll back or commit the transaction itself because it does not know the state of recent transactions in the Secondary. The Primary does not know which of the following happened:
   - The Secondary was down before the transaction was committed.
   - The Secondary already committed the transaction, but the Primary server did not receive acknowledgement, for example because of network failure.

   While the server is in PRIMARY UNCERTAIN state, the current transaction and new transactions that a user tries to commit are blocked and the user might perceive that the server is unresponsive.
5. If the HAC detects that the Secondary is down or the network has failed, it can switch the Primary server to the PRIMARY ALONE state. When the Primary server is set to PRIMARY ALONE, it commits the pending transactions that were sent to the Secondary and resumes accepting new transactions.
6. Changes are accumulated to the transaction log file until the Secondary server is back in operation or until the Primary server is out of disk space. If the server runs out of disk space for the transaction log, the Primary changes to read-only mode.
7. If the Secondary server is out of operation for a long time and the server is likely to run out of disk space for the transaction log, you might want to switch the Primary server from PRIMARY ALONE to STANDALONE state. This means that the transaction log does not store all transactions since contact was lost with the Secondary, and therefore the Secondary cannot catch up only by reading the transaction logs from the Primary. If the Secondary cannot be brought up to date with the transaction logs, the only way to synchronize the Secondary with the Primary is to copy its database files to the Secondary. This can be done with the **hotstandby netcopy** command.
8. To execute **hotstandby netcopy**, the Primary must be in the PRIMARY ALONE state. After **hotstandby netcopy**, the Primary server remains in the PRIMARY ALONE state, regardless of whether the command succeeds or fails.
9. In order for the Primary to start sending its transactions to the Secondary again, the Primary server must be connected explicitly to the Secondary server

by using the command **hotstandby connect**. After the Primary server is connected to the Secondary server, the Primary operates in the PRIMARY ACTIVE state.

After the servers are connected, they start performing catchup – when all pending changes are automatically written from the transaction log to the Secondary to keep in sync with the Primary. Before server catchup, the Primary and Secondary exchange information and determine where to begin the catchup so that a transaction is not committed twice on the Secondary.

Related tasks:

"Copying a primary database to a secondary over the network" on page 54
To send a copy of the database file from the Primary server to the Secondary server, use the **netcopy** command. The Secondary server must already be running.

"Copying a database file from the primary server to a specified directory" on page 57
If the directory that the Secondary uses for the database is visible to the Primary, you can use the **hotstandby copy** command to copy the database from the Primary to the Secondary.

3.4.6, "Connecting HotStandby servers," on page 59

Asynchronous HotStandby with 1-safe replication

Optionally, asynchronous replication from Primary to Secondary can be used. This is called *1-safe replication*.

With 1-safe replication, the transactions are acknowledged immediately after they have been committed at the Primary. This offers significant performance gains. After the commit, the transactions are sent to the Secondary, in an asynchronous way. The trade-off is that, when a failure occurs at Primary, a few transactions that were in transfer, might be lost.

Either of the two replication methods can be chosen dynamically, or even per session or transaction. The replication delay that is involved with 1-safe replication can be controlled, too.

1.1.5 Durability and logging

The durability level controls how solidDB handles transaction logging. The solidDB server supports three durability levels: strict (safe), relaxed (fast), and adaptive. Relaxed durability yields best performance while strict durability minimizes loss of transactions. The adaptive durability level is available only in HotStandby configurations.

- *Strict durability*: If a transaction is written to the transaction logs as soon as the transaction is committed, this is called "strict durability". This type of durability maximizes safety.
- *Relaxed durability*: If the server is permitted to defer the transaction write until the server is less busy, or until it can write multiple transactions together, this is called "relaxed durability" (or "relaxed logging"). If you use relaxed durability in a server that is not part of an HSB pair, you risk losing the most recent few transactions if the server terminates abnormally. If the server is part of an HSB pair, however, a copy of the transaction is on the other server (the Secondary). Also, even if the Primary server fails before it has logged the transaction, the transaction is not lost. Thus, when relaxed durability is used with HSB, relaxed durability causes little reduction in safety. Relaxed durability can also improve the performance of the system, especially in situations where the server load consists of many small write transactions.

- *Adaptive durability*: Adaptive durability applies only to HotStandby Primary servers. Adaptive durability means that if the server is in Primary Active state (sending transactions to the Secondary), it uses relaxed durability. In any other state, strict durability is used. This gives you high performance (with little loss of safety) when HSB is active, yet maintains high safety if only one server is operating. Adaptive durability can be enabled effectively only when the 2-Safe replication is used (default).

Adaptive durability can significantly increase performance while it provides a high degree of data safety in failure situations. It can increase overall system throughput and it can reduce latency, that is, the time the user must wait before being told that the transaction has committed.

The durability level can be set as a server default with the **Logging.DurabilityLevel** parameter, or per session or transaction with the SET [TRANSACTION] DURABILITY statements.

Regarding replication protocols, `STRICT` corresponds to `2-safe` and `RELAXED` to `1-safe`.

1-Safe replication

With 1-safe replication, the commit statement is acknowledged immediately after the commit processing is completed at the Primary. The committed transaction is transmitted to the secondary asynchronously, after the control has been returned to the application.

The delay in transmitting the transaction can range from few milliseconds to a few hundred milliseconds. 1-safe replication offers significant performance gains because the latencies are reduced dramatically at Primary. The downside of 1-safe is that, in the case of a failure, a few transactions can be lost in a failover.

You can set the 1-safe replication for the server with the parameter **HotStandby.SafenessLevel=1safe**. Possible values of the **HotStandby.SafenessLevel** parameter are `1safe`, `2safe`, and `auto`; default is `2safe`.

You can also control the safeness level dynamically with the SET commands:

```
SET SAFENESS {1SAFE| 2SAFE| DEFAULT}
```

SET SAFENESS sets the safeness level for the current session, until it is changed.

```
SET TRANSACTION SAFENESS {1SAFE| 2SAFE| DEFAULT}
```

SET TRANSACTION SAFENESS sets the safeness level for the current transaction. After commit, the safeness level returns to the value set for the session, or the startup value, or the system default (which is 2-safe).

The option `DEFAULT` denotes the current setting for the session.

If the **HotStandby.SafenessLevel** parameter is set to `auto` (that is, "automatic"), you can control the safeness level with the programmatic durability controls (such as, SET DURABILITY RELAXED).

2-safe acknowledgement policy

When the 2-safe replication is enabled (default), the Primary server does not tell the client that the transaction has been successfully committed until the Primary receives acknowledgement that the Secondary has the transaction.

There are three different acknowledgement policies:

- 2-safe received: The Secondary server sends acknowledgement when it receives the data (default).
- 2-safe visible: The Secondary has updated its copy of the data, and the change is now visible. In other words, a client application that is connected to the Secondary server would be able to see the update.
- 2-safe durable: The Secondary server acknowledges when it has made the data durable, that is, when it has committed the data and written the data to the disk.

Choosing the 2-safe acknowledgement policy

2-safe received is faster. 2-safe durable is safer. Because the acknowledgement policies apply only when the Primary and Secondary server are both active (that is, both are applying the transactions), even 2-safe received is considered safe. You risk losing transactions only if both servers fail practically simultaneously (within a second of each other).

Using 2-safe received reduces latency (the amount of time between the start of the commit and the time that the user receives confirmation of the commit). The 2-safe received policy has little impact on overall throughput.

1.1.6 Load balancing of read-only workloads

In addition to the built-in transparent failover functionality, the Transparent Connectivity functionality in the solidDB JDBC and ODBC drivers provide support for load balancing of read-only workloads.

Load balancing is based on the fact that there are two synchronized (HotStandby) databases that running at the same time. All read queries provide the same result regardless of whether it is executed in the Primary or the Secondary database.

When the load balancing is activated, the JDBC or ODBC driver uses two physical connections, one to each database, and allocates the query load to the workload connection. The workload connection is selected based on query type (such as read or write), and the then-current load in the database servers.

The load balancing implementation is based on the following principles:

- Read-only queries can be executed in either database, if the isolation level is READ COMMITTED.
- Read queries that need high isolation level (repeatable read, select for update) are executed in the Primary database.
- Write queries are always executed in the Primary database.
- Read queries after any write operation within the same transaction are executed in the Primary database. Using the Primary database ensures that updated rows are visible for subsequent reads.
- Internal read/write level consistency of the databases is ensured so that after a write transaction is committed, the secondary database is not used for reading from the same connection until the secondary database is up-to-date for that write transaction. Using the same connection consistently ensures that if the 1-safe or 2-safe received HotStandby replication protocol is used, the next read transaction does not see committed data from the previous write transaction.

solidDB makes the selection of the workload connection automatically; the load balancing is automatic and transparent to the application. Especially read-centric applications can easily balance out the load between the two database servers, and use the full processor capacity of both servers.

The load balancing is activated with the solidDB ODBC driver by setting the `PREFERRED_ACCESS` connection attribute to value `READ_MOSTLY`; or with the solidDB JDBC driver by setting the property called `solid_preferred_access` to value `READ_MOSTLY`. Each application that connects to the database can choose to use the load balancing functionality, or choose to use the Primary database for all queries (default). You can also control load balancing dynamically on session or transaction level by altering the transaction isolation level of the servers.

Related concepts:

"Static load balancing configuration" on page 89

"Controlling load balancing dynamically" on page 90
When using load balancing (READ_MOSTLY or LOCAL_READ), you can change the assigned workload server from Secondary to Primary programmatically.

4.2.1, "Defining the Transparency Connectivity connection," on page 75
Transparent Connectivity is specified using non-standard ODBC connection string settings or JDBC connection properties.

1.1.7 HotStandby and SMA

The SMA server node can be made highly available with the solidDB HotStandby component.

In an SMA with HotStandby setup, there can be one or more SMA applications on each node. The application connection to the database can be configured as a regular SMA connection (SMA Basic Connectivity) or as a Transparent Connectivity SMA connection (SMA TC). With both connectivity types, the application on the Primary node uses an SMA connection to execute reads and writes locally, and the application on the Secondary uses an SMA connection to execute reads locally. Additionally, with SMA TC connection, write transactions from the application on the Secondary can be executed on the Primary server using a network connection. Furthermore, if the load balancing option is enabled with the SMA TC connection, the applications can operate in an active-active manner; on each node, the full functionality of database access is available.

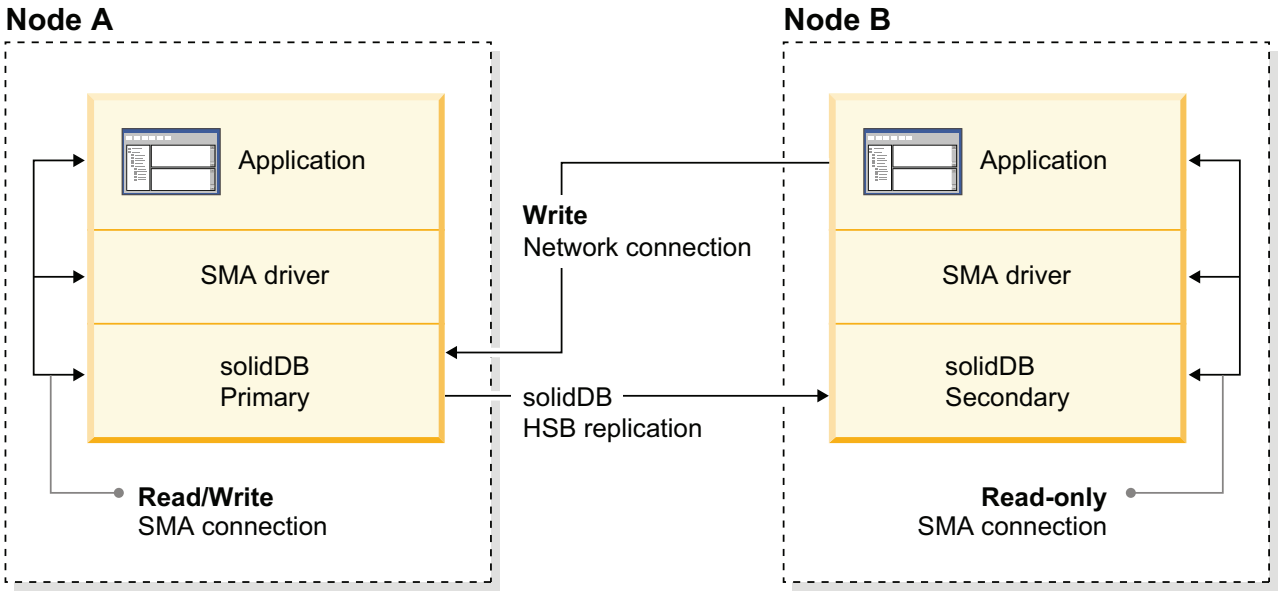


Figure 6. Architecture of SMA Transparent Connectivity with HotStandby

With SMA TC, the application on each node must be able to connect to the local server with a SMA connection and to the remote server with a network-based connection.

Failover and switchover handling

- The connection handle is maintained over switchovers and failovers for as long as one of the servers is in the PRIMARY ACTIVE, PRIMARY ALONE, or STANDALONE state.
- If the SMA server fails, the application might fail also. To ensure high availability in such a failure scenario, your system needs to include an application-level failover mechanism that moves the service offered by the application from the failed application instance to another one.

1.1.8 HotStandby and advanced replication

The solidDB HotStandby component can be used in combination with solidDB advanced replication. Advanced replication provides bidirectional, periodically occurring data synchronization that allows you to create a distributed system that contains *master* and *replica* servers. With HotStandby, you can make any of the database servers of the distributed system highly available.

Figure 7 on page 18 shows a simple distributed system that contains a master database and two replica databases. Each replica contains at least a subset of the data of the master database. Each of the database servers has been made fault-tolerant with HotStandby replication. Advanced replication occurs between the Primary servers of the database server hierarchy. In case of a problem with any of the Primary database servers, the failed node can do a HotStandby failover making the Secondary server of that node the new Primary. Advanced replication can now continue with the new Primary server.

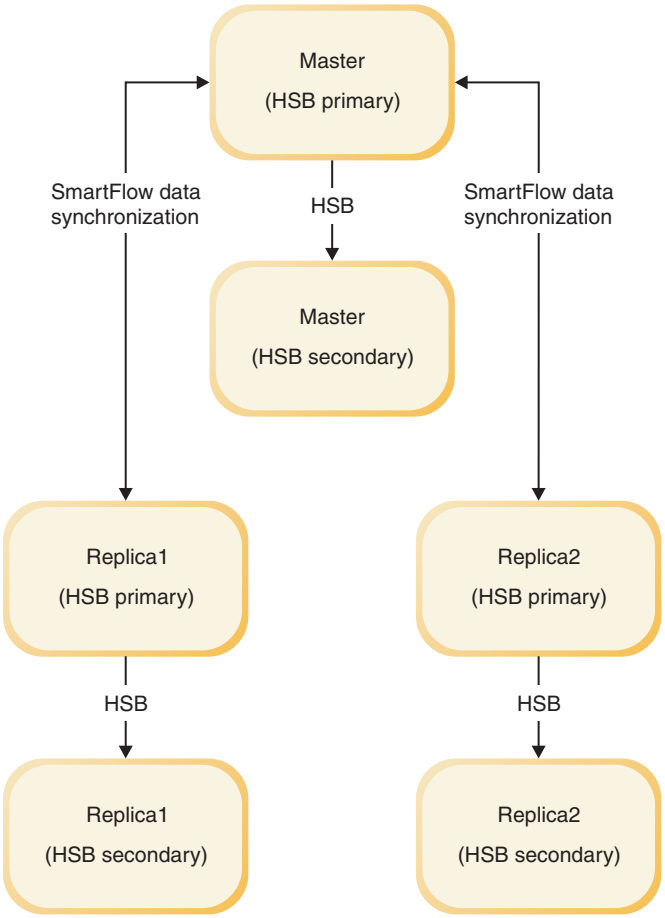


Figure 7. HotStandby with master and replica server scheme

1.2 Performance and HotStandby

There are several aspects to consider when tuning performance for a HotStandby system. With a HA (redundant) system, the effect of replicating the changed data reliably and consistently to another node plays an important role.

This section describes performance considerations in HotStandby setups. However, before you consider how to tune HotStandby setups, remember to optimize the SQL query and schema design. You can often correct poor performance by improving SQL queries, indexes, and so on. Actual throughput and response times depend on many factors, including (but not limited to) the speed of the network, the amount of other traffic on the network, the complexity of the SQL statements, and the number of SQL statements per transaction. The general environment factors, such as amount of memory and disk speed, also affect performance.

In a HotStandby setup, consider the following main performance elements:

- Latency or response times – How quickly is a single read or a write operation completed?
- Throughput – How much of the total query or transaction volume the two-node system can handle?

- Data safeness – Does the system ensure that every transaction is safely persisted on the same node (to disk) or to the next node (over the network)?
- Failover times – How quickly can the system continue to provide its service after a single-node failure, including the error detection time?
- Recovery times – How quickly and how automatically does the system recover to an HA state after the failure has been resolved?

In solidDB HotStandby setups, the following configuration and setup options can be used to optimize HotStandby performance:

- Adaptive durability is useful when you want to preserve transactions over single failures.
- 1-Safe replication protocol is useful when minor transaction loss over failures is acceptable.
- 2-Safe replication protocol and a suitable 2-Safe Acknowledgement Policy enforces maximum safety.
- Load balancing directs read-only transactions to the Secondary server.

  With the load balancing feature in HotStandby, clients can connect to the Secondary and perform read-only operations. In some situations, you can spread the load and improve overall system performance by having read-only clients connect to the Secondary and perform their reads there. Load balancing is useful for work such as report-generation or "data warehousing" queries, where you want to read several records and do not want to change any of them.
- Internal parallelism

  To ensure that your system takes advantage of parallelism, consider spreading your transactions across several connections rather than submitting all transactions through the same connection.

  When you use the HotStandby (HSB) component, every transaction that contains a write operation is executed twice, once on the Primary, and once on the Secondary. Subsequently in some situations, a single transaction might take twice as long with HSB as without HSB. However, this does not mean that overall throughput decreases 50%. The servers have a high degree of parallelism, and while the Secondary is working on one transaction, the Primary can work on another transaction.

  Note: The more queries you run in parallel, the more memory the server needs. Thus, adding connections and running queries in parallel does not always increase throughput, especially in systems that do not have a large amount of memory. You might have to experiment to find the optimal number of queries to run at a time.

In summary, unless you need the highest possible level of safety, you can increase performance by using the following configuration settings:

- Use adaptive logging by setting **`Logging.DurabilityLevel`** to 2.
- Use 2-safe received mode by setting **`HotStandby.2SafeAckPolicy`** to 1.

Even if you use the less safe settings (adaptive durability and 2-safe received mode), you are still protected by HotStandby unless there are at least two failures, for example, when both servers go down nearly simultaneously. To protect against power failure, each server must be connected to an Uninterruptible Power Supply (UPS). Furthermore, as with any database system, important data should be backed up and archived at a separate site. HotStandby is not a substitute for backing up your data.

Tip: You can run the backup (**ADMIN COMMAND backup**) on either of the servers of the HSB pair. Often it is the secondary server that has more resources available for creating the backup.

Related concepts:

3.5, “Performance tuning,” on page 65

1.3 High Availability Controller (HAC)

High Availability Controller (HAC) is an automatic redundancy management program for solidDB HotStandby configurations. It maintains the availability of the database service by detecting failures, performing failovers to standby units, and restarting failed processes when necessary.

A failure can be caused by a hardware problem in a database node, a database process failure, or a broken HSB link. HAC monitors the HSB states of the servers, and in the case of a failure, ensures that the server that is not affected by the failure holds the Primary role and that the server is ready to accept the transaction load.

In other words, HAC plays the role of a watchdog program. In its implementation, the solidDB's event mechanism is used to monitor the server states. Every time the state of the HSB server changes, it sends an event to HAC, which then deduces potential needs for actions that are executed by using the HotStandby admin command API.

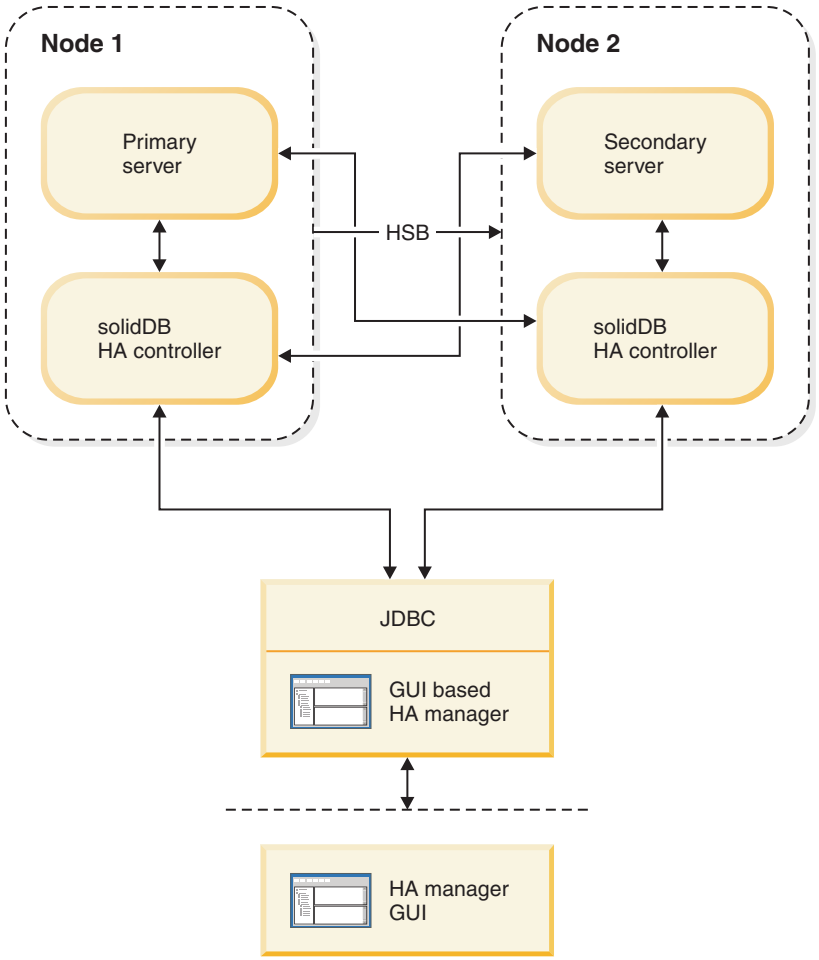


Figure 8. High Availability Controller architecture

HAC has two main purposes:

1. HAC can be used as a watchdog to automatically maintain the availability of the database service. In this mode, called the AUTOMATIC mode, HAC performs the following actions:
   - Starts, restarts, and terminates the database server processes (optional)
   - Monitors the state of the servers and the HSB link between them
   - Infers the necessary action to be performed
   - Performs the action
2. HAC can also play a role of a monitor for an HSB system by monitoring the state changes of the HSB servers (triggered by someone else) and reporting the status of the system. This is called the ADMINISTRATIVE mode. In this mode, HAC does not execute any HSB state transitions or otherwise modify the HSB system.

High Availability Controller is configured through the `solidhac.ini` configuration file. Before HAC is started, the configuration file must be in the HAC working directory. You can specify the working directory with the command-line option `-c`.

The solidDB package includes a HAC configuration file template which includes all available configuration parameters with comments, and examples. The `solidhac_template.ini` is available in the root of the `samples\hac\` directory.

A sample graphical user interface (GUI) component called High Availability Manager (HAM) is also available for HAC. It is included in the sample directory `samples\hac\`. The HA Manager is configured in the same as the HA Controller. The `samples\hac\` directory contains a sample configuration file (`HAManager.ini`) for the High Availability Manager.

1.3.1 Recognized failures

HAC monitors the health and status of the HotStandby servers. In failure situations, such as a database process failure or a computer node failure, HAC performs failovers and other necessary state transitions to maintain the best possible availability of the database service.

For all failures considered, it is assumed that they happen in a normal, fully operational state that is expressed by the PRIMARY ACTIVE and SECONDARY ACTIVE states of the two HSB servers. HAC takes care of single failures only. In other words, it is assumed that a failure cannot occur before the system has recovered from a previous failure. There is, however, a number of predefined multiple-failure scenarios that HAC can handle.

As far as single failures are concerned, HAC maintains an almost uninterruptible database service. If multiple failures occur, HAC attempts to avoid an erroneous system state (such as dual primaries).

The failures HAC can handle are:

- Single failures
  - The primary (ACTIVE) database server process fails
  - The secondary (ACTIVE) database server process fails
  - Primary node fails
  - Secondary node fails
  - If an External Reference Entity is used, HAC can also handle a HotStandby link failure, that is, a lost connection between the two HotStandby database processes. For more information about the External Reference Entity, see 1.3.3, “External Reference Entity (ERE),” on page 23.
  - Server is unresponsive to external clients
- Double failures
  - While recovering from a previous failure, HAC recognizes an error in the synchronization between the Primary and the Secondary database.
  - HAC also takes care several less common failures, such as a server process failure while servers are establishing HSB link after previous failure.

For detailed descriptions of the failure and recovery scenarios, see 5, “Failure handling with High Availability Controller (HAC),” on page 103.

1.3.2 Controlling database server processes

HAC can be configured to start database processes, and restart failed processes.

When a HAC instance loses connection with a local database server, it calls the start script that is specified in the `solidhac.ini` configuration file. The script is provided by the user. An example script is provided with the package.

Important: A HAC instance assumes that the server is running, and responsive at the moment the start script terminates. Since HAC does not handle failures that occur in server startup, the script must not exit unless the server accepts connections.

HAC restarts the database server whenever it fails or disappears for some other reason, except in the following cases:

- The database process is available in the process list of the operating system.
- The database server was shut down by using HA Manager.
- The database server was shut down by using the **`ADMIN COMMAND 'hsb shutdowndb'`** command.

1.3.3 External Reference Entity (ERE)

One of the failure situations you must prepare for is when the communication link between the database nodes fails, and both database servers might assume that the other one is down. This can lead into a dual primary situation (*split brain*), and you might lose transactions when databases are later synchronized. To avoid a wrong decision by HAC, you can use a network reference device, an External Reference Entity (ERE) to check the health of the network. For example, if a network adapter fails in one computer, HAC can detect this situation and is able to set the correct database node to continue as the Primary database, while the other one continues as Secondary.

If ERE is used, HAC checks the status of the physical link between the HotStandby node and the ERE device by pinging ERE. If the physical link to the nearest ERE is not operational, the local HAC sets the local server to the SECONDARY ALONE state. If the nearest physical link is operational, and no connection is available to the other server, the local HAC concludes that the local server is the one to continue offering the service, and sets it to PRIMARY ALONE. Consequently, The HotStandby node, which loses its connection to the opposite HotStandby node and to the nearest ERE, becomes the Secondary. In this way, the two Primaries (a split brain) situation is prevented in the case of network failure.

For more information about configuring HAC for ERE, see 3.3, “Configuring HA Controller and HA Manager,” on page 42.

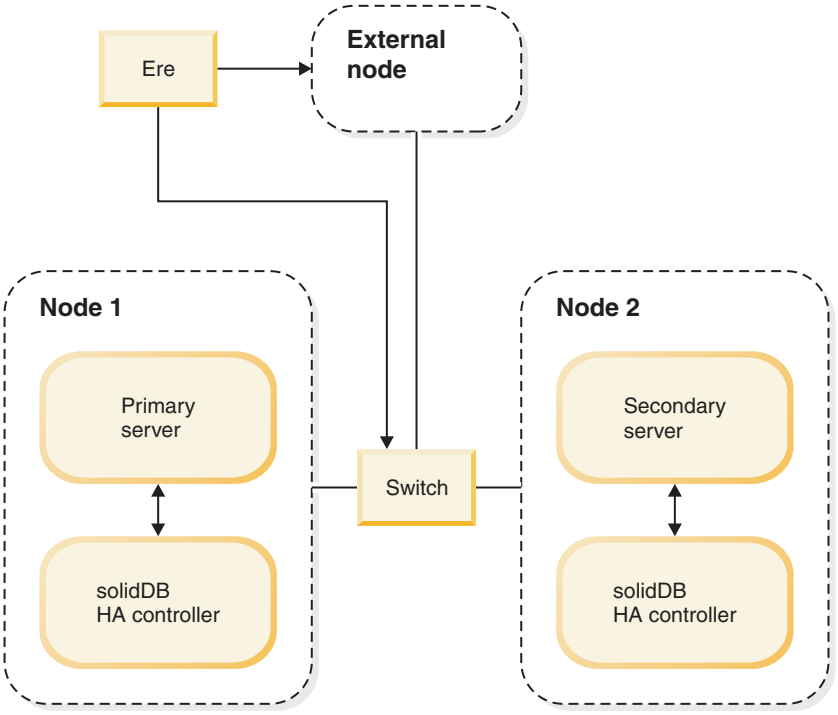


Figure 9. External Reference Entity components

The figure above depicts two possible locations of ERE:

- The cluster switch
- Any computer in the network outside the cluster. If a redundant network (that is, duplicate network controllers, cables and switches) is used in a cluster, define ERE outside the cluster.

Important: If the HotStandby link is considered unreliable (including all the cases where ERE is used), the following HotStandby server parameter must be set to its factory value:

```
HotStandby.AutoPrimaryAlone=no
```

ERE must use the same HSB link that the keepalive messages do.

1.3.4 Networking in HAC

To optimize your HotStandby setup, use a single logical network access (IP address) in each HSB server for all communications. The servers can use different port numbers for different purposes.

You can use multiple, or redundant, lower-level network components (network interface cards, cables and switches), that are transparent at the network interface API level.

When only one logical network access is used, HAC (with ERE) detects network breaks that affect both the HSB transaction replication and the database client applications. If separate interfaces are used, HAC cannot detect failures in database client communications because HAC monitors the health of the HSB link only.

However, regardless of the underlaying network access implementation, the ERE feature of HAC can be used if the network access is regarded unreliable for whatever reason.

1.3.5 HAC logging

HAC writes log records to the `hacmsg.out` file in the HAC working directory.

The log contains information about the following HAC operations:

- Warnings
- Unrecoverable errors
- Configuration-related information
- Initialization-related information
- All input events
- HotStandby state changes
- User commands, which cause changes in the system
- HAC state changes
- HAC mode changes (AUTOMATIC/ADMINISTRATIVE)
- Events, which cause state changes in the system

HAC log file has a maximum size of 64 megabytes. When the size limit exceeds, `hacmsg.out` is renamed to `hacmsg.bak`, and a new `hacmsg.out` is created. The files contain at most 128 MB of the most recent logs.

1.4 High Availability Manager (sample)

The solidDB package includes a sample program called High Availability Manager (HAM). HAM is a Java program with a GUI for displaying the state of the HotStandby servers and the state of the HACs. It also provides basic functionality for managing the HAC, for example, by switching the roles of HotStandby servers and to suspend and resume HAC.

The following figure shows the HAM interface.

Figure 10. High Availability Manager

The High Availability Manager is configured through the `HAManager.ini` configuration file.

You can perform the following actions by using the High Availability Manager:

- Perform a switchover between the HotStandby servers
- Switch the HAC mode between AUTOMATIC and ADMINISTRATIVE
  - In the ADMINISTRATIVE mode, HAC monitors only the HSB cluster, and the user can perform administrative tasks on HSB servers
  - In the AUTOMATIC mode, HAC acts as a watchdog, handles failures, and maintains the availability of the database service.
- Shut down a HotStandby database server process
- Start a HotStandby database server process
- Suspend a High Availability Controller
- Resume a High Availability Controller

Note:

Shutting down and starting the database server process is possible only if the **`HAController.EnableDBProcessControl`** parameter is set to `Yes` in the `solidhac.ini` configuration file.

2 Getting started with HotStandby

This section provides step-by-step instructions for setting up two solidDB HotStandby servers (a Primary server and a Secondary server).

The getting started steps assume that you have already installed solidDB on one or two computers.

You can set up an evaluation configuration for HotStandby using one computer; you can run two instances of solidDB on one computer and set up one of the instances as a Primary server and the other one as Secondary.

2.1 HotStandby quick start procedure

With the quick start procedure, you can reach a state where your HSB system is ready to serve applications.

About this task

You can use the quick start procedure to set up a pair of HotStandby servers on one or two computers. For evaluation purposes, you can use a single node setup. For testing and production environments, use two computers.

To set up and run an HSB server pair, you need two networked computers. To set up your HotStandby servers (without any other solidDB components), follow the procedure below.

Tip: The quick start procedure does not assume use of any other components than HSB servers. For example, HAC is not needed. A similar step-by-step procedure for HAC is described in 2.2.1, "Starting and stopping HA Controller," on page 31. There is also a sample watchdog application included in the solidDB server package. To use the sample watchdog, you need to provide configuration settings for it.

Procedure

1. Install the solidDB server on one or two computers.
   - If you are using a single computer, create two working directories and copy the license file to each directory. An evaluation license file is available in the root directory of the solidDB installation directory.

     For example, if you have installed solidDB server into the `C:\solid` directory, create two directories named `node1` and `node2`:

     ```
     mkdir C:\solid\node1
     mkdir C:\solid\node2
     ```
   - If you are using two computers, create a working directory on each computer and copy the license file to the directories.

     For example, if you have installed solidDB server into the `C:\solid` directories on both computers, create a directory named `node1` on one computer and a directory named `node2` on the other:

     Node 1

     ```
     mkdir C:\solid\node1
     ```

Node 2

```
mkdir C:\solid\node2
```

2. Configure the Primary and Secondary nodes. Create `solid.ini` configuration files in the working directories.

   At minimum, HotStandby requires that you configure the **HotStandby.HSBEnabled** and **HotStandby.Connect** parameters in the `solid.ini` configuration file on both the Primary (Node 1) and Secondary (Node 2) servers:

   - **HotStandby.HSBEnabled**=`Yes`
   - **HotStandby.Connect**=*connect string for the opposite HSB server*

   Additionally, the **Com.Listen** parameter must be set.

Table 5. Example: `solid.ini` configuration files in single-computer and two-computer setups

	Single computer setup	Two-computer setup

Node1	`C:\solid\node1\solid.ini`
	`[Com]`
	`Listen=tcp 2315`
	
	`[HotStandby]`
	`HsbEnabled=yes`
	
	`Connect=tcp 2325`
	
	The server instance Node 1 connects to the Node 2 instance using the connect string `tcp 2325`. Because the servers are on the same computer, you do not need to define a node name or IP address for Node 2.
		`C:\solid\node1\solid.ini`
		`[Com]`
		`Listen=tcp 2315`
		
		`[HotStandby]`
		`HsbEnabled=yes`
		
		`Connect=tcp node2 2325`
		
		The server on Node 1 connects to the server on Node 2 using the connect string `tcp node2 2325`. Instead of using a node name such as `node2`, you can use the IP address of the Node 2 computer, for example, `tcp 10.0.0.2 2325`.

Node2	`C:\solid\node2\solid.ini`
	`[Com]`
	`Listen=tcp 2325`
	
	`[HotStandby]`
	`HsbEnabled=yes`
	`Connect=tcp 2315`
	
	The server instance Node 2 connects to the Node 1 instance using the connect string `tcp 2315`. Because the servers are on the same computer, you do not need to define a node name or IP address for Node 1.
		`C:\solid\node2\solid.ini`
		`[Com]`
		`Listen=tcp 2325`
		
		`[HotStandby]`
		`HsbEnabled=yes`
		`Connect=tcp node1 2315`
		
		The server on Node 2 connects to the server on Node 1 using the connect string `tcp node1 2315`. Instead of using a node name such as `node1`, you can use the IP address of the Node 1 computer, for example, `tcp 10.0.0.1 2315`.

3. Start both HSB servers the way you would start any solidDB server.

   The servers read the HotStandby configuration information from their own `solid.ini` files. The state of both servers after startup is SECONDARY ALONE.

4. Choose the server that will become the Primary, and switch the state of the chosen server to PRIMARY ALONE by issuing the following command:

   ```
   ADMIN COMMAND 'hsb set primary alone';
   ```

   Tip: In the HotStandby ADMIN COMMANDs, you can use the abbreviation **hsb** in place of **hotstandby**.

5. Connect the Primary to the Secondary by issuing the following command in either server:

   ```
   ADMIN COMMAND 'hsb connect'
   ```

   To verify that the connection was successful, issue the following command:

   ```
   ADMIN COMMAND 'hsb state'
   ```

The Primary server responds that its state is PRIMARY ACTIVE. However, if the state of the Primary is something else than expected (for example, PRIMARY ALONE), the status of **hsb connect** can be checked by issuing the command:

```
ADMIN COMMAND 'hsb status connect'
```

If the result is ACTIVE, then connect process is still active. If the result is BROKEN, the databases of the servers must be synchronized before you connect them.

6. Synchronizing the databases by issuing the following command in the Primary:

   ```
   ADMIN COMMAND 'hsb netcopy'
   ```

   Check the status of database copy process by issuing the command:

   ```
   ADMIN COMMAND 'hsb status copy'
   ```

7. As soon as the result of **hsb status copy** is SUCCESS, the databases are synchronized, and the servers can be connected. Reissue the following command in either server:

   ```
   ADMIN COMMAND 'hsb connect'
   ```

   Verify the success by issuing the command `ADMIN COMMAND 'hsb state'`.

8. Start using applications.

Related concepts:

3.2.1, “Defining primary and secondary node HotStandby configuration,” on page 37
At minimum, to configure HotStandby, you must set the **HotStandby.HSBEnabled** parameter to `yes` on both nodes, and define the connection settings between the two nodes.

3.2, “Configuring HotStandby,” on page 37
HotStandby is configured by using the `solid.ini` configuration files at both the Primary and Secondary nodes. The `[HotStandby]` section contains all the HotStandby-specific configuration parameters. Other sections and parameters, such as the **Com.Listen** parameter, must be set also.

2.2 HotStandby with HAC quick start procedure

This section describes a quick start procedure for HotStandby with solidDB High Availability Controller (HAC).

About this task

The procedure is similar to 2.1, “HotStandby quick start procedure,” on page 27. However, instead of just setting up two HSB servers, this procedure guides you to set up a highly available HSB system, where availability is enforce by the High Availability Controller (HAC). Following the procedure below, you will get a failure tolerant HSB system ready to serve applications.

To set up and run a high availability HSB system, you need two networked computers. One instance of the HAC is set up on each HSB server.

Procedure

1. Follow the 2.1, “HotStandby quick start procedure,” on page 27 to configure the Primary and Secondary servers.

   As a result, one of the servers is in the PRIMARY ACTIVE state and the other in the SECONDARY ACTIVE state.

2. Configure the HACs on the Primary and Secondary servers. HAC reads its configuration from the `solidhac.ini` file in the working directory. The following mandatory configuration parameters are needed:

   [HAController] section

   - **Listen**=*<listen address, 'tcp' and chosen port #>*
   - **Username**
   - **Password**
   - **DBUsername**
   - **DBPassword**

   [LocalDB] section

   - **Connect**=*<connect address, protocol, ip/hostname, port #>*
   - **StartScript** (mandatory if **EnableDBProcessControl**=Yes, which is the default value)

   [RemoteDB] section

   - **Connect**=*<connect address, protocol, ip/hostname, port #>*

3. Start HAC on both nodes as instructed in 2.2.1, "Starting and stopping HA Controller," on page 31.

   HAC automatically finds out, which server becomes the new Primary according to the previous roles, and log positions. The mechanism is described in more detail in 3.4.9, "Choosing which server to make primary," on page 63. In some special situations, for example when started with empty databases, both servers are equally good candidates for the new Primary. In these situations HAC chooses its local server if in the **PreferredPrimary** parameter in the [LocalDB] section in `solidhac.ini` is set to Yes.

4. After the second step, there should be one server in the PRIMARY ACTIVE state and the other in the SECONDARY ACTIVE state. You can switch the roles, by issuing the following command on the Secondary server:

   ```
   ADMIN COMMAND 'hsb switch primary'
   ```

   or on the Primary server:

   ```
   ADMIN COMMAND 'hsb switch secondary'
   ```

   After this point, HAC switches the roles of the servers when necessary. At least one server executes read and write transactions.

5. Start using applications.

Related concepts:

3.3, "Configuring HA Controller and HA Manager," on page 42
High Availability Controller (HAC) is deployed on each of the HotStandby server nodes; it is configured through the `solidhac.ini` configuration file. High Availability Manager (HAM) is configured through the `HAManager.ini` configuration file.

Related reference:

A.3, "High Availability Controller (HAC) parameters," on page 123
This section describes the High Availability Controller (HAC) configuration parameters in the `solidhac.ini` configuration file.

A.5.2, "The solidhac.ini configuration file," on page 129
A sample excerpt of the High Availability Controller (HAC) configuration file (`solidhac.ini`).

2.2.1 Starting and stopping HA Controller

Before you can start HAC, you must have a `solidhac.ini` configuration file in the HAC working directory. See 2.2, "HotStandby with HAC quick start procedure," on page 29 for a short description of configuring HAC. For more information about configuring HAC, see 3.3, "Configuring HA Controller and HA Manager," on page 42. For a list of parameters and detailed parameter descriptions, see A.3, "High Availability Controller (HAC) parameters," on page 123 and A.5.2, "The solidhac.ini configuration file," on page 129.

Note: Depending on the platform you are using, the HAC binary is named either `solidhac`, or `solidhac.exe`. In the examples, `solidhac.exe` is used for clarity. Similarly, the name of the sample solidDB start script is either `start_solid.sh`, or `start_solid.bat`. In the examples, `start_solid.bat` is used.

Starting HAC

The syntax for starting HAC is:

```
solidhac.exe [-c working_directory | -?]
```

The **?** argument, or any other argument except **-c** prints the usage message.

When HAC starts, it starts to listen to the port specified **HAController.Listen** parameter in the `solidhac.ini` configuration file. That port is used, for example, for transporting commands between HAC and the HAManager, and for issuing HAC-specific ADMIN COMMANDs.

Stopping HAC

To issue the command, you must first connect to HAC by specifying the port that is defined with the **HAController.Listen** parameter in the `solidhac.ini` configuration file. For example, you can use **solsql** or the ODBC interface to connect to HAC.

You can stop (terminate) HAC by executing the following command:

```
ADMIN COMMAND 'hacontroller shutdown'
```

Example: Starting HAC

If `solidhac.exe` is located in `c:\solid\hac`, which is also the current directory, and you use `c:\solid\run\server1` as the working directory for HAC, you start HA Controller with the following command:

```
solidhac.exe -c c:\solid\run\server1
```

or

```
solidhac.exe -c ..\run\server1
```

Related reference:

C.2, "High Availability Controller commands (ADMIN COMMAND)," on page 159

2.3 Summary of startup sequences

The following figure and table present side-by-side sequences for an installation with HAC and for an installation without HAC. In the figure, the installation sequence for HAC is on the left and the installation sequence without HAC is on the right. The table after the figure explains each numbered step in the figure for both installation types.

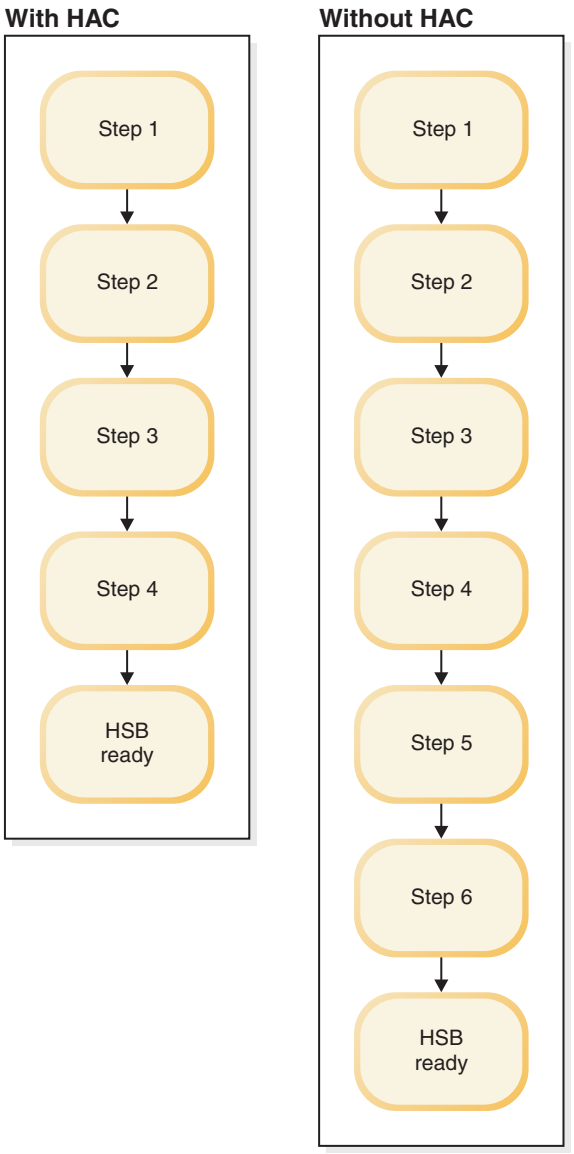


Figure 11. Summary of startup sequences

Table 6. Installation sequence steps

Installation with HAC	Installation without HAC
Step 1. Configure HSB servers	

Table 6. Installation sequence steps (continued)

Installation with HAC	Installation without HAC
Step 2. Configure HA Controller Step 3. Start HA Controllers in both nodes Step 4. Switch HSB roles, if necessary	Step 2. Start HSB servers in both nodes Step 3. Switch the state of the database server to `PRIMARY ALONE` Step 4. Connect Primary to the Secondary Step 5. If connect failed, start db copy from Primary to Secondary Step 6. After **`netcopy`**, reconnect
HSB is ready. You can start using applications.	

2.4 HotStandby samples

The solidDB server package includes samples for running demonstrations of the HotStandby component with HAC or the sample watchdog implementations.

HotStandby with HAC

A sample for HotStandby with High Availability Controller (HAC) is available in the `samples\hac` directory. Detailed information about the sample is available in the file: `samples\hac\readme.txt`

HotStandby with watchdog sample

A sample for HotStandby with watchdog is available in the `samples\hsb` directory. Detailed information about the sample is available in the file: `samples\hsb\readme.txt`

3 Administering and configuring HotStandby

This section describes how to maintain your HotStandby installation, including HSB servers and HAC instances, using the HotStandby related parameters and ADMIN COMMANDs.

This description supplements the information in the section.

3.1 Basics of HotStandby administration

The HotStandby related components are administered with configuration parameters and ADMIN COMMANDs.

Configuration parameters

Parameters are grouped according to section categories in the `solid.ini` configuration file. When you are using HotStandby, you mainly use the `[HotStandby]` section of the `solid.ini` configuration file. Other sections can be used to configure the overall behavior of the solidDB server.

The High Availability Controller (HAC) is configured by using the `solidhac.ini` configuration file. The parameters in the `solidhac.ini` configuration file are also grouped according to different section categories.

The High Availability Manager (HAM) is configured by using the `HAManager.ini` configuration file. The parameters in the `HAManager.ini` identify the HA Controller instances so that the High Availability Manager has access to them.

You can change configuration parameters in the following ways:

- Edit the configuration files `solid.ini`, `solidhac.ini` and `HAManager.ini` manually.

  Note: The server reads the configuration files during startup only, and therefore any changes to any configuration file will not take effect until the next time that the corresponding program is started.
- Use the following ADMIN COMMAND syntax to change the settings of a running solidDB server dynamically:

  ```
  ADMIN COMMAND 'parameter section_name.param_name=value';
  ```

  For example:

  ```
  ADMIN COMMAND 'parameter hotstandby.2SafeAckPolicy=2';
  ADMIN COMMAND 'parameter com.listen="tcp sf_server 1315"';
  ```

Note: All configuration parameters cannot be changed dynamically.

HotStandby ADMIN COMMANDs (HotStandby API)

The HSB API is provided as a syntax extension to solidDB SQL, in the form of ADMIN COMMAND:

```
ADMIN COMMAND hotstandby hsb-command options
```

or

```
ADMIN COMMAND hsb hsb-command options
```

The HSB commands can be issued via any SQL-capable interactive tool (like **solsql**), or programmatically, through ODBC or JDBC.

HotStandby and access rights

There are no specific access rights for HotStandby, normal administrator access rights suffice. To execute the HotStandby administrative commands, SYS_ADMIN_ROLE or SYS_CONSOLE_ROLE access rights are required.

HotStandby and solidDB tools

The solidDB data management tools can be used with HotStandby servers.

Console tools for issuing ADMIN COMMANDs

You can issue HotStandby-specific administrative commands (**ADMIN COMMAND 'hotstandby <option>'**) with solidDB SQL Editor (**solsql**), and solidDB Remote Control (**solcon**).

When you are using **solsql**, the command name must be given with single quotation marks and a semicolon at the end of the command. For example:

```
ADMIN COMMAND 'hotstandby status connect';
```

When you are using **solcon**, the command name must be given without the ADMIN COMMAND prefix, quotation marks, and without the semicolon at the end of the command. For example: :

```
 hotstandby status connect
```

Import and export tools

The import and export tools, solidDB Speed Loader (**solloado** or **solload**), solidDB Export (**solexp**), and solidDB Data Dictionary (**soldd**) can also be used with HotStandby.

High Availability Manager (HAM) sample

The solidDB package also contains a sample tool, High Availability Manager (HAM) that can be used to monitor solidDB HotStandby states, and to control HSB servers and High Availability Controllers (HAC). HAM can be used only with the HA Controller.

3.1.1 Querying HotStandby configuration parameters

Standard parameter manipulation commands can be used to query the values and properties of the HotStandby parameters.

The commands are:

```
ADMIN COMMAND '[describe] parameter[section_name[.parameter_name]]';
```

For example:

```
ADMIN COMMAND 'parameter logging.durabilitylevel';
RC TEXT
-- ----
0 Logging DurabilityLevel 3 3 2

ADMIN COMMAND 'parameter hotstandby.MaxLogSize';
RC TEXT
-- ----
0 HotStandby MaxLogSize 10000000 0 0
```

The result set shows the following 3 values:

- *Current value* - set dynamically or inherited from the default or factory value.
- *Default value* - read originally from the `solid.ini` file or inherited from the factory value.
- *Factory value* - preset in the product.

3.1.2 Modifying HotStandby configuration parameters

Normally, you change the value of a parameter by changing the value in the `solid.ini` configuration file and then restarting the server. However, most of the HotStandby parameters can also be changed with an ADMIN COMMAND.

About this task

The syntax for modifying parameters is:

```
ADMIN COMMAND 'parameter section_name.parameter_name=value [temporary]';
```

Unless the **temporary** option is used, all the changes made to the parameters will be saved in the `solid.ini` file at the next checkpoint. The saving can be expedited also with the command:

```
ADMIN COMMAND 'save parameters [file_name]';
```

By default, the command rewrites the default `solid.ini` file. By using the *file_name* option, the output can be directed to a different location.

3.2 Configuring HotStandby

HotStandby is configured by using the `solid.ini` configuration files at both the Primary and Secondary nodes. The `[HotStandby]` section contains all the HotStandby-specific configuration parameters. Other sections and parameters, such as the **Com.Listen** parameter, must be set also.

3.2.1 Defining primary and secondary node HotStandby configuration

At minimum, to configure HotStandby, you must set the **HotStandby.HSBEnabled** parameter to `yes` on both nodes, and define the connection settings between the two nodes.

The solid.ini configuration files on each node must have the following minimum configuration information in the `[HotStandby]` sections of the solid.ini files:

- The **HotStandby.HSBEnabled** parameter must be set to `yes`.

  If the **HotStandby.HSBEnabled** is not set to `yes`, the server starts as a non-HotStandby server and HotStandby replication is not used.
- The **HotStandby.Connect** parameter must be set. This parameter defines the network name that is used to connect to the other server (either Primary or Secondary). If you do not set this parameter in the `solid.ini` file, the server can run only in the states that do not require a connection, for example, PRIMARY ALONE, SECONDARY ALONE, and STANDALONE. When the server is running, you can set or change this parameter by using the ADMIN COMMAND 'parameter' command.
- The **HotStandby.Connect** string for each server must match the **Com.Listen** string of the other server. For example, if the server on Node 1 is listening at `tcp 2315`,

the **HotStandby.Connect** parameter on the server at Node 2 must be set to `tcp <node_name | ip_address> 2315`. The *node_name* or *ip_address* of the server on Node 1 must be specified also.

Example: Node 1 minimum configuration

```
[Com]
Listen=tcp 2315

[HotStandby]
HsbEnabled=yes

Connect=tcp node2 2325
;The server on Node 1 connects to the server on Node 2
;using the connect string 'tcp node2 2325':
;The host name 'node2' can be replaced by IP address.
```

Example: Node 2 minimum configuration

```
[Com]
Listen=tcp 2325

[HotStandby]
HsbEnabled=yes

Connect=tcp node1 2315
;The server on Node 2 connects to the server on Node 1
;using the connect string 'tcp node1 2315'.
;The host name 'node1' can be replaced by IP address.
```

3.2.2 Setting HotStandby server wait time to help detect broken or unavailable connections

A HotStandby server uses timeout parameters to control how long it waits before it concludes that an existing connection is broken or a new connection cannot be established.

About this task

The timeout parameters are:

- **HotStandby.PingTimeout**
- **HotStandby.PingInterval**
- **HotStandby.ConnectTimeout**

If a HotStandby server that is in the PRIMARY ACTIVE or SECONDARY ACTIVE state tries to contact the other server and receives no reply within a specified amount of time, it changes to PRIMARY UNCERTAIN, PRIMARY ALONE, or SECONDARY ALONE.

Procedure

To control how long the server waits, you can:

- Set the **PingTimeout** parameter to specify the amount of time that the server must wait before it changes to the PRIMARY UNCERTAIN state.
- Set the **PingInterval** parameter to specify the interval between the "ping" messages the server sends to indicate that it is working correctly.
- Set the **ConnectTimeout** parameter to specify the amount of time that the server must wait when it tries to establish a new connection to the other server (for example, in an **ADMIN COMMAND 'hotstandby connect'** operation).

PingTimeout and PingInterval parameters [HotStandby]

The optional **PingTimeout** and **PingInterval** parameters in the [HotStandby] section control the ping operation.

A *ping* operation is essentially an "I'm alive" message that is sent by one database server to another. The ping system in the HotStandby setup is a passive heartbeat system. When enabled, both the Primary and Secondary servers send ping messages to each other at regular intervals. For more information, see “Heartbeat” on page 3.

- **HotStandby.PingTimeout** specifies how long a server should wait before concluding that the other server is down or inaccessible. Default is 4000 (4 sec.)
- **HotStandby.PingInterval** specifies the interval, in milliseconds, between two pings. Default is 1000 (1 sec.)

For example, if the **PingInterval** is 10 seconds, then the servers will send ping messages to each other after every 10 seconds. If **PingTimeout** is 20 seconds and one server (S1) does not hear from the other (S2) within 20 seconds, then S1 concludes that S2 is down or inaccessible. Server S1 then switches to another state, for example, from PRIMARY ACTIVE to PRIMARY UNCERTAIN.

If the values of the parameters are different, the precedence take values set in Primary during execution of the **hsb connect** command. The values do not change during switchovers. However, they can be changed dynamically with the **ADMIN COMMAND 'parameter'** command.

If **PingTimeout** is set to zero, pinging is disabled.

Ping requires little overhead and a solidDB server is set up to respond quickly to missing ping messages. You can set the **PingInterval** value to a fairly short interval, such as a second, or even less.

If it is important that you detect a failover quickly, set the **PingTimeout** value to a relatively short time. However, shorter values also mean a higher chance for false alarms. If your network has a lot of traffic and thus causes delays before a ping message is received, you might need to set the **PingTimeout** to a large value to avoid false alarms.

Note: Some networking software also has a ping operation, but the **HotStandby.PingTimeout** and **HotStandby.PingInterval** parameters apply only to the solidDB server pings, not general network pings.

ConnectTimeout parameter [HotStandby]

In some network implementations, a connect operation might not respond for an indefinite period of time. One possible reason is that the remote server is a known node but unavailable during the connect attempt. By specifying a connect timeout value, you can set the maximum time in seconds that a HotStandby connect operation waits for a connection to a remote server.

You set the connect timeout value with the **HotStandby.ConnectTimeout** parameter. The unit is milliseconds. The default is 0, which means no timeout. You can set it to a different value, for example:

```
[HotStandby]
 ; Set ConnectTimeout to 20 seconds (20000 milliseconds).
ConnectTimeout=20000
```

The **HotStandby.ConnectTimeout** parameter is used with the following administration commands:

- **hotstandby connect**
- **hotstandby switch primary**
- **hotstandby switch secondary**

3.2.3 Defining transaction durability level

The transaction durability level is set with the **Logging.DurabilityLevel** parameter. The parameter has three different values: relaxed (1), adaptive (2), and strict (3) durability.

Adaptive durability is used only with HotStandby. Adaptive durability means:

- If Primary and Secondary are connected and operating normally (in PRIMARY ACTIVE and SECONDARY ACTIVE states), the server uses relaxed durability.
- In all other situations (for example, PRIMARY ALONE, STANDALONE), the server uses strict durability.

For an explanation of the differences between strict and relaxed durability, or for more information about the **Logging.DurabilityLevel** parameter, see *IBM solidDB Administrator Guide*.

3.2.4 Defining name and location for HotStandby database copy operation

The optional **HotStandby.CopyDirectory** parameter defines the name and location of the directory that the HotStandby **copy** operation copies to.

The HotStandby copy operation is specified with the command:

```
ADMIN COMMAND 'hotstandby copy [directory_name]';
```

The **HotStandby.CopyDirectory** parameter has no default value, so if the directory is not specified in the `solid.ini` file, it must be provided in the copy command. If you provide a relative path for the **HotStandby.CopyDirectory** parameter, the path is relative to the working directory of the Primary server.

The **HotStandby.CopyDirectory** parameter is not needed if you perform the HotStandby database copy operations by using the **ADMIN COMMAND 'hotstandby netcopy'** command.

Important: The **copy** command is deprecated. Instead, use the **netcopy** command.

3.2.5 Defining primary server behavior during a secondary failure

You can use the **HotStandby.AutoPrimaryAlone** parameter to control whether the Primary server automatically switches to PRIMARY ALONE state or stays in PRIMARY UNCERTAIN state after it has lost contact with the Secondary server.

If **AutoPrimaryAlone** is set to `Yes`, when Primary loses contact with Secondary, Primary automatically switches automatically to the PRIMARY ALONE state . The PRIMARY ALONE state means that the Primary continues to accep transactions. If **AutoPrimaryAlone** is set to `No`, when Primary loses contact with Secondary, Primary switches automatically to the PRIMARY UNCERTAIN state.

By default, **AutoPrimaryAlone** is set to No.

```
[HotStandby]
AutoPrimaryAlone =  No
```

The PRIMARY UNCERTAIN state prevents Primary from accepting new transactions or committing the currently active ones. Primary does not switch to PRIMARY ALONE state until HAC, the Watchdog, or System Administrator tells it to do so.

If **AutoPrimaryAlone** is set to No, the server can be set to the PRIMARY ALONE state by executing the **ADMIN COMMAND 'hotstandby set primary alone'** command. The **ADMIN COMMAND 'hotstandby set primary alone'** command does not change the value of **AutoPrimaryAlone** in the configuration file.

If you change the default to Yes, the Primary server state changes from PRIMARY ACTIVE to PRIMARY ALONE rather than to PRIMARY UNCERTAIN.

3.2.6 Ensuring that Primary and Secondary parameter values are coordinated

Some of the HotStandby or other parameters must be the same on the Primary and Secondary servers, some must be different.

Certain parameters should be the same on both the . To ensure that after a failover, the original Secondary becomes the new Primary and behaves the same as the old Primary, some of the HotStandby parameters must be set to same values on the Primary and the Secondary. However, the usage of the same values is not an absolute requirement nor enforced by the system. If you use different values, the servers do not fail, but clients might see different behavior.

Some parameters that are not in the [HotStandby] section, but which are indirectly related, must also be the same on both the Primary and Secondary servers. For example, the **Logging.DurabilityLevel** parameter generally must be the same on the Primary and Secondary.

Certain parameters must be different on the Primary and Secondary servers, for example, to ensure that the servers can be uniquely identified and can talk to each other.

The following HotStandby parameters must be the same on both the Primary and Secondary:

- [HotStandby]
  - **2SafeAckPolicy**
  - **AutoPrimaryAlone**
  - **ConnectTimeout**
  - **HSBEnabled**
  - **PrimaryAlone** (deprecated, but must be the same if used)
- [IndexFile]
  - **FileSpec** must be compatible, which means that the number of **FileSpec** parameters must be the same and the sizes of the corresponding **FileSpec** parameters must match.
  - **BlockSize**
- [Logging]

  - **BlockSize**

The following parameters must be different:

- [HotStandby]
  - **Connect**

The following parameters can be the same or different, depending upon circumstances such as the disk drive configuration on the computer:

- [General]
  - **BackupDirectory**
- [HotStandby]
  - **CopyDirectory**

There are also some settings of "non-HSB" parameters that affect HSB performance. For example, the **Logging.DurabilityLevel** parameter has a setting that you can use to optimize performance with HotStandby. See 1.1.5, "Durability and logging," on page 13 and see the description of **Logging.DurabilityLevel** in *IBM solidDB Administrator Guide*.

3.2.7 Determining whether the Primary settings take precedence over the Secondary settings

Some parameters must be the same for both the Primary and Secondary servers. If you do not set the parameters to same values, you might expect that each server uses the value that is defined in solid.ini file of server. However, this is not necessarily the case.

Even for some parameters that control the behavior of the Secondary, such as **2SafeAckPolicy**, the value on the Primary is the value that determines the behavior. The principle is that all safeness and durability parameters are controlled at the Primary. For example, the Primary reads its value of **2SafeAckPolicy** and sends that value to the Secondary to use. The value stored in the Secondary's solid.ini file is used only if the Secondary becomes the Primary.

The values set at the Primary take precedence over the values set at the Secondary for the following parameters:

- **HotStandby.SafenessLevel**
- **HotStandby.2SafeAckPolicy**
- **Logging.DurabilityLevel**
- **HotStandby.NetcopyRpcTimeout**

When the command **hsb connect** is issued, the following Primary parameters take precedence:

- **HotStandby.PingTimeout**
- **HotStandby.PingInterval**

3.3 Configuring HA Controller and HA Manager

High Availability Controller (HAC) is deployed on each of the HotStandby server nodes; it is configured through the solidhac.ini configuration file. High Availability Manager (HAM) is configured through the HAManager.ini configuration file.

HA Controller

The HAC configuration file `solidhac.ini` must be located in the HAC working directory. Parameters in the `solidhac.ini` configuration file are grouped under the following sections:

- `HAController`
- `LocalDB`
- `RemoteDB`
- `ERE`

For descriptions of the parameters, see A.3, "High Availability Controller (HAC) parameters," on page 123

All the configuration parameters are shown also in the `solidhac.ini` example file in A.5.2, "The solidhac.ini configuration file," on page 129.

HA Manager

The HAM configuration file `HAManager.ini` must be located in the HAM working directory.

For descriptions of the parameters, see A.4, "High Availability Manager (HAM) configuration parameters," on page 128

All the configuration parameters are shown also in the `HAManager.ini` example file in A.5.3, "The HAManager.ini configuration file," on page 132.

3.4 Administering HotStandby with ADMIN COMMANDs (HotStandby API)

The HotStandby API (HSB API) is used for monitoring and controlling the server processes. For example, the solidDB High Availability Controller (HAC) uses the HSB API. Another example is the Watchdog sample program that is included in the solidDB server package.

The HSB API is provided as a syntax extension to solidDB SQL, in the form of ADMIN COMMAND:

```
ADMIN COMMAND hotstandby hsb-command options
```

or

```
ADMIN COMMAND hsb hsb-command options
```

The HSB commands can be issued via any SQL-capable interactive tool (like **`solsql`**), or programmatically, through ODBC or JDBC.

You can use the HSB commands to program your own application to manage high availability in solidDB, for example, to implement an integration to an external cluster management software.

3.4.1 Overview of administration tasks

The HotStandby administration includes tasks such as recovery and maintenance.

Table 7. Administration tasks

Topic	Description	Page
Performing HotStandby recovery and maintenance tasks	Describes HotStandby tasks in the case of a system failure (resulting from either a broken communication link or an inoperable HotStandby server). These tasks include: • Switching server states • Shutting off HotStandby operations • Synchronizing Primary and Secondary servers • Connecting HotStandby servers	3.4.2, "Performing HotStandby recovery and maintenance"
Copying a Primary database to a new Secondary over the network	Describes how to create a remote (network) copy of a database when the remote server is a new addition to the HotStandby configuration (that is, it is a new Secondary), or the remote server data becomes corrupted and must be replaced.	"Copying a primary database to a secondary over the network" on page 54
Checking HotStandby status	Describes how to check HotStandby status information for the Primary and Secondary servers.	3.4.7, "Checking HotStandby status," on page 59
Verifying HotStandby server states	Describes how to check the state (Primary, Secondary, or stand-alone) of a HotStandby server.	3.4.8, "Verifying HotStandby server states," on page 62
Changing a HotStandby server to a non-HotStandby server	Describes how to set a server that is configured for HotStandby to a normal, non-HotStandby server.	3.4.10, "Changing a HotStandby server to a non-HotStandby server," on page 64

3.4.2 Performing HotStandby recovery and maintenance

After a system failure or during server maintenance, you might have to perform HotStandby tasks such as, switching server states, shutting off HotStandby operations, synchronizing Primary and Secondary servers, and connecting HotStandby servers.

Procedure

1. Perform some or all of the following operations:
   a. Switch the server state.

      This includes setting the Primary server to PRIMARY ALONE state, which continues accumulating transactions in the transaction log so that they can be sent to the Secondary later, or shutting down HotStandby.

b. Synchronize the servers to be sure the Primary and Secondary databases are identical.
c. Connect the Primary server to the Secondary server if the communication link is broken for some reason.

2. The same steps can be taken with HAC and HA Manager as follows:
   a. Press the **Switch** button in HA Manager. If the server needs to be shut down, press the **Shutdown** button in HA Manager.
   b. Set HAC instances to the ADMINISTRATIVE mode by pressing the **Administrative** buttons in HA Manager.
   c. Set HAC instances to AUTOMATIC mode by pressing the **Automatic** buttons in HA Manager.

What to do next

For details on reconnecting applications to Secondary or Primary databases, see 4.3.1, "Reconnecting to primary servers from applications," on page 94.

Important:

If HAC is used, either use HA Manager to perform administrative steps, or set HAC instances to ADMINISTRATIVE mode before you start the administration task.

3.4.3 Switching server states

The HotStandby component requires that the server state is switched, when necessary, either automatically or manually by a user.

In production use, the server state is chosen by using automatic state switching, that is, by performing failovers. The automatic state switching can be the responsibility of, for example, the solidDB High Availability Controller (HAC).

Switchover means reversing the roles of the Primary and Secondary when they are running. Switchover can be used for various maintenance purposes.

Failover is the action by Secondary for taking up the role of the Primary when the Primary fails.

Performing switchovers

The roles of the servers can be reversed by issuing the following command at the Secondary: `ADMIN COMMAND 'hotstandby switch primary';` or, at the Primary:`ADMIN COMMAND 'hotstandby switch secondary';`

The **`switch`** commands can be used regardless of whether the two servers are connected. If the servers are connected, the states are reversed; the old Secondary becomes the new Primary, and the old Primary becomes the new Secondary. If the servers are not connected, the old Secondary becomes the new Primary, and the state of the other server is unchanged.

The diagram below shows what happens if you issue the command **`hsb switch secondary`** or **`hsb switch primary`** when the servers are connected. The command **`hsb switch primary`** can be issued only on a server that is in a SECONDARY state (for example, SECONDARY ACTIVE), while the command **`hsb switch secondary`** can only be used on a server that is in a PRIMARY state (for example, PRIMARY ACTIVE).

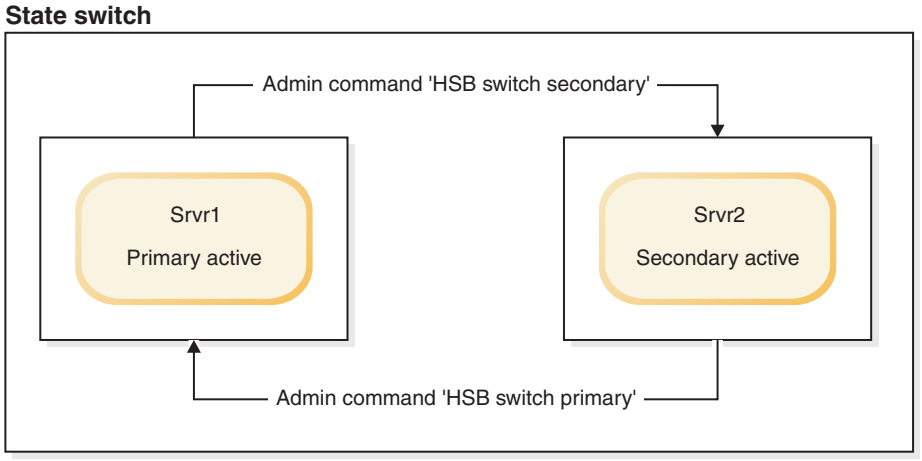


Figure 12. State switch

When you issue the command **hotstandby switch primary** to switch the Secondary server (Srvr2) to Primary, if the Secondary server (Srvr2) is not connected to the other server (Srvr1), an error is returned.

If the two servers are connected, they switch states. In other words, the old Primary (Srvr1) becomes the new Secondary and old Secondary (Srvr2) becomes the new Primary.

If the old Secondary (Srvr2) cannot connect to the other server (Srvr1), both servers switch to SECONDARY ALONE. Even if the **AutoPrimaryAlone** configuration parameter is set to `yes`, the new Primary switches to SECONDARY ALONE, not PRIMARY ALONE.

Switching Secondary to Primary:

You can switch a Secondary server to PRIMARY state by issuing the command: `ADMIN COMMAND 'hotstandby switch primary';`.

When the **hotstandby switch primary** command is issued, it starts a process to switch the state. If the switch process started successfully, the following message is displayed: `Started the process of switching the role to primary`. You can monitor the status of the switch by using the command **hotstandby status switch**.

If you issue a commit after the **hotstandby switch primary** command, the commit fails with the error `replicated transaction is aborted`.

All transactions are terminated during the switch. However, administrative commands (ADMIN COMMAND) are not transactional commands and cannot be rolled back.

Note: Administrative commands force the start of a new transaction if one is not already open. To avoid leaving an open transaction, or having a transaction start time be different from what you expected, issue COMMIT WORK after administrative commands.

If a configuration error causes both servers to have the state of PRIMARY (for example, both are PRIMARY ALONE), you can use the command **hotstandby switch secondary** to switch one of the servers back to a SECONDARY state.

- If the servers have the same data, normal operations on both servers are resumed.
- If the servers do not have the same data, the Primary server rejects the connect operation from the Secondary and issues the following message: `14525: HotStandby databases are not properly synchronized`.

  HotStandby replication is not started. In this case, a full copy of the Primary database is required at the Secondary server. First you must decide which database is correct.

  Also, if the `14525` error occurs, the database states do not change. Both servers remain in the same state they were in before the command was issued.

Related tasks:

"Displaying switch status information" on page 60
You can use the **ADMIN COMMAND 'hotstandby status switch'** command to verify whether a state switch occurred between two HotStandby servers.

Switching Primary to Secondary:

You can switch a Primary server to SECONDARY state by issuing the command **ADMIN COMMAND 'hotstandby switch secondary'**.

The **ADMIN COMMAND 'hotstandby switch secondary'** can be useful if two servers have switched states and you want to switch them back to their original states. For example, when the new Secondary comes back in service, you can switch its state back to Primary and switch the new Primary back to Secondary.

When the **hotstandby switch secondary** command is issue, it starts a process to switch the state. If the switch process started successfully, the following message is displayed: `Started the process of switching the role to secondary`. You can monitor the status of the switch by using the command **hotstandby status switch**.

When you execute the **hotstandby switch secondary** command, if the servers are not already connected to each other, the old Primary tries to connect to the old Secondary.

If the two servers are connected, they switch states. The old Primary becomes the new Secondary and the old Secondary becomes the new Primary.

Related tasks:

"Displaying switch status information" on page 60
You can use the **ADMIN COMMAND 'hotstandby status switch'** command to verify whether a state switch occurred between two HotStandby servers.

Performing failovers

A failover is performed by executing the command **ADMIN COMMAND 'hotstandby set primary alone'** at the Secondary.

After all the pending transactions received from the Primary have been processed, server gains the new state once. No transactions are lost, and the database state reflects the state at the Primary just before the failure. However, if the safeness level used is 1-safe, some transactions can be lost in failover.

Running the new Primary in PRIMARY ALONE state

Although the connection to the Secondary server is broken when the Primary server is in the PRIMARY ALONE state, you can run a Primary server with

continuous updates to the transaction log. After the Secondary server comes back up, the server in PRIMARY ALONE state can resume sending transactions to the Secondary server.

Procedure

There are three ways to set a server to PRIMARY ALONE state:

- Issue the following command:

  ```
  ADMIN COMMAND 'hotstandby set primary alone';
  ```

- Perform a controlled disconnect at either the Primary or the Secondary with the following command:

  ```
  ADMIN COMMAND 'hotstandby disconnect';
  ```

  If you do a controlled shutdown by executing `ADMIN COMMAND 'shutdown';` on the Secondary, the Secondary disconnects implicitly before shutting down, and the Primary will switch safely to the PRIMARY ALONE state.

- Set the **HotStandby.AutoPrimaryAlone** parameter to `yes` to default to the PRIMARY ALONE state.

  When **HotStandby.AutoPrimaryAlone=yes**, the server is automatically put in PRIMARY ALONE state when the connection to the Secondary is broken. Otherwise, after a server fails, the state of the server remains PRIMARY UNCERTAIN unless the command `ADMIN COMMAND 'hotstandby set primary alone'` is issued by the HAC, the administrator, or the watchdog program. By default, the **HotStandby.AutoPrimaryAlone** parameter is set to `no`, which specifies that the Primary server operating in its PRIMARY ACTIVE state is switched to PRIMARY UNCERTAIN automatically if the Secondary server fails.

Results

The PRIMARY ALONE state persists until one of the following occurs:

- A connection is successfully made to the Secondary server.
- The server runs out of space for the transaction log.
- The log size limit (**MaxLogSize**) is reached.
- Another command switches the server to another state, such as STANDALONE.
- The Primary server is shut down.

Important: Do not shut down the Primary simultaneously with commanding Secondary to the PRIMARY ALONE state. The two operations are conflicting and might result in the Secondary gaining the SECONDARY ALONE state instead. These two actions do not coincide in normal operation.

Also, even during the test phase, do not try to simulate Primary failure using shutdown because shutdown is no substitute for failure. Shutdown is a complex distributed operation that involves both Primary and Secondary. Also, after a shutdown the Primary server that starts as a new Secondary cannot catchup with the new Primary. To shut down Primary, the correct sequence is:

1. Perform the switchover.
2. Shut down the new Secondary.
3. The new Primary switches automatically to the PRIMARY ALONE state.

Bringing the secondary server back online

Procedure

To bring the Secondary server back online, connect the Primary with the Secondary server. For more information, see 3.4.6, “Connecting HotStandby servers,” on page 59.
After you bring a Secondary node online, it might require catchup. Changes in the Primary have accumulated over time. While the Primary was set to PRIMARY ALONE state, the Primary wrote transactions and data to the transaction log. When the Secondary is connected again to the Primary, the pending changes at the Primary are written from the transaction log to the Secondary server for synchronization. While the changes are written to the Secondary, the Secondary is in SECONDARY ALONE state and the Primary is in PRIMARY ALONE state. If you issue the command **ADMIN COMMAND 'hsb status connect'**, the return message indicates whether the servers are performing catchup.

Note: If the Primary server was set to the STANDALONE state by using the command **hotstandby set standalone**, the full database must be copied from the Primary to the Secondary before the Secondary can be put in SECONDARY ACTIVE state. For more information, see 3.4.5, “Synchronizing primary and secondary servers,” on page 50.

Results

After the Secondary finished processing the pending changes, the Primary and Secondary server states are automatically changed to PRIMARY ACTIVE and SECONDARY ACTIVE.

3.4.4 Shutting off HotStandby operations

You might want to shut off HotStandby operations temporarily in the Primary server, for example, if you are taking the Secondary server out of service to upgrade it and the Primary does not have enough disk space to store the transaction logs that accumulates while the Secondary is out of service.

About this task

(See 3.6.3, “Running out of space for transaction logs,” on page 67 for more details.)

To shut off HotStandby at the Primary server:

Procedure

1. Disconnect the servers (if they are currently connected).
2. Set the Primary server to STANDALONE state, using the following sequence of commands:

   ```
   ADMIN COMMAND 'hotstandby disconnect'; -- if servers are connected
   ADMIN COMMAND 'hotstandby set standalone';
   ```

   The Primary server continues to operate as a non-HotStandby server.

   Note:

   After you have stopped storing transaction logs to send to the Secondary, you can no longer have the Primary and Secondary servers catch up merely by

connecting them again. Instead, you must synchronize the servers manually when you resume HotStandby operations.

What to do next

If you want to permanently stop using the server as a HotStandby server, see 3.4.10, “Changing a HotStandby server to a non-HotStandby server,” on page 64.

3.4.5 Synchronizing primary and secondary servers

To start HSB replication, the databases on the Primary and Secondary must be identical. The secondary database must be an exact copy of the primary database. The process of making the databases of a HotStandby system identical is called *HotStandby synchronization*.

The Primary and Secondary must be synchronized, for example, in the following situations:

- The Secondary is new and does not yet have a copy of the Primary database to start with.
- The Secondary was not running for awhile and its copy of the data is not up-to-date.
- Both the "Primary" and the "Secondary" were running in Primary Alone state at the same time, and thus have conflicting data.
- The Secondary disk drive fails, or the file is corrupted and must be replaced.

There are two main ways of synchronizing the data on the servers: *catchup* and *full copy*.

Catchup

Catchup can be used if and only if the Primary server has stored a copy of all of the transactions that the Secondary server "missed" while the servers were disconnected. If the Primary has stored all those transactions, when it is reconnected to the Secondary, it automatically forwards those transactions to the Secondary so that the Secondary can *catch up* with the Primary.

The solidDB server stores transactions (to forward to the Secondary) only while it is in the PRIMARY ALONE state, not while it is in the STANDALONE state or is operating as a non-HotStandby server. Therefore, if the server has been in STANDALONE state or has been operating as a non-HotStandby server since it last was connected with the Secondary, it does not have all the transactions and cannot do catchup. Instead, you must do a full copy.

There is no explicit catchup command. The servers try to catch up automatically when you connect them using the **`ADMIN COMMAND 'hotstandby connect'`** command.

When the Primary and Secondary are connected, they automatically check to see whether the Primary server has data in its transaction logs to send to the Secondary. If the data is there, the servers automatically attempt to catch up.

During the catchup process, the Primary and Secondary servers stay in PRIMARY ALONE and SECONDARY ALONE states. Clients can continue to submit queries and commit transactions. The catchup process is transparent to the client applications.

If the servers recognize that the Primary and Secondary databases are not identical even after the transactions are copied from the Primary to the Secondary, you will get an error message.

If catchup fails (or if you know ahead of time that it will not work because the Primary server was in STANDALONE state, for example), you must do a full copy.

Catchup applies only when the Secondary has already been running in SECONDARY ACTIVE state at some point. If you have a brand new Secondary server, and even if the Primary was running in PRIMARY ALONE state and has stored all transactions since the time that the Primary itself started, you must do a full copy to give the Secondary its initial copy of the database.

For more information about the catchup process, see "Bringing the secondary server back online" on page 49.

Full copy (`hsb netcopy`)

A full copy is just what its name implies: copying all the data from the Primary to the Secondary. Full copy is made by copying the database files themselves.

Full copy is used in the following situations:

- The Secondary server is brand new and is getting its initial copy of the Primary database.
- The Primary server has written transactions when it was not in the PRIMARY ALONE state, and therefore catchup is not possible.
- The Secondary database is corrupted or missing.
- The Secondary is diskless and has failed. When a Secondary diskless server is started after a failure, the diskless server requires a complete copy of the database by using the **hotstandby netcopy** command. Unlike a disk-based Secondary, the Secondary diskless server cannot read the transaction log and apply the changes that occurred while it was inoperable.
- The Primary server has all of the data that is needed for catchup, but catchup is expected to take longer than copying the current data files.

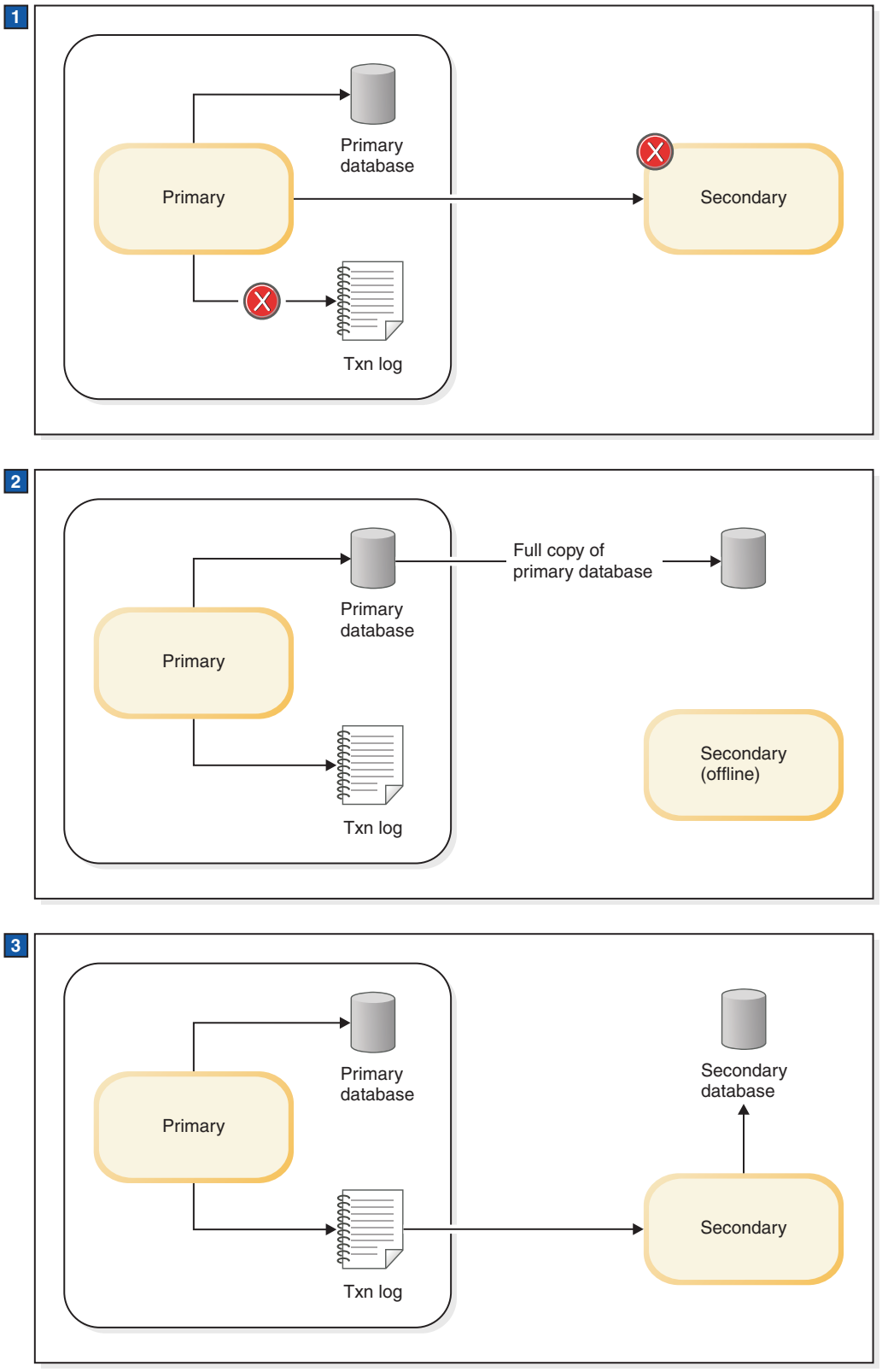
CAUTION:
If the Secondary server has old database files, a full copy writes over those old files. If for any reason the files on the Secondary contain data that was not in the Primary (for example, if both servers were operating in PRIMARY ALONE state at the same time), that data is lost.

There are two HotStandby commands that can do a full copy, that is, they copy the database files from the Primary to the Secondary. You can use either of the following commands:

```
ADMIN COMMAND 'hotstandby netcopy';
ADMIN COMMAND 'hotstandby copy [<directory_name>]';
```

The **netcopy** operation copies the database over the network to a Secondary server that is running and can receive the files over the network. The **copy** operation copies the database files to a specified disk drive directory that is visible to the Primary server. The secondary server must not be running during the **copy** operation. The **netcopy** command is usually preferable to the **copy** command, so most of the examples show only **netcopy**, not **copy**.

The **copy** and **netcopy** commands are described in "Copying a database file from the primary server to a specified directory" on page 57 and "Copying a primary database to a secondary over the network" on page 54.



1. Secondary is down for a long time so the primary stops using the transaction log to store data for the secondary. The log is still used for local recovery.
2. Primary database is copied to the secondary node and the primary starts writing to the transaction log.
3. Database is copied and the primary sends transaction log file changes to the secondary.

Figure 13. Manual full copy procedure

Note:

The preceding diagram oversimplifies the usage of the transaction log. In the first part of the diagram, when the Primary and Secondary are not connected, the Primary actually continues to write data to the transaction log, but keeps only enough data for recovery, not enough to allow the Secondary to catch up with all the changes since the connection was broken.

Using a watchdog to synchronize servers

The commands synchronize servers manually can also be used by a watchdog program to synchronize servers automatically.

If catchup is sufficient, all that the watchdog must do is monitor the Secondary to see when it comes up, and then issue the command to connect the Primary to the Secondary. If a full copy is required, the watchdog can instruct the Primary server to do a **netcopy** operation. A full copy writes over any data on the Secondary.

Copying a primary database to a secondary over the network

To send a copy of the database file from the Primary server to the Secondary server, use the **netcopy** command. The Secondary server must already be running.

Before you begin

- To start the **netcopy** operation, the Primary server must be in PRIMARY ALONE state.
- To use data compression during the **netcopy** operation, set the **HotStandby.NetcopyRpcCompress** parameter to `yes`.

About this task

There are two main situations in which you use **netcopy** to create a copy of the database for the Secondary server:

- You want to create a database for a brand-new Secondary that has never had one before. This method is also used to copy a database to a diskless Secondary, since after a failure it loses its entire database and must be treated as a brand-new Secondary.
- You want to replace an existing Secondary database. For example, one that has been corrupted.

Procedure

To start the **netcopy** operation, issue the following command:

```
ADMIN COMMAND 'hotstandby netcopy';
```

When the Primary does a **netcopy**, the Primary uses the connect string that is specified in the [HotStandby] section of `solid.ini`.
For details on the **HotStandby.Connect** parameter, which defines the connect string, see 3.2.1, “Defining primary and secondary node HotStandby configuration,” on page 37.
When you execute the **hotstandby netcopy** command, a database checkpoint is created before the copy of the Primary database is sent to the Secondary.
The Primary continues accepting transactions and connections during the netcopy (however, any ADMIN COMMAND that changes the server state are rejected.) The Secondary does not continue accepting transactions and connections. When the **netcopy** starts, if the Secondary has any open connections or transactions, it rolls back the open transactions and disconnect from its clients, it starts receiving the

netcopy. While the Secondary receives the **netcopy**, the Secondary communicates only with the Primary server.
When the **netcopy** is completed successfully, the Secondary state changes to SECONDARY ALONE (if it was not already in that state).
The Primary server stays in the PRIMARY ALONE state during the **netcopy** operation. After the **netcopy** is completed, the Primary server continues to stay in the same state. Before you can resume full hot standby operations, you must connect the Primary and Secondary servers. Connecting the servers sets the Primary server to PRIMARY ACTIVE state. For information about connecting the servers, see 3.4.6, "Connecting HotStandby servers," on page 59.

Creating a new database for the secondary server

Normally, when you start the solidDB server, it prompts you to create a new database if there is not already a database. However, if the server is a Secondary server, it must use a copy of the Primary database rather than create its own database. Therefore, when you start a Secondary server that does not have an existing database, you must give it a command-line parameter to tell it to wait to receive a copy of the database from the Primary. The command-line parameter is **-x backupserver**. For example, you start the Secondary server with the command:

```
solid -x backupserver
```

The space between the `-x` and `backupserver` is optional.

The **-x backupserver** command-line parameter sets the server into *netcopy listening mode* (also called *backup listening mode*). In the netcopy listening mode, the only possible operation for the Secondary server is to receive a database copy from the Primary server. The Secondary does not respond to any other command, and it does not even accept a connection request from a client program such as **solsql**, your application, or a watchdog program.

If there is a Secondary database, you can start the server in a normal way. The server starts in the SECONDARY ALONE state.

After the Secondary has been started with **-x backupserver**, or is in the SECONDARY ALONE state, you can execute the **netcopy** command on the Primary.

First, make sure that the Primary is in PRIMARY ALONE state. Then, issue the following command on the Primary:

```
ADMIN COMMAND 'hsb netcopy';
```

On the Primary, the **hotstandby netcopy** command uses the connect string that is defined with the **Connect** parameter in the `solid.ini` configuration file to connect to the Secondary server. After the servers are connected, the Primary copies the database files through the network link. In the netcopy listening mode, the Secondary server attempts to open the Secondary database only after it has received a new database copy through the **hotstandby netcopy** command at the Primary.

Following is the procedure to create the Secondary database copy:

1. Ensure that you have configured the `solid.ini` file so that it is valid for the HotStandby configuration. For more information about the **Connect** parameter, which defines the connect string, see 3.2.1, "Defining primary and secondary node HotStandby configuration," on page 37.

The connect string is used to connect to the Secondary server from the Primary and to copy the database files over the network.

2. Start the Primary server.
3. Start the Secondary server in netcopy listening mode by executing the following command:

   ```
   solid -x backupserver
   ```

   Or, alternatively, start the Secondary server with an existing database.
4. Set the Primary server to PRIMARY ALONE state if it is not already in that state:

   ```
   ADMIN COMMAND 'hotstandby set primary alone';
   ```
5. Issue the following command at the Primary server:

   ```
   ADMIN COMMAND 'hotstandby netcopy';
   ```
6. After the **netcopy** has completed, you can connect the two servers and start (or resume) full hot standby operation by issuing the command:

   ```
   ADMIN COMMAND 'hotstandby connect';
   ```

When the Secondary server receives a new copy of the database through the network link, it opens the Secondary database. After the servers are connected (with the **hsb connect** command), the Secondary server runs in its normal SECONDARY ACTIVE state and is ready to accept user transactions from the Primary.

If HAC is used, the procedure to get the Primary server database copied to the Secondary is as follows:

1. Ensure that the servers have the correct **Connect** parameters.
2. Ensure that the HAC in Primary node has the **PreferredPrimary**=Yes parameter set in solidhac.ini, and that the HAC in the Secondary node does not. For further information of configuring HACs see 3.3, “Configuring HA Controller and HA Manager,” on page 42.
3. Start the HAC instances, or optionally set HACs to the AUTOMATIC mode.

Note: If **netcopy** is sent to a server that is in the SECONDARY ALONE state, the existing database is overwritten with the copied database. Netcopy is useful if you want to resynchronize databases or repair a corrupted Secondary database.

Replacing an existing database on the secondary server

Although **netcopy** is used primarily to send a database to a Secondary that has never had a database before, **netcopy** can be used in other situations as well. For example, if the Secondary copy of the database is corrupted, for example, after a disk drive failure, you can send the Secondary a copy of the Primary database by using the **netcopy** command.

Before you begin

If you are replacing an existing database, then the Secondary server does not have to be in "netcopy listening mode"; in other words, you do not have to start the Secondary server with **-x backupserver**.

Procedure

1. Make sure that the Primary is in PRIMARY ALONE state and the Secondary is in SECONDARY ALONE state.
2. Issue the following command to the Primary:

```
ADMIN COMMAND 'hotstandby netcopy';
```

After **netcopy** completes, the Primary server is in PRIMARY ALONE state and the Secondary server is automatically put into SECONDARY ALONE state (if it was not already in that state).

If you do a **netcopy** while the Secondary is in SECONDARY ALONE state, and if any clients are connected to the Secondary (to do read-only queries), then the Secondary server rolls back any open transactions and breaks any client connections. Once the **netcopy** is completed, the Secondary server remains in the SECONDARY ALONE state.

3. The servers do not connect automatically: To connect the servers, issue the following command:

```
ADMIN COMMAND 'hotstandby connect';
```

Verifying netcopy status

When you start a **netcopy** command, it runs asynchronously in the background. The servers do not display a message when the **netcopy** completes – instead, you have to monitor the status of the **netcopy**.

About this task

The servers do not display a message even if the **netcopy** fails due to a problem such as a network error.

Procedure

Issue the following command at the Primary server:

```
ADMIN COMMAND 'hotstandby status copy';
```

Note: The command uses the keyword `copy`, not `netcopy`. The same command is used for both the **copy** and **netcopy** operations.
The command displays a status message that indicates whether the **netcopy** was successful, is still in progress, or failed.

Copying a database file from the primary server to a specified directory

If the directory that the Secondary uses for the database is visible to the Primary, you can use the **hotstandby copy** command to copy the database from the Primary to the Secondary.

Before you begin

This task is only possible in cases where the Primary and Secondary servers acan see some of the same disk drives and therefore can read and write some of the same directories.

CAUTION:
Before you issue the `hotstandby copy` command, shut down the Secondary server. The Secondary server must not try to access the database file while the Primary is writing that file.

Note: The Primary server must be in PRIMARY ALONE state when you issue the **hotstandby copy** command, and the Primary server remains in that state during (and after) the command.

About this task

The key difference between the **hotstandby copy** command and the **hotstandby netcopy** command is that the **netcopy** command can be used only when the Secondary is running. The **copy** command can be used only when the Secondary server is not running. Performance-wise, there is no significant difference between the two database copy methods.

Procedure

1. To copy the file by using **hotstandby copy**, issue the following command at the Primary server:

   ```
   ADMIN COMMAND 'hotstandby copy[directory_name]';
   ```

   where *directory_name* is the name of the directory that you want to copy the file to. The format of the directory name is operating system dependent. The directory name is optional. If you do not specify a directory name, then the server uses the value that is specified by the **HotStandby.CopyDirectory** parameter.

   When you issue the **hotstandby copy** command, the server creates a checkpoint and makes a copy of the Primary database before it sends the copy to the Secondary.

   Since the server is in PRIMARY ALONE state, transaction processing on the Primary continues normally during the **copy** command, and the Primary stores the transactions in the transaction log so that they can be forwarded to the Secondary later.
2. After a copy operation, the Secondary is still down. You must bring it back up and then issue the **hotstandby connect** command to connect the two servers.

   When the Primary database is connected to the Secondary using the administrative command **hotstandby connect**, the Primary and Secondary servers automatically perform "catchup" to bring the Secondary up-to-date.

Starting the secondary server and catching up:

When the copy is completed, you must start the Secondary server with the newly copied database.

Procedure

1. Start the Secondary server.
2. Use the **hotstandby connect** command at the Primary server to connect the Primary server to the Secondary server.

   ```
   ADMIN COMMAND 'hotstandby connect';
   ```

   For more information about he **hotstandby connect** command, see 3.4.6, “Connecting HotStandby servers,” on page 59.

Results

After the Primary is connected to the Secondary, the Primary server and Secondary server automatically start performing catchup. During catchup, the Primary server brings the Secondary database up-to-date by copying the Primary transaction log to the Secondary. Then the Secondary rolls forward the transaction log and updates its copy of the database.

3.4.6 Connecting HotStandby servers

About this task

The connect string that the Primary uses to connect to the Secondary server is specified using the **HotStandby.Connect** parameter.

You can view current connect settings in the Primary and Secondary nodes by issuing the command:

```
ADMIN COMMAND 'hotstandby cominfo';
```

Procedure

If the connection between the Primary and Secondary servers is broken or not yet established, you must issue the following command at the Primary or Secondary node:

```
ADMIN COMMAND 'hotstandby connect';
```

For example, after performing a **netcopy**, you normally connect the servers.
Since there is no automatic connect mechanism in the HotStandby servers, your high availability control application must issue the command when the connection between the servers is broken.
After issuing this command, a confirmation message is displayed if the connection between the Primary and Secondary servers is successful. If the Primary and Secondary are connected, but the transaction log is not yet fully copied at the Secondary, the Primary server returns the following message: `Started the process of connecting the servers`.
If the state of the Primary server was PRIMARY UNCERTAIN or PRIMARY ALONE when you issued the command, and if the connection is successful, the state of the Primary server changes to PRIMARY ACTIVE. If unsuccessful, the state remains PRIMARY UNCERTAIN or PRIMARY ALONE.

What to do next

For more information about querying the connect status at Primary and Secondary servers, see “Displaying connect status information” on page 60. For more information about reconnecting an application to the Primary server, see 4.3.1, “Reconnecting to primary servers from applications,” on page 94.

3.4.7 Checking HotStandby status

You can request information about the HotStandby status from both the Primary and Secondary servers.

Procedure

To check status, issue the following command in the Primary or Secondary server:

```
ADMIN COMMAND 'hotstandby status option';
```

where *option* can be one of the following:

Option	Description
catchup	Indicates whether the server is doing catchup. Catchup occurs after the Primary server connects to the Secondary. During catchup, the Primary sends accumulated transaction logs so that the Secondary can apply the changes. Possible values are: 'ACTIVE' and 'NOT ACTIVE'.
connect	Shows whether the last attempt to connect the servers was successful.
copy	Shows whether the last attempt to **copy**/**netcopy** was successful.
switch	Shows whether the last attempt to switch the server into PRIMARY ACTIVE or SECONDARY ACTIVE state was successful.

Example

```
ADMIN COMMAND 'hotstandby status catchup';
```

Displaying switch status information

You can use the **ADMIN COMMAND 'hotstandby status switch'** command to verify whether a state switch occurred between two HotStandby servers.

About this task

To check HotStandby switch status information:

Procedure

Issue the following command in the Primary or Secondary server:

```
ADMIN COMMAND 'hotstandby status switch';
```

- When no prior switch has occurred between the two servers, the following message is displayed: `NO SERVER SWITCH OCCURRED BEFORE`.
- When the switch process is still active, the following message is displayed: `ACTIVE`.
- When the most recent prior switch process has completed successfully, the following message is displayed: `SUCCESS`.
- When the most recent attempt to switch has failed, the following message is displayed: `ERROR` *number*, where *number* identifies the type of error that occurred during the switch.

Displaying connect status information

You can query connect status information between the Primary and Secondary servers. This capability is equivalent to the SQL function HOTSTANDBY_CONNECTSTATUS, which you can use in the application code.

Procedure

To check connect status, issue the following command in the Primary or Secondary server:

```
ADMIN COMMAND 'hotstandby status connect';
```

The possible return values are:

Table 8. Connect status return values

Error Code	Text	Description
0	CONNECTED	Connect active. Returned from both the Primary and Secondary server.
14007	CONNECTING	Primary server is connecting to the Secondary server. Returned from both the Primary and Secondary servers.
14008	CATCHUP	Primary server is connected to the Secondary server, but the transaction log is not yet fully copied. Returned from both the Primary and Secondary server.
14010	DISCONNECTING	The servers are in the process of disconnecting.
14537	BROKEN	Connection is broken. Returned from both the Primary and Secondary servers.

Displaying connection information

You can query the connection information that the HotStandby server uses to connect to the other server by using the **ADMIN COMMAND 'hotstandby cominfo'** command.

Procedure

To display connection information, issue the following command in the Primary or Secondary server:

```
ADMIN COMMAND 'hotstandby cominfo';
```

The command returns the connection information that is defined with the **HotStandby.Connect** parameter.

Displaying role start time

Sometimes it is important to know when the server entered the current role of Primary or Secondary.

You can retrieve information about the start time of the Primary or Secondary role by using the **ADMIN COMMAND 'info'** command with the options `primarystarttime` and `secondarystarttime`.

For example:

```
admin command 'info primarystarttime';
       RC TEXT
       -- ----
        0 2005-06-09 14:22:18
admin command 'info secondarystarttime';
       RC TEXT
       -- ----
        0 2005-06-09 18:24:44
```

The reported time is the time the server role became Primary or Secondary. The STANDALONE state is considered to be a state of the Primary role.

The primary start time is set when the following transitions occur:

- SECONDARY ALONE => PRIMARY ALONE
- SECONDARY ALONE => STANDALONE
- SECONDARY ACTIVE => PRIMARY ACTIVE

The secondary start time is set when the following transitions occur:

- The server is started in the SECONDARY ALONE state
- OFFLINE (started with **-x backupserver**) => SECONDARY ALONE
- PRIMARY ALONE => SECONDARY ALONE
- STANDALONE => SECONDARY ALONE
- PRIMARY ACTIVE => SECONDARY ACTIVE

If the current role contradicts the query, the query returns an empty string. For example, if the role is SECONDARY and the command `info primarystarttime` is issued, it returns an empty string.

3.4.8 Verifying HotStandby server states

You can check the state of HotStandby servers by using the **ADMIN COMMAND 'hotstandby state'** command.

Procedure

To check the current state of a HotStandby server, issue the following HotStandby command in the server:

```
ADMIN COMMAND 'hotstandby state';
```

Results

For descriptions of possible states the command returns, see "Description of server states" on page 8.

Note:

1. If the server is configured as a HotStandby server, when the server is started, the server starts in the SECONDARY ALONE state.
2. The server cannot return the state OFFLINE because when the server is in the OFFLINE state, you cannot connect to it and issue any query (such as ADMIN COMMAND 'hotstandby state').
3. If **ADMIN COMMAND 'hotstandby state'** is issued on a server that is not configured as a HotStandby, the following error message is returned:

   ```
   14527: This is a non-HotStandby Server
   ```

   If HAC, and HA Manager are used, the HA Manager displays the HSB states of both servers. The information is queried periodically every second.

What to do next

For a summary of HotStandby state transitions that occur while performing administrative and troubleshooting operations, see Appendix D, "Server state transitions," on page 161.

Server state combinations

All combinations of server states are not possible. For example, if the Primary is in PRIMARY ACTIVE state, the Secondary can be only in SECONDARY ACTIVE state.

The following table shows possible server states of a HotStandby server when its associated server is in a particular state.

Table 9. Server states

State of the server	Possible states of the associated server
PRIMARY ACTIVE	SECONDARY ACTIVE
PRIMARY ALONE	PRIMARY ALONE * PRIMARY UNCERTAIN SECONDARY ALONE STANDALONE *
PRIMARY UNCERTAIN	PRIMARY ALONE PRIMARY UNCERTAIN SECONDARY ALONE STANDALONE
SECONDARY ACTIVE	PRIMARY ACTIVE
SECONDARY ALONE	PRIMARY ALONE PRIMARY UNCERTAIN SECONDARY ALONE STANDALONE
STANDALONE	PRIMARY ALONE * PRIMARY UNCERTAIN SECONDARY ALONE STANDALONE *

* If one server is in the PRIMARY ALONE state or STANDALONE state, the other server cannot be in the PRIMARY ALONE or STANDALONE state. This is because if changes are made to both servers independently, there is no way to merge the two databases into one.

3.4.9 Choosing which server to make primary

In some error recovery situations, you might not know which server you must make the Primary. You can use the **hsb logpos** command to determine which server to make the Primary.

About this task

This procedure can be used, for example, in a case where both databases have failed; the server that was the Primary before the servers lost contact with each other is not necessarily the server that should become the Primary now.

Procedure

1. Ensure that both servers are in SECONDARY ALONE state.
2. Connect to both servers.
3. In each server, issue the following command:

```
ADMIN COMMAND 'hsb logpos';
```

This commands returns the log operation ID and the server's role (PRIMARY, SECONDARY, STANDALONE) at the time of the last operation. Successful admin commands return error code 0, a string, and the previous role of the server.

For example:

```
ADMIN COMMAND 'hsb logpos';
RC TEXT
-- ----
0 00000000000000000871:PRIMARY
```

The application must regard the string as an opaque value, which has no defined structure.

4. Compare the string values.

   For example, in C, use the strcmp() function.

   In principle, the server that has the greater value for the log operation ID has accepted more transactions, and thus should become the Primary. However, if you have made updates to both servers after the HSB connection has failed, the log operation ID values can no longer be compared reliably.

   If HAC is used and the STRINGS are equal, the **LocalDB.PreferredPrimary** HAC parameter defines whether the local server becomes the Primary.

5. Select the Primary by using the following command on the server that will become Primary:

   ```
   ADMIN COMMAND 'hsb set primary alone';
   ```

6. Connect the HotStandby servers with each other by using the following command:

   ```
   ADMIN COMMAND 'hsb connect';
   ```

Results

- If the **hsb connect** command succeeds, the Secondary catches up with the Primary, and the HotStandby pair is functional again.
- If the **hsb connect** command fails, you must synchronize the nodes separately. For more information, see 3.4.5, "Synchronizing primary and secondary servers," on page 50.

Important: This procedure does not ensure that the server with the higher string value is a superset of the other server. The two servers might have each accepted transactions that the other did not. For example, both servers might have been running in the PRIMARY ALONE state. To detect the possibility that neither server is a superset of the other, the servers compare information when the **hsb connect** command is issued. If neither server is a superset, the **hsb connect** command fails with an appropriate error message.

3.4.10 Changing a HotStandby server to a non-HotStandby server

You can change a Primary or Secondary server to become a normal, non-HotStandby server by editing the [HotStandby] section of the solid.ini file.

Procedure

1. Remove the **HSBEnabled** parameter (or set it to no).
2. (Optional) Remove or comment out the **Connect** parameter.
3. To make the changes in the solid.ini file effective, restart the server.

Example

If you want the server to temporarily stop acting as a HotStandby server, but you would like it to resume acting as a HotStandby server later, you can leave the `solid.ini` file unchanged and change the state of the server to STANDALONE.

3.5 Performance tuning

3.5.1 Tuning replication performance with safeness and durability levels

The performance of data replication during normal operation depends on the setting of the durability level and safeness level. Additionally, when 2-safe replication is used, the acknowledgement policy that is used in 2-safe mode affects the latency time, as perceived by the application. For more information, see 1.2, “Performance and HotStandby,” on page 18.

3.5.2 Tuning netcopy performance

The **`netcopy`** command allows the Primary database to be copied to a remote Secondary. The **`netcopy`** command is also used to copy a database from a Primary server to a Secondary server when one or both servers are diskless. The Primary database files are copied through a network link.

The connect string that is used to connect to the Secondary server for the netcopy operation is specified with the **`HotStandby.Connect`** parameter.

Controlling netcopy block size

You can tune the performance of netcopy (and the performance of backup) by increasing or decreasing the block size of the database file when it is copied from the Primary to the Secondary server. You can define the block size with the **`General.BackupBlockSize`** parameter. Generally, larger block size means faster netcopy and backup, but at the cost of possibly slowing down the server response time to other requests while the netcopy or backup is being done.

By default, the **`General.BackupBlockSize`** parameter is set to 64K.

You can set the value in bytes, or by using the suffixes "M" and "K". For example:

```
[General]
BackupBlockSize = 32K
```

or

```
[General]
BackupBlockSize = 32768
```

Note:

- The minimum value for **`General.BackupBlockSize`** is the server block size (defined with **`IndexFile.BlockSize`** parameter).
- The maximum value is `8MB`. If the parameter value exceeds the maximum value, the default value is used (64K).
- The value of **`General.BackupBlockSize`** needs to be a multiple of the database block size of the server (defined with **`IndexFile.BlockSize`** parameter).

Related tasks:

"Copying a primary database to a secondary over the network" on page 54
To send a copy of the database file from the Primary server to the Secondary server, use the **netcopy** command. The Secondary server must already be running.

3.5.3 Tuning database catchup performance

When a failed Secondary server is back in service and connected to Primary, HotStandby continues sending the Primary transaction log file contents to the Secondary node in an automated process that is known as *HotStandby database catchup*.

You can tune the performance of the database catchup by adjusting how much time the server spends on catchup, as opposed to servicing current client database queries. The catchup time is controlled with the **HotStandby.CatchupSpeedRate** parameter, which defines in percentages the time the server spends on catchup. For example:

```
[HotStandby]
CatchupSpeedRate = 90
```

If **HotStandby.CatchupSpeedRate** is assigned a value of 90, the server spends approximately 90% of its time on catchup and about 10% of its time responding to user queries.

The higher the **HotStandby.CatchupSpeedRate** value is, the faster the catchup. However, a higher catchup rate also affects other activities, such as user queries. By default, **HotStandby.CatchupSpeedRate** is set to 50.

3.6 Special considerations for using solidDB with HotStandby

When you are using solidDB with HotStandby, you must pay special attention to the transaction isolation level, transaction log space, unplanned network partitioning, and throttling.

3.6.1 Transaction isolation level and in-memory tables

If you are connected to the Secondary and you are reading data from in-memory tables, the transaction isolation level is automatically set to READ COMMITTED. The transaction level is READ COMMITTED even if you have specified it as REPEATABLE READ, or SERIALIZABLE.

Additionally, if load balancing is used, the isolation level is READ COMMITTED by default.

3.6.2 Network partitions and dual primaries

In some circumstances, both servers can be in the PRIMARY ALONE state. Having *dual primaries* can lead to serious, unrecoverable errors.

In the dual primary situation, if each server commits any transactions that the other does not, you cannot resynchronize the servers, because the databases cannot be merged to create a single database that has correct information. In practice, the transactions committed in the "wrong primary" database during the dual primary situation, will be lost. Having dual primaries can also lead to other errors.

The dual primaries problem is most likely to be caused by a *network partition*. In a network partition situation, some but not all network connections are lost and your single network effectively becomes divided into separate subpieces. Each subpiece

can communication within the piece but not with other pieces. Thus both servers lose connections with each other, but are still up and running, and in some cases can still communicate with some clients.

The dual primaries scenario can be avoided by using a single-instance watchdog in a node that is external to the HSB system. By using such a watchdog, it is easy to decide which server must be set to Primary and to ensure that clients see only one Primary.

Although HAC in composed of two instances that running in the same nodes with HSB servers, the dual primary situation is impossible when HAC is used with ERE.

Even if you have dual primaries, you do not have inconsistent data unless someone is able to perform a write operation on the original Secondary (after it has switched to PRIMARY ALONE). If the original Secondary is cut off from the rest of the network, no one can write to it. The original Primary is a superset of the Secondary, and you can get a single consistent set of data (after you reconnect the servers and allow the Secondary to catch up with the changes made on the original Primary).

Although dual primaries are rare, they are dangerous when they do occur. You must plan you environment so that you can prevent your data from becoming inconsistent. For example, use ERE.

The chances of dual primaries increase if you set the configuration parameter **AutoPrimaryAlone** to Yes for one or both servers. The **AutoPrimaryAlone**=Yes parameter setting means that your system can respond to failures quickly, but it also means that the system no longer has any independent observer (HAC, watchdog or human) to prevent dual primaries. If you have any doubts on your network reliability, keep the **AutoPrimaryAlone** parameter in its factory value, that is, No.

3.6.3 Running out of space for transaction logs

When you use HotStandby, if you put a server in PRIMARY ALONE state, you must be careful that it does not run out of disk space for transaction logs. In a non-HotStandby server, if you checkpoint frequently, then the transaction log does not grow large because after each checkpoint the server deletes the old transaction logs.

In particular, the non-HotStandby server deletes the logs with the data changes that occurred before the checkpoint. For more information about checkpointing, see *solidDB Administration Guide*.

However, in a HotStandby server that is operating in PRIMARY ALONE state, the server must keep the transaction logs that have accumulated since the time that the Primary lost contact with the Secondary. If the Secondary is down for a long time, the server might keep a large amount of transaction log data that it would otherwise throw away after each checkpoint. In a worst-case situation, if the Secondary cannot be brought back up in a reasonable time and there is not enough disk space to store all the transactions that occur, the Primary transaction logs can fill up all of the available disk space. When the disk is full, the server switches to read-only mode.

To prevent running out of disk space for transaction logs, you can control the log size with the parameter **MaxLogSize**. After the log reaches the specified total log size, the server automatically switches to the STANDALONE state at the next checkpoint. In a diskless server, the state remains PRIMARY ALONE, even though there is no disk writing at all.

If the server is set to the STANDALONE state, it no longer keeps all transactions logs since the time that the Primary lost contact with the Secondary. Without complete transaction logs, you cannot synchronize your system merely by connecting the Primary to the Secondary and allowing the Secondary to "catch up" by reading old logs. You have to copy the entire database from the Primary to the Secondary by using the **copy** or **netcopy** command.

If HAC is used, it identifies the situation described above, and leads the servers to the ACTIVE state by automatically performing the necessary actions.

3.6.4 Throttling and multiprocessing in Secondary

To keep the data in the Primary and Secondary server up-to-date, the Primary writes to its transaction log and forwards it to the Secondary so that the Secondary can make the same changes to its copy of the database. If the Secondary cannot keep up with the processing pace of the Primary, the solidDB throttling mechanism slows down the processing on the Primary. From the application standpoint, throttling results in increased response times.

As of V7.0, the Secondary can use multiple threads for processing the write loads. The Secondary receives operations from the Primary in one stream. Operations are parsed and each transaction is attached in execution queue; there are as many execution queues as there are open parallel transactions. To ensure that conflicting operations are executed in the correct order, row locks are used.

You can monitor the use of threads with the following performance counters (pmons):

Table 10. Pmon counters for monitoring multiprocessing in the Secondary

Perfmon Variable	Description
HSB grpcommits	Number of transactions in the most recent group commit Transaction commits are grouped in one-log bursts which are sent to Secondary as one packet. You can use this counter only on the Primary.
HSB secondary ops in packet	Number of log records the Secondary received from the Primary in the most recent log record packet.
HSB secondary trx count	Number of open transactions the Secondary has received from the Primary
HSB secondary locks	Number of row-level locks on the Secondary
HSB secondary lock reqs	Number of lock requests on the Secondary
HSB secondary lock waits	Number of lock waits on the Secondary since the server was started
HSB secondary op waits	Number of times operations (transactions) on the Secondary have been waiting to continue execution
HSB secondary buffers	Number of buffered log record packets the Secondary has received from the Primary

Table 10. Pmon counters for monitoring multiprocessing in the Secondary (continued)

Perfmon Variable	Description
HSB secondary serial mode count	Number of times the Secondary parallel executor has switched to serial mode instead of running in parallel
HSB secondary dispatch queuelength	Size of the most recent dispatch thread (operations to dispatch) on the Secondary

You can also optimize the multiprocessing level of the Secondary with the **HotStandby.SecondaryThreads** parameter. The optimal number of threads depends on the environment and requires experimenting with loads typical to your environment. In principle, the more processing capacity (cores) the computer has, the higher you can set the value of the **HotStandby.SecondaryThreads** parameter. However, excessive use of threads is not likely to improve performance.

3.7 Configuring for lower cost versus higher safety

The HotStandby solution uses pairs of Primary and Secondary servers to provide true high availability. However, using pairs of servers might not be optimal for every use scenario. If near-instantaneous failover is not required, you might not be able to justify the expense of having a Secondary for every Primary server. At the other extreme, some business cases might need extra reliability and also warrant the resources to purchase *spares for the spares*. Spare for spares means that you might want to purchase not only a Secondary for every Primary, but also one or more spare servers so that when a Primary fails and its Secondary replaces it, a spare can be used as the new Secondary if the original Primary cannot be repaired quickly.

To help you to reduce costs or increase reliability, HotStandby supports various alternatives to the so called N+N or 2N standard hot standby model (in the standard model the number of Primary and Secondary servers is the same (N)). The alternatives include:

- N + 1 Spare or N + M Spares: This is the Spare Node scenario for stand-alone servers. There are N "primary" servers and one or more spares. There are no Secondary servers. A failed (primary) server is replaced with a spare. The stand-alone server case is not a true hot standby scenario. It can be called *warm standby*: the spare computer is available but it does not have a copy of the database.
- 2N + 1 Spare or 2N + M Spares: This is the Spare Node scenario for HotStandby. There are N HotStandby pairs, that is, every Primary has a Secondary. In addition, there are M spares, where M is at least one and usually less than N. When a Primary or Secondary fails, a spare is brought in as the new Secondary. Thus a Primary server never operates alone for long, even if its original partner has failed.

3.7.1 Reducing cost: N + 1 spare and N + M spares scenarios

In the N + 1 spare and N + M spares scenarios, there are N "primary" servers, each of which operates in STANDALONE state, that is, without being connected to a Secondary. In addition, there are M spare servers, where M is at least 1 and usually less than N. If a "primary" server fails, one of the spares replaces it. Data is copied from the original server to the spare, the original server is taken offline, and the spare is configured to act as the original server. Any spare can replace any Primary server (no spare is dedicated to a particular Primary server). Also, failover is not almost instantaneous.

The *N + 1* approach is referred to as *single-spare* scenario and *N + M* is referred to as *multiple-spare* scenario.

Both approaches require that you have a copy of the data of the original server somewhere. The single-spare and multiple-spare scenarios do not prevent data lost if the disk drive of the original server is damaged and there is no backup of the data.

The N+M approach can be useful in the following situations:

1. You are using the spare nodes to handle scheduled maintenance, not unexpected failures.
2. You have reliable backups that you can quickly copy to the spare server.
   a. You have backups on tape or on some other safe location.
   b. You are using solidDB advanced replication technology, and you can copy or re-create enough of the data by reading from the advanced replication master or replicas of the server that failed.
3. Individual pieces of data are not critical or are not unique.
   a. For example, if what you need is the "computing horsepower" (load-spreading capability) rather than the specific data, you might be able to meet your needs by copying a standard or "seed" database, or getting the data from clients, and then continue to run.
   b. Similarly, if all the servers have approximately the same data and are responding almost entirely to read requests with few or no write requests, you can copy a useful database from any one of your computers. For example, if you are running a large number of servers that all use the same internet routing tables, or telephone directory information, you can use any of the server for recovery of data.

3.7.2 Increasing reliability: 2N + 1 spare and 2N + M spare scenarios

Normal HotStandby operation is highly reliable. The odds of both the Primary and Secondary failing at nearly the same time are low, if they use separate reliable power supplies. But suppose that you want to reduce even the risk of power failure, or suppose that the server that failed cannot be repaired rapidly? Ideally, when a Primary fails and you replace it with a Secondary (or when a Secondary fails), you want to have a new Secondary that replaces the old Secondary so that you can continue to run with a complete pair of servers.

This situation is called the *2N + 1 Spare* (or *2N + M Spares*) scenario. You have N Primary servers, N Secondary servers, and at least 1 spare that can replace any Secondary that has failed or has been converted to a Primary. Spares are not dedicated to a particular server (or HSB pair of servers), and some configuration is required before the spare can replace the failed server.

3.7.3 How solidDB HSB supports the N+1 (N+M) and 2N+1 (2N+M) approaches

You must make a spare server look like the server that it is replacing. Typically, this means:

1. You must copy data to the spare.
2. You must tell the spare to "listen" at the same network address as the server that it is replacing, or at another address that the client programs know to communicate through.

3. In addition, in the 2N+1 (2N+M) scenario, you must also tell the new Secondary server and the current Primary server how to communicate with each other, In other words, you must tell each of them the address to use to connect to the other.

solidDB has two features to support such needs:

- You can copy data to the spare server without shutting down the spare server.
- You can set certain configuration parameters dynamically.

1. Although solidDB configuration parameters are normally set by shutting down the server, updating the `solid.ini` configuration file, and then restarting the server, it is also possible to change some configuration parameters (such as the "com.listen" and "hotstandby.connect" parameters) by executing ADMIN commands similar to the following examples:

   ```
   ADMIN COMMAND 'parameter com.listen="tcp SpareServer1 1315"';
   ADMIN COMMAND 'hsb parameter connect "tcp srvr27 1316"';
   ```

   This means that a spare can be configured dynamically to take the place of another server without shutting down first. Similarly, a Primary can be told the Connect string of its new Secondary.

   Tip: Executing the commands do not write the updated parameter values to the `solid.ini` file. To ensure that the server has the new values the next time it restarts, update the parameter values also in the `solid.ini` file.

   Important:

   The spare server must be started with the command-line option **`-x backupserver`** so that it is ready to receive the netcopy from the Primary server. For more information about the **`-x backupserver`** option, see "Creating a new database for the secondary server" on page 55.
2. The **`netcopy`** command copies a database to a server that is already up and running.
   a. Set the new value of the "connect" parameter:

      ```
      ADMIN COMMAND 'hsb parameter connect "tcp srvr27 1316"';
      ```

   b. Execute the **`netcopy`** command:

      ```
      ADMIN COMMAND 'hsb netcopy';
      ```

   c. Connect the current Primary with the new Secondary by executing the command:

      ```
      ADMIN COMMAND 'hsb connect';
      ```

3.7.4 Using HAC with spares

HAC has limited support on spare scenarios. It can be used in a spare, but before HAC in the Primary node is able to start monitoring the HSB state of the spare server, the connect information for the Primary HAC (**`RemoteDB.Connect`**) must be updated. You must update the `solidhac.ini` file and restart the HAC in the Primary node.

Similarly, the HAC in the spare node needs the connect information of the Primary server. If the connect information is not known beforehand, the information must be added in the `solidhac.ini` file. After you have added the information to `solidhac.ini`, you must restart HAC.

4 Using HotStandby with applications

To use HotStandby, your application design must address how your application connects to the HotStandby server and how the application handles failures and failovers.

The way the application connects to the HotStandby server pair might vary depending on where the application resides (compared to the database it is using), and depending on the preferred level of automation in failover situations.

In general, when an application uses a database, it connects to one database, and uses that database for all the queries, reads, and writes. If that database becomes unavailable, the connection is broken and the application must wait until it can reconnect again to the same database. With a HotStandby pair, there are two databases, both live and running, mirroring each other and having the same content. The application has more options available and more responsibilities of making sure that the service the application provides to its users is able to continue even if one of the database nodes fails.

4.1 Connecting to HotStandby servers

You can use two connection methods for your applications in HotStandby environments, *Basic Connectivity* and *Transparent Connectivity* (TC). With Basic Connectivity, the client connects to each of the HSB servers explicitly. With the Transparent Connectivity, the client enacts only one logical connection that is called the TC Connection.

Both connectivity types are supported in the solidDB ODBC and JDBC drivers, as well as with SMA servers. The connectivity type is defined within the connect string. With Basic Connectivity, the standard solidDB connection syntax is used. With Transparent Connectivity, a TC-connection-specific syntax is used.

Basic Connectivity

With Basic Connectivity, the application has to take care of connecting to each server of the HotStandby configuration separately, by using specific server addresses. If a failover happens, the active connection is lost, and the application has to reconnect to the new Primary server.

Transparent Connectivity

With Transparent Connectivity, the application does not have to deal with connecting to any specific server, or to reconnect in the case of a failover. The application maintains a logical connection (handle) called a *TC Connection*. In simple terms, the Transparent Connectivity relieves the application from taking care of the multiplicity of servers and their addresses.

The connection handle is maintained over failovers and switchovers for as long as there is any server in the PRIMARY ACTIVE, PRIMARY ALONE or STANDALONE state, within the specified set of servers. At failovers and switchovers, the driver performs an internal operation called *connection switch*. The

application is notified about the connection switch, because the application must reconstruct some of the session states (depending on the failure transparency level).

Transparent Connectivity with network-based connection

With network-based connections, the application and the servers can be located on the same or different nodes.

Applications can use the load-balancing functionality to send read-only loads to the Secondary server.

Transparent Connectivity with SMA connection

With SMA, there are two applications, one on each HotStandby node. The application on the primary node uses a SMA connection for reads and writes. The application on the secondary uses a SMA connection to execute reads locally; write transactions from the application on the secondary are executed on the primary server using a network connection.

Server failovers and switchovers are handled in principle the same way as with network-based. However, if the application fails because the SMA server has failed, an application-specific high availability handling is needed.

4.1.1 Choosing the connectivity type

Procedure

The following compatibility matrix helps you choose the connectivity type by indicating the supported feature against the selected connect info:

Table 11. Choosing the connectivity type

Feature	Standalone config.	HSB configuration
Basic connectivity	Yes (BC Info)	Yes (BC Info)
Transparent failover	No	Yes (TC Info)
Load balancing	No	Yes (TC Info)

4.2 Transparent Connectivity

When using solidDB Transparent Connectivity, the solidDB ODBC or JDBC driver hides the existence of two HSB servers from the application. Transparent Connectivity also offers transparent load balancing between the primary and secondary servers.

The driver offers a single logical TC Connection that is mapped to the internal active connection. In an ideal case, when both Primary and Secondary servers are running in the active state, the driver also maintains the *standby connection*, that is, the connection to Secondary. This connection will be set to the event wait mode, where it is ready to receive HSB state change events. Those events are the primary source of information about failovers and switchovers that the driver will use. In some cases (such as the Primary Alone operation), the standby connection will be missing, but the driver will try to enact it whenever possible.

The standby connection is handled totally transparently to the application. Also, any occurrence of a connection switch, that is, changing the mapping of the TC Connection to an internal active connection, will be notified to the application by way of a special error code.

Important: solidDB tools, such as solidDB SQL Editor (**solsql**), do not support the TC connection.

4.2.1 Defining the Transparency Connectivity connection

Transparent Connectivity is specified using non-standard ODBC connection string settings or JDBC connection properties.

Syntax of Transparent Connectivity Info – ODBC

When using solidDB Transparent Connectivity, the client enacts only one logical connection called the TC Connection. With ODBC, the TC connection is specified in the TC Info. TC Info enacts transparent failover and load balancing in both HSB configurations.

With ODBC applications, the TC connection can be specified in the following ways:

- SQLConnect function:

```
rc = SQLConnect(comHandle, "<solidDB_TC_Info>", ...
```

- Client-side `solid.ini` file:

```
[Com]
Connect = <solidDB_TC_Info>
```

The syntax of the solidDB TC Info is:

```
<solidDB_TC_Info>::= {[<failure_transparency_level_attribute>]
[<preferred_access_attribute>] [<encryption_attribute>]
<connect_target_list>} | <cluster_info>

failure_transparency_level_attribute ::= TF_LEVEL={NONE |
CONNECTION | SESSION}

preferred_access_attribute::= PREFERRED_ACCESS={WRITE_MOSTLY |
READ_MOSTLY | LOCAL_READ}

encryption_attribute::=USE_ENCRYPTION={YES|NO} [USE_GSKIT={YES|NO}]

connect_target_list::= [SERVERS=]<connect_string>[, <connect_string > ...]

cluster_info::= CLUSTER= <connect_string>[, <connect_string>...]
```

The following abbreviations can be used.

Table 12. TC Info abbreviations

Abbreviation	Corresponding syntax
TF	TF_LEVEL
CO	CONNECTION
SES	SESSION
PA	PREFERRED_ACCESS
RM	READ_MOSTLY
WM	WRITE_MOSTLY
LR	LOCAL_READ
S	SERVERS

Failure transparency attribute

Failure transparency handles the masking of failures. The failure transparency level is set with the TF_LEVEL attribute of the TC Info. Three levels are available:

1. NONE – failure transparency is disabled. This is the default value.
2. CONNECTION – the server connection is preserved, that is, it is unnecessary to reconnect in the case of failover or switchover.
3. SESSION – certain session attributes that have non-default values are preserved. Additionally, prepared statements are preserved. However, open cursors are closed, and ongoing transactions are aborted.

Preferred access attribute – load balancing

The preferred access attribute (PREFERRED_ACCESS) indicates whether the load balancing of read-only loads is applied or not. The following levels are available:

- WRITE_MOSTLY – no load balancing (default). All transactions are executed on the Primary.
- READ_MOSTLY – load balancing by distributing read-only transactions between Primary and Secondary, as defined by **Cluster.ReadMostlyLoadPercentAtPrimary**
- LOCAL_READ – load balancing by executing read-only transactions locally when possible. Read-only transactions are always directed to the local server, be it Primary or Secondary. If local server cannot be found, the Primary server's assigned workload connection is used (first network-based connection defined in the connect string). Write transactions are always executed at the Primary server.

  LOCAL_READ is typical with SMA setups where there is at least one SMA application on each node.

Important: When using SMA with Transparent Connectivity (TC), if you set the load balancing method to READ_MOSTLY or WRITE_MOSTLY (default), a network connection is used instead of the SMA connection. Thus, when using SMA with TC, always set the load balancing method to LOCAL_READ.

Connect target list and cluster info – server addresses

The solidDB TC Info includes a list of server addresses, called *connect target list*. If the HotStandby configuration includes SMA, a SMA-specific connect target list must be used.

The syntax of the connect target list for network-based applications is:

```
[SERVERS:] <network_connect_string>, <network_connect_string>
```

The syntax of the connect target list for SMA applications is:

```
[SERVERS:] <sma_connect_string>, <network_connect_string>
```

network_connect_string

The format of a connect string for network-based connections is the following:

protocol_name [*options*] [*host_computer_name*] *server_name*

where

- *options* can be any combination of the following:

Table 13. Connect string options

Option	Description	Protocol
-4	Specifies that client connects using IPv4 protocol only.	TCP/IP
-6	Specifies that client connects using IPv6 protocol only. In Windows environments, this option is mandatory if IPv6 protocol is used.	TCP/IP

Table 13. Connect string options (continued)

Option	Description	Protocol
-i*source_address*	Specifies an explicit connecting socket source address for cases where the system default source IP address binding does not meet application needs. *source_address* can be an IP address or a host name.	TCP/IP
-z	Enables data compression for the connection **Important:** • Data compression is not available for HotStandby connections (**HotStandby.Connect**) and NetBackup connections (**ADMIN COMMAND 'netbackup'**). • Data compression for **netcopy** connections cannot be enabled with the **-z** option. Instead, use the **HotStandby.NetcopyRpcCompress=yes** parameter setting.	All
-c *milliseconds*	Specifies the login timeout (the default is operating-system-specific). A login request fails after the specified time has elapsed.	TCP/IP
-r *milliseconds*	Specifies the connection (or read) timeout. A network request fails when no response is received during the time specified. The value 0 (default) sets the timeout to infinite (operating system default timeout applies).	TCP/IP
-o*filename*	Turns on the Network trace facility and defines the name of the trace output file See *Network trace facility* in the *IBM solidDB Administrator Guide* for details.	All
-p*level*	Pings the server at the given level (0-5). Clients can always use the solidDB Ping facility at level 1 (0 is no operation/default). Levels 2, 3, 4 or 5 may only be used if the server is set to use the Ping facility at least at the same level. See *Ping facility* in the *IBM solidDB Administrator Guide* for details.	All
-t	Turns on the Network trace facility See *Network trace facility* in the *IBM solidDB Administrator Guide* for details.	All

- *host_computer_name* is needed with TCP/IP and Named Pipes protocols, if the client and server are running on different machines.
- *server_name* depends on the communication protocol:
  - In TCP/IP protocol, server_name is a service port number, such as '2315'.
  - In other protocols, server_name is a name, such as 'soliddb' or 'chicago_office'.

  For details on the syntax in different communication protocols, see *Communication protocols* in the *IBM solidDB Administrator Guide*.

Note:

- The *protocol_name* and the server_name must match the ones that the server is using in its network listening name.
- If given at the connection time, the connect string must be enclosed in double quotation marks.
- All components of the connect string are case insensitive.

sma_connect_string

The format of a connect string for SMA-based connections is the following:

```
sma protocol_name port_number | pipe_name
```

When SMA is used, applications on each node must be able to connect to the local server with a SMA connection and to the remote server with a network-based connection. This means that the list of server addresses takes the following format:

```
connect_target_list::= [SERVERS=]<sma_connect_string>, <network_connect_string>

cluster_info::= CLUSTER <sma_connect_string>, <network_connect_string>
```

For example:

```
sma tcp 2315, tcp 192.168.255.1 1315
```

The driver will scan the list from left to right and try to find the Primary and Secondary servers. Therefore, the preferable configuration must be put at the beginning of the list. The rest of the list may contain some spare addresses that might be activated at some point during the system lifetime. Keep the list short; in error situations, it can take a long time before the error is returned to the application. The addresses are tried one by one, involving the login timeouts specified (network-based connections). The number of addresses in the list is unlimited.

If none of the attributes TF_LEVEL nor PREFERRED_ACCESS is specified (or TF_LEVEL=NONE), the connection behavior falls back to Basic Connectivity. If more than one connect string is given, the connection is established to the first server on the list that accepts the connection request.

Configuring server addresses with multi-home servers

If your setup uses multi-home servers to deploy different networks for the connections between the application and the servers and the servers themselves, you need to define the server addresses for the TC connection with the **HotStandby.TCConnect** parameter.

From the application connection perspective, the address specified with the **HotStandby.TCConnect** parameter precedes the address defined with the **HotStandby.Connect** parameter. The TC connection will thus use the server addresses specified with **HotStandby.TCConnect** parameter, while the HotStandby connection between the servers uses the server addresses defined with the **HotStandby.Connect** parameter.

For an example of a multi-home server configuration, see “Example: TC connection with multi-home servers” on page 85.

CLUSTER

The CLUSTER keyword sets TF_LEVEL to SESSION and PREFERRED_ACCESS to READ_MOSTLY automatically.

For example, the following TC Info and the CLUSTER string are interchangeable:

```
TF_LEVEL=SESSION PREFERRED_ACCESS=READ_MOSTLY
SERVERS=tcp srv1.acme.com 1315, tcp srv2.acme.com 1315
```

```
CLUSTER=tcp srv1.acme.com 1315, tcp srv2.acme.com 1315
```

Tip: The cluster configuration can also be defined in the client-side `solid.ini` file, in which case the connect string in SQLConnect can use the logical name. For example:

```
rc = SQLConnect(comHandle, "Cluster1", ...
```

```
[Data Sources]
Cluster1=
   TF_LEVEL=SESSION
   PREFERRED_ACCESS=READ_MOSTLY
   SERVERS=
      tcp -c 1000 srv1.dom.acme.com 1315,
      tcp srv2.dom.acme.com 1315,
      tcp srv3.dom.acme.com 1316
```

Encryption attributes

The encryption attributes control whether the password for the connection is encrypted. The USE_ENCRYPTION keyword controls whether encryption is used. The USE_GSKIT controls whether IBM Global Security Kit (GSKit) is used for the encryption.

- If USE_ENCRYPTION=NO, encryption is disabled.
- If USE_ENCRYPTION=YES and USE_GSKIT=NO, the solidDB built-in DES algorithm is used for encryption.
- If USE_ENCRYPTION=YES and USE_GSKIT=YES, the GSKit is used for encryption.

SQLConnect examples

```
rc = SQLConnect(comHandle, "TF=CONNECTION
      USE_ENCRYPTION=YES PA=READ_MOSTLY
      SERVERS=
      tcp -c 1000 srv1.dom.acme.com 1315,
      tcp srv2.dom.acme.com 1315,
      tcp srv3.dom.acme.com 1316", ...
rc = SQLConnect(comHandle, "CLUSTER=
      tcp -c 1000 srv1.dom.acme.com 1315,
      tcp srv2.dom.acme.com 1315,
      tcp srv3.dom.acme.com 1316", ...
```

SMA setup: SQLConnect on the application on Node1 (srv1.dom.acme.com)

```
rc = SQLConnect(comHandle, "TF=CONNECTION
      PA=LOCAL_READ
      SERVERS=
      sma tcp 1315,
      tcp srv2.dom.acme.com 2315", ...
```

SMA setup: SQLConnect on the application on Node2 (srv2.dom.acme.com)

```
rc = SQLConnect(comHandle, "TF=CONNECTION
      PA=LOCAL_READ
      SERVERS=
      sma tcp 2315,
      tcp srv1.dom.acme.com 1315", ...
```

Client-side `solid.ini` examples

Note: For layout reasons, the Com.Connect parameter values in the following examples are split on several lines. In your `solid.ini` file, the entire parameter entry must be on one line.

```
[Com]
Connect = TF=CONNECTION USE_ENCRYPTION=YES
      PA=READ_MOSTLY
      SERVERS=
         tcp -c 1000 srv1.dom.acme.com 1315,
         tcp srv2.dom.acme.com 1315,
         tcp srv3.dom.acme.com 1316
```

SMA setup: solid.ini file on Node1 (srv1.dom.acme.com)

```
[Com]
Connect = TF=CONNECTION USE_ENCRYPTION=YES
      PA=LOCAL_READ
      SERVERS=
         sma tcp 1315,
         tcp srv2.dom.acme.com 2315
```

SMA setup: solid.ini file on Node2 (srv2.dom.acme.com)

```
[Com]
Connect = TF=CONNECTION USE_ENCRYPTION=YES
      PA=LOCAL_READ
      SERVERS=
         sma tcp 2315,
         tcp srv1.dom.acme.com 1315
```

Transparent connectivity with JDBC

With JDBC, transparent connectivity is enabled with non-standard connection properties. The list of server addresses is given as a part of the JDBC connect string.

Note: When using transparent connectivity in JDBC, you have to take care of dropping the statement objects explicitly. The garbage collector will not detect unreferenced statement objects.

Failure transparency level (solid_tf_level)

Failure transparency is enabled with the solid_tf_level connection property. The value is a string; you can specify it as a mnemonic (for example, NONE) or as a number (0 for NONE). For clarity, use of mnemonics is preferable.

Three levels are available:

1. NONE | 0 – failure transparency is disabled. This is the default value.
2. CONNECTION | 1 – the server connection is preserved, that is, it is unnecessary to reconnect in the case of failover or switchover.
3. SESSION | 3 – certain session attributes that have non-default values are preserved. Additionally, prepared statements are preserved. However, open cursors are closed, and ongoing transactions are aborted.

Preferred access attribute (solid_preferred_access)

The preferred access attribute indicates whether a read-only load is distributed or not. The preferred access attribute is enabled with the solid_preferred_access connection property. The value is a string; you can specify it as a mnemonic or as a number. For clarity, use of mnemonics is preferable.

The following levels are available:

- WRITE_MOSTLY | 0 – the read workload is directed to Primary. This is the default value. WRITE_MOSTLY also sets the connection to the WRITE MOSTLY mode. It is not possible to do that by specifying a numeric value.
- READ_MOSTLY | 1 – the read workload is directed to Secondary and Primary as defined by the **Cluster.ReadMostlyLoadPercentAtPrimary** parameter. The write transactions are handed over to the Primary.

  By default, the **Cluster.ReadMostlyLoadPercentAtPrimary** parameter is set to 50, which means that 50 percent of the read loads are directed to the Primary.

Reconnect timeout (solid_tf1_reconnect_timeout)

The solid_tf1_reconnect_timeout property specifies the connection reconnect timeout in milliseconds. The default value is 10 000 milliseconds (10 seconds).

Server addresses

The list of server addresses is given as a part of the extended JDBC connect string:

```
conStr= "jdbc:solid://host_name:port [,host_name:port].../user_name/password";
```

The number of addresses in the address list is limited to 20.

Configuring server addresses with multi-home servers

If your setup uses multi-home servers to deploy different networks for the connections between the application and the servers and the servers themselves, you need to define the server addresses for the TC connection with the **HotStandby.TCConnect** parameter.

From the application connection perspective, the address specified with the **HotStandby.TCConnect** parameter precedes the address defined with the **HotStandby.Connect** parameter. The TC connection will thus use the server addresses specified with **HotStandby.TCConnect** parameter, while the HotStandby connection between the servers uses the server addresses defined with the **HotStandby.Connect** parameter.

SMA connection (solid_shared_memory)

With SMA configurations, the non-standard property solid_shared_memory must be set to yes. You must also define that you are using a local server at a given port.

Example: TC connection with failure transparency level of CONNECTION

```
...
String conStr = "jdbc:solid://srv1.acme.com:1323,srv2-acme.com:1423/dba/dba";
Properties prop = new Properties();
prop.setProperty("solid_tf_level", "CONNECTION");
...
Connection c = DriverManager.getConnection(conStr, prop);
...
```

Example: TC connection with SMA

```
...
String conStr = "jdbc:solid://localhost:1323,srv2-acme.com:1423/dba/
dba?solid_shared_memory=yes";
Properties prop = new Properties();
prop.setProperty("solid_tf_level", "CONNECTION");
Connection c = DriverManager.getConnection(conStr, prop);
...
```

Example: TC connection with load balancing

```
...
String conStr = "jdbc:solid://12.345.67.88:1323,12.345.67.89:1423/dba/dba";
Properties prop = new Properties();
prop.setProperty("solid_tf_level", "CONNECTION");
```

```
prop.setProperty("solid_preferred_access", "READ_MOSTLY");
...
Connection c = DriverManager.getConnection(conStr, prop);
...
```

Example: JDBC URL defined TC connection with load balancing

```
jdbc:solid://12.345.67.88 1323, 12.345.67.89 1423/
dba/dba?solid_tf_level=1?solid_preferred_access=READ_MOSTLY
```

TC attribute combinations

The following table summarizes the possible combinations of the TC attributes and presents the resulting connection capabilities:

Table 14. Possible combinations of TC Info attributes

PREFERRED_ ACCESS:	TF_LEVEL: Not specified or NONE	TF_LEVEL: CONNECTION	TF_LEVEL: SESSION
Not specified	• No failover or switchover support • No load balancing (Basic connectivity)	• Transparent failover (session state not preserved) • Transparent switchover • Workload in Primary only • No load balancing	• Transparent failover (session state preserved) • Transparent switchover • Workload in Primary only • No load balancing
WRITE_MOSTLY (default)	• No transparent failover support • Transparent switchover • Workload in Primary only • No load balancing	• Transparent failover (session state not preserved) • Transparent switchover • Workload in Primary only • No load balancing	• Transparent failover (session state preserved) • Transparent switchover • Workload in Primary only • No load balancing
READ_MOSTLY	• No transparent failover support • Transparent switchover • Workload in Secondary and Primary • Load balancing	• Transparent failover (session state not preserved) • Transparent switchover • Workload in Secondary and Primary • Load balancing	• Transparent failover (session state preserved) • Transparent switchover • Workload in Secondary and Primary • Load balancing

Table 14. Possible combinations of TC Info attributes (continued)

PREFERRED_ ACCESS:	TF_LEVEL: Not specified or NONE	TF_LEVEL: CONNECTION	TF_LEVEL: SESSION
LOCAL_READ	• No transparent failover support • Transparent switchover • Primary reads and writes executed locally using SMA connection • Secondary reads executed locally using SMA connection, writes on Primary using network connection	• Transparent failover (session state not preserved) • Transparent switchover • Primary reads and writes executed locally using SMA connection • Secondary reads executed locally using SMA connection, writes on Primary using network connection	• Transparent failover (session state preserved) • Transparent switchover • Primary reads and writes executed locally using SMA connection • Secondary reads executed locally using SMA connection, writes on Primary using network connection

Connect error processing

When a connect request is issued for a TC Connection, it is considered successful if at least one applicable server is found and connected to.

The server may be in one of the states: PRIMARY ACTIVE, PRIMARY ALONE, or STANDALONE. Otherwise, the connect effort is considered failed. The address list is scanned once.

There may be various reasons for the connect request to fail. Most of them are masked by the following error cases:

Table 15. Connect request errors

SQLSTATE	Native code	Message text and description
08001	25217	`Client unable to establish a connection` Description: The driver has used the TC connect info to find an applicable server and connect to it. The effort has failed due to one of he following reasons: • No host listed in the address list was found • A host was found but the login timed out • A host was found but the login was rejected • Hosts found but not in the PRIMARY/ STANDALONE state

Table 15. Connect request errors (continued)

SQLSTATE	Native code	Message text and description
HY000	21307	`Invalid connect info...` Description: a syntax error is found in an elementary connect string or in the TC connect info (data source info).
HY000	21300	`Protocol ... not supported.` Description: the string "TC" in the beginning of the TC connect info is misspelled (or, an incorrect protocol name is given in the elementary connect string).

There are cases when the connection is accepted with a warning.

Table 16. Warnings

SQLSTATE	Native code	Message text and description
0100	25218	`Connected to Standalone or Primary Alone server.` An effort has been made to set any non-default value of TF_LEVEL or PREFERRED_ACCESS, and there is only one server available. On this case, neither failure transparency nor load balancing is available.

Example: TC connection

This example shows a Transparent Connectivity configuration for a typical HotStandby setup where the application and the servers are within the same network.

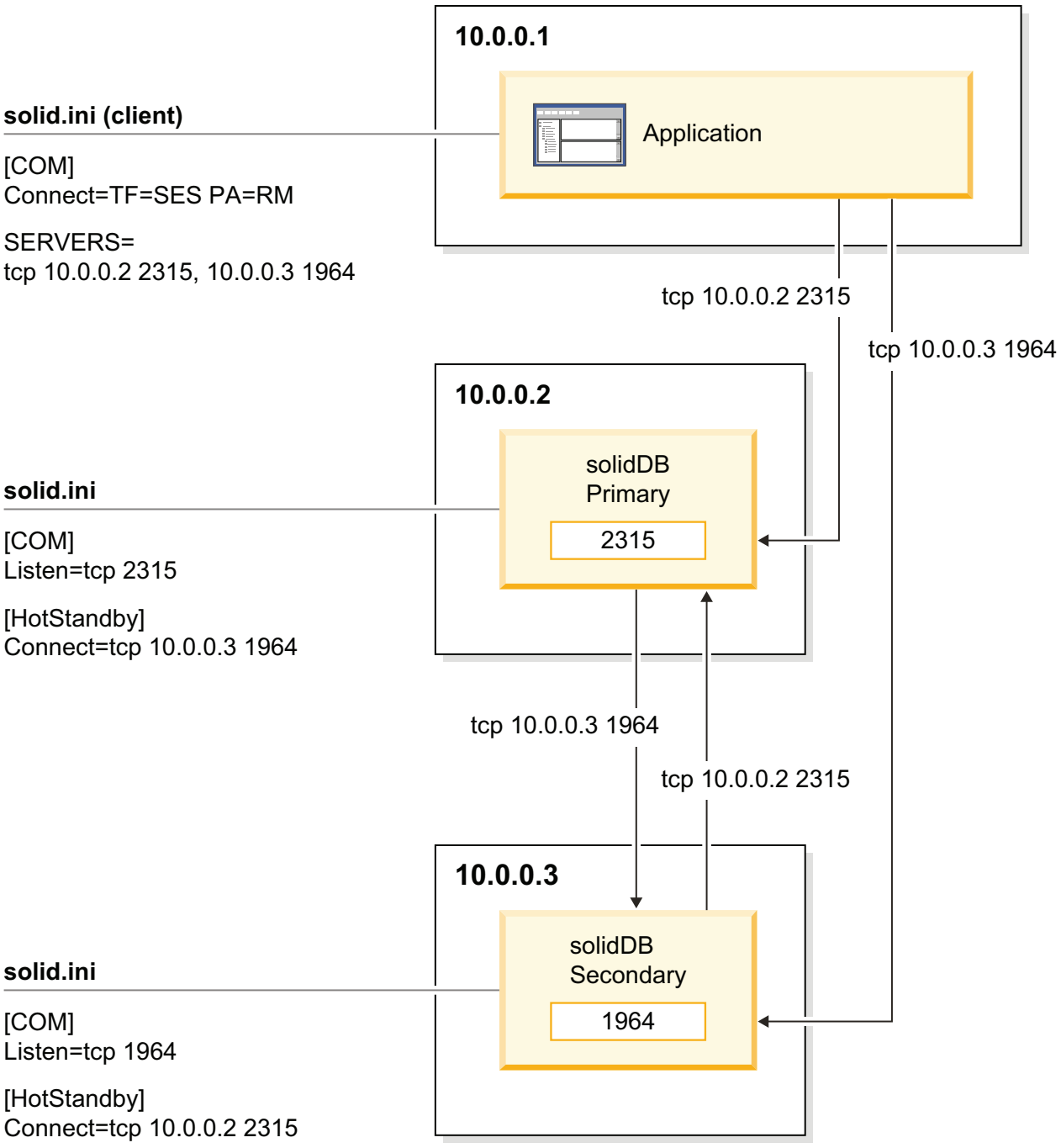


Figure 14. Example: TC connection

Example: TC connection with multi-home servers

This example shows a Transparent Connectivity configuration for a HotStandby setup where the Primary and Secondary are multi-home servers. It is typical in setups where the application and the servers are in different networks or the application cannot or should not connect to the servers using the same network that is used for the HotStandby connection between the servers.

To differentiate between the TC connection and the HotStandby connection, you need to use two parameters that define the server addresses:

- The **HotStandby.Connect** parameters specify the server addresses that are used for the HotStandby connection between the primary and secondary servers.
- The **HotStandby.TCConnect** parameters specify the server addresses that are used with the TC connection.

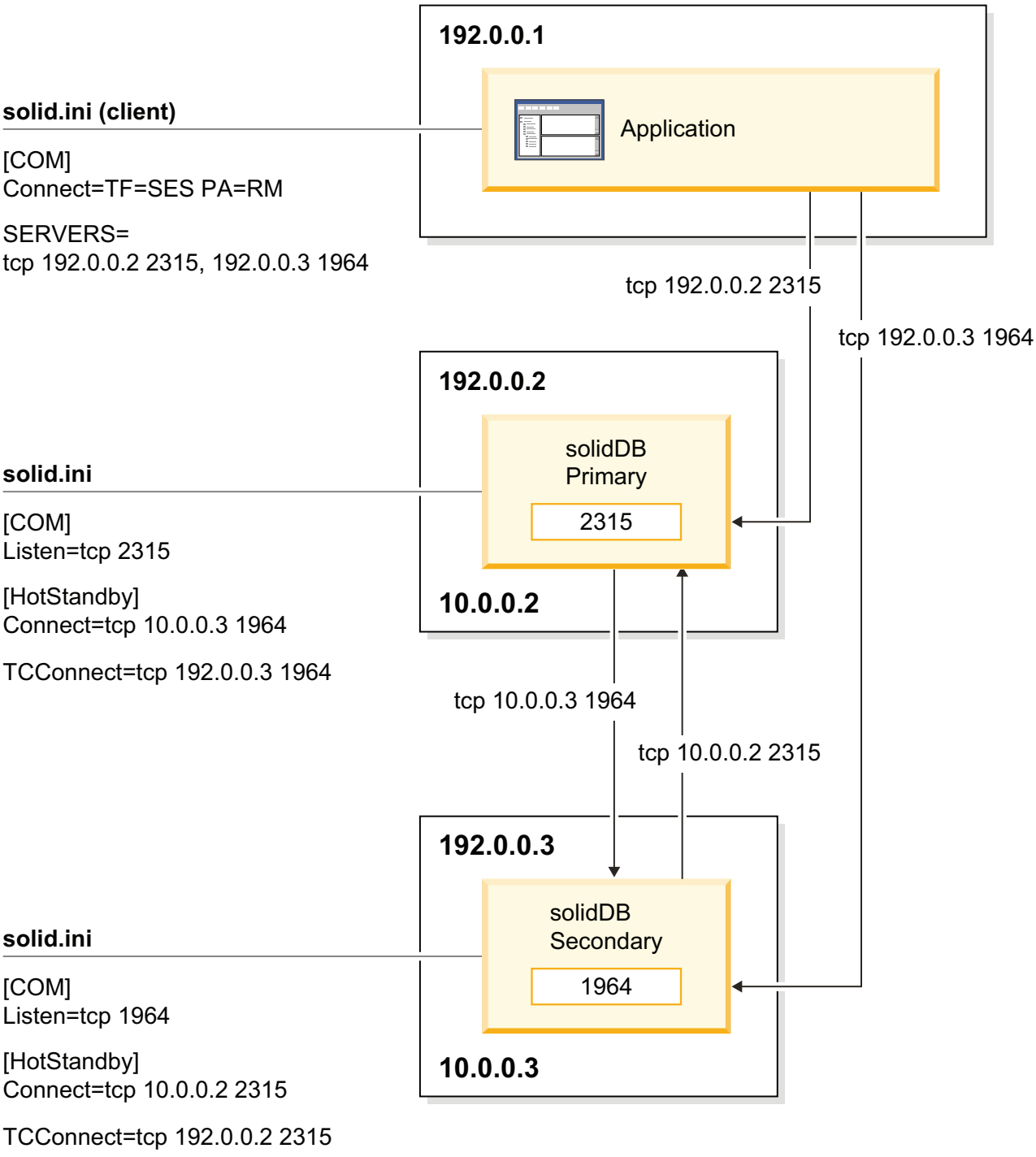


Figure 15. Example: TC connection with multi-home servers

4.2.2 Failure transparency in Transparent Connectivity

Failure transparency handles the masking of failures. The failure transparency level is set with the TF_LEVEL attribute.

Three failure transparency levels are available:

TF_LEVEL=NONE
Failure transparency is disabled (default).

TF_LEVEL=CONNECTION
The server connection is preserved, that is, it is unnecessary to reconnect in the case of failover or switchover.

TF_LEVEL=SESSION
Most nondefault session level settings are preserved. Additionally, prepared statements are preserved. However, open cursors are closed, and ongoing transactions are aborted.

Principles of connection switch handling

A connection switch refers to a situation where the driver changes the active server connection. Generally, the reason for a connection switch is a failover to the Secondary server or a switchover between the servers.

More specifically, a need for a connection switch is detected from one of the following events:

- Event from the Secondary server about the state change to PRIMARY ALONE (failover) or PRIMARY ACTIVE (switchover). This is the main (and the fastest) mode of performing the connection switch.
- Indication of the state change at Primary.
- Link failure on the active connection.
- Connection timeout on the active connection.

The driver executes the connection switch in two steps:

1. The need for the connection switch is detected. The driver returns the following connection switch error on a pending request, or the following request:

Table 17. Connection switch request

SQLSTATE	Native code	Message text and description
HY000	25216	`Connection switch, some session context may be lost` Description: The driver has discovered the need of the connection switch. The client is expected to issue a transaction rollback call to finalize the connection switch. This error code and message will be received at each consecutive network request call until the rollback call is issued

2. The Client program issues a rollback command (ODBC: SQLEndTran() with SQL_ROLLBACK; JDBC: Connection.rollback()). If the rollback is successful, a new active connection has been mapped to the TC connection that may be used.

   Note: The connection switch error may be returned on a few consecutive ODBC calls. Therefore, a provision must be made to always respond with a rollback to this error, on any ODBC network request. If this happens in the middle of a transaction, the transaction must be re-executed.

   On the other hand, if a new active connection cannot be established, the following error code is returned:

Table 18. Communication link failure

SQLSTATE	Native code	Message text and description
08S01	14503	`Communication link failure` Description: The driver has failed to establish a new active connection. The TF connection is lost and the Client has to reconnect (using a Data Source Info) in order to continue.

After receiving the rollback call, the driver will use a few alternative ways of finding the new active connection. In the simplest case, it will use the standby connection for the purpose. If that connection is not in the right state, the driver will wait for two seconds for the proper event to arrive. If the event does not arrive, and in other cases, the driver will fall back to the address list in the TC connect info and will repeat the connect sequence iteratively for a maximum time of 10 seconds. If all the efforts fail, the driver returns the above error.

The effect of the error is that the connection is lost, as seen by the application. Any further request issued on that connection will result in the same error.

Preservation of session state

When the connection switch is executed by the driver, some of the session context can be lost and the application must reconstruct it. The amount of the preserved state is dictated by the Transparent Failover level, expressed with the TC Info attribute TF_LEVEL.

With the TF level CONNECTION, no state is preserved. With the SESSION level, most of the session state is preserved. The preservation of the session state is implemented by caching the necessary data in the driver. The higher transparency level is achieved at the expense of the response time of the requests requiring caching, as well as increased memory usage in the driver.

Regardless of the TF level, the following applies in the case of failovers:

- The updates of the current transactions are lost (because of the transaction rollback).
- Open cursors and their positions are lost.

Table 19. Session state preservation

TF_LEVEL	Preserved state
CONNECTION	No session state is preserved.
SESSION	Prepared statements • The prepared states are preserved. The effects of the following statements are preserved: • SET CATALOG • SET SQL INFO • SET SQL SORTARRAYSIZE • SET SQL CONVERTORSTOUNIONS • SET SQL JOINPATHSPAN • SET LOCK TIMEOUT *<seconds>* • SET IDLE TIMEOUT • SET OPTIMISTIC LOCK TIMEOUT • SET STATEMENT MAXTIME • SET ISOLATION LEVEL • SET DURABILITY • SET SAFENESS • SET SCHEMA • SET SYNC USER • SET SYNC MODE The following standard ODBC attributes are preserved • SQL_ATTR_ACCESS_MODE (SET READ ONLY, SET READ WRITE) • SQL_ATTR_CURRENT_CATALOG (duplicates SET CATALOG above) • SQL_ATTR_AUTOCOMMIT

4.2.3 Load balancing in Transparent Connectivity

With TC, the driver uses two methods to direct the transaction load; one to handle read intensive load and the other to handle write intensive load. For load balancing, the logical TC Connection is mapped to a lower level server connection called *Workload Connection*. The workload connection can change over time and it is normally of no concern to the application. However, if necessary, there is a way to find out what is the current workload connection.

Static load balancing configuration

The load balancing methods are:

PREFERRED_ACCESS=WRITE_MOSTLY – no load balancing (default)
With WRITE_MOSTLY, all transactions are executed at the Primary server. This corresponds to the typical HotStandby operation. WRITE_MOSTLY method is useful with write-intensive loads.

PREFERRED_ACCESS=READ_MOSTLY – load balancing by distributing read-only transactions between Primary and Secondary

With READ_MOSTLY, read-only transactions can be executed at both the Secondary and Primary. Write transactions are always executed at the Primary server.

For read-only transactions, the *assigned workload server* is selected on the basis of the **Cluster.ReadMostlyLoadPercentAtPrimary** parameter setting, which specifies the percentage of the total read-mostly load that is directed to Primary.

The default value of the **Cluster.ReadMostlyLoadPercentAtPrimary** parameter is 50, which means that by default, half of the connections use the Primary and half the Secondary. This is a preferable value for most mixed loads. If the value is set to zero, all the load is directed at the Secondary. This is suitable in cases where very read-intensive (or read-only) applications use PREFERRED_ACCESS=READ_MOSTLY and (in the same time) write-intensive applications use PREFERRED_ACCESS=WRITE_MOSTLY.

With READ_MOSTLY, the Primary server tells the driver which server to connect to for the workload connection. If the load for a given connection is directed to Secondary, and a write operation is issued, a handover to Primary takes place and the transaction is executed in the Primary server. After the transaction commit, the load is directed back to Secondary. If Secondary fails, the connection fails over from Secondary to Primary.

PREFERRED_ACCESS=LOCAL_READ – load balancing by executing transactions locally when possible

With LOCAL_READ, read-only transactions are always directed to the local server, be it Primary or Secondary. Write transactions are always executed at the Primary server.

The LOCAL_READ method is typically used with SMA setups. The application on the Primary server uses the SMA connection for read and write transactions. The application on the Secondary uses SMA connection for reads and a network-based connection for writes on the Primary.

Important: When using SMA with Transparent Connectivity (TC), if you set the load balancing method to READ_MOSTLY or WRITE_MOSTLY (default), a network connection is used instead of the SMA connection. Thus, when using SMA with TC, always set the load balancing method to LOCAL_READ.

Isolation levels and load balancing

Load balancing operates only at the isolation level READ COMMITTED. If the server's isolation level (startup) default is set to a different value, the settings PREFERRED_ACCESS=READ_MOSTLY and PREFERRED_ACCESS=LOCAL_READ force the isolation level of this session to READ COMMITTED. The isolation level may be dynamically reset to a higher one (for example, REPEATABLE READ), but then the load balancing is disabled.

Autocommit and load balancing

To use load balancing, autocommit for the session must be disabled.

Controlling load balancing dynamically

When using load balancing (READ_MOSTLY or LOCAL_READ), you can change the assigned workload server from Secondary to Primary programmatically.

At the session level, the workload connection server can be changed to Primary with the following statements:

- `SET WRITE`
- `SET ISOLATION LEVEL REPEATABLE READ`
- `SET ISOLATION LEVEL SERIALIZABLE`

The statement takes effect immediately, if it is the first statement of a transaction, or from the next transaction, otherwise.

At the transaction level, the following statements change the workload connection server to Primary for the time of one transaction:

- `SET TRANSACTION WRITE`
- `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ`
- `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE`

The affected transaction is the one that is started by using the statement, or the next one, in other cases. After the transaction has been executed at the Primary, the workload connection server is reverted to the one assigned for the session.

The effect of the SET [TRANSACTION] WRITE statement may be reverted with the statement SET [TRANSACTION] READ WRITE. Also, the following isolation level statements have the same effect:

- `SET ISOLATION LEVEL READ COMMITTED`
- `SET TRANSACTION ISOLATION LEVEL READ COMMITTED`

Note: With READ_MOSTLY, the proportional distribution of the read loads between the Primary and Secondary is defined with the **`Cluster.ReadMostlyLoadPercentAtPrimary`** parameter. By default, the value is set to `50`, which means that 50 percent of the read loads are directed to the Primary. Changing the value of the **`Cluster.ReadMostlyLoadPercentAtPrimary`** parameter requires a server restart.

Related information:

A.1.1, “Cluster section,” on page 118

Failover transparency with load balancing

When both failure transparency is set (**`TF_LEVEL`** is other than `NONE`) and load balancing is enabled (**`PREFERRED_ACCESS`**=`READ_MOSTLY` or **`PREFERRED_ACCESS`**=`LOCAL_READ`), the applied failover policy is the following:

1. Primary failure: all the load is directed to the new Primary being in the PRIMARY ALONE state.
2. Secondary failure: all the load is directed to the Primary (PRIMARY ALONE)
3. Connection break between the servers; the servers are in the PRIMARY ALONE and SECONDARY ALONE states: if there is an ongoing read-only transaction executing in the Secondary, it is also successfully committed in the Secondary. All the subsequent transactions are directed to the Primary (in PRIMARY ALONE).

When the normal hotstandby operation is resumed (with servers being in PRIMARY ACTIVE and SECONDARY ACTIVE states), the load is rebalanced between the Primary and the Secondary.

Note: Even when failure transparency is not enabled (TF_LEVEL=NONE), some rudimentary failover capability is available: failover from Secondary to Primary

when the Secondary fails. All other failures result in a communication link failure. Thus, in most failure cases where TF_LEVEL=NONE, the application must reconnect with the same TC Info. To avoid reconnection, enable failure transparency when load balancing is used.

Executing stored procedures under load balancing

All SQL stored procedures are executed in the Primary unless they are specified as read-only procedures with the SQL standard clause *SQL_data_access_indication* in the procedure declaration.

```
<SQL_data_access_indication> ::=
     NO SQL |
     READS SQL DATA |
     CONTAINS SQL |
     MODIFIES SQL DATA
```

Only the keyword MODIFIES SQL DATA inflicts transaction handover. This is the default behavior.

To avoid unnecessary handovers of read-only procedures and functions, use one of the following values:

- NO SQL
- READS SQL DATA
- CONTAINS SQL

4.2.4 Handling TC Info contradictions

The attributes of the TC Info may contradict the actual service made available. In those situations, the connection is granted, but the SUCCESS_WITH_INFO warning is issued.

This is done in the following cases:

- PREFERRED_ACCESS is specified, but HSB is not enabled. Basic connectivity is enabled.
- TF_LEVEL is specified, but HSB is not enabled. Basic connectivity is enabled.

4.3 Basic Connectivity

With Basic Connectivity, the application has to take care of connecting to each server of the HotStandby or Cluster configuration separately, by using specific server addresses. If a failover happens, the active connection is lost, and the application has to reconnect to the new Primary server.

Basic Connectivity syntax

The Basic Connectivity uses the standard solidDB connect string syntax:

protocol_name [*options*] [*host_computer_name*] *server_name*

where

- *options* can be any combination of the following:

Table 20. Connect string options

Option	Description	Protocol
-4	Specifies that client connects using IPv4 protocol only.	TCP/IP

Table 20. Connect string options (continued)

Option	Description	Protocol
-6	Specifies that client connects using IPv6 protocol only. In Windows environments, this option is mandatory if IPv6 protocol is used.	TCP/IP
-i*source_address*	Specifies an explicit connecting socket source address for cases where the system default source IP address binding does not meet application needs. *source_address* can be an IP address or a host name.	TCP/IP
-z	Enables data compression for the connection **Important:** • Data compression is not available for HotStandby connections (**HotStandby.Connect**) and NetBackup connections (**ADMIN COMMAND 'netbackup'**). • Data compression for **netcopy** connections cannot be enabled with the **-z** option. Instead, use the **HotStandby.NetcopyRpcCompress=yes** parameter setting.	All
-c *milliseconds*	Specifies the login timeout (the default is operating-system-specific). A login request fails after the specified time has elapsed.	TCP/IP
-r *milliseconds*	Specifies the connection (or read) timeout. A network request fails when no response is received during the time specified. The value 0 (default) sets the timeout to infinite (operating system default timeout applies).	TCP/IP
-o*filename*	Turns on the Network trace facility and defines the name of the trace output file See *Network trace facility* in the *IBM solidDB Administrator Guide* for details.	All
-p*level*	Pings the server at the given level (0-5). Clients can always use the solidDB Ping facility at level 1 (0 is no operation/default). Levels 2, 3, 4 or 5 may only be used if the server is set to use the Ping facility at least at the same level. See *Ping facility* in the *IBM solidDB Administrator Guide* for details.	All
-t	Turns on the Network trace facility See *Network trace facility* in the *IBM solidDB Administrator Guide* for details.	All

- *host_computer_name* is needed with TCP/IP and Named Pipes protocols, if the client and server are running on different machines.
- *server_name* depends on the communication protocol:
  - In TCP/IP protocol, server_name is a service port number, such as '2315'.
  - In other protocols, server_name is a name, such as 'soliddb' or 'chicago_office'.

  For details on the syntax in different communication protocols, see *Communication protocols* in the *IBM solidDB Administrator Guide*.

Note:

- The *protocol_name* and the server_name must match the ones that the server is using in its network listening name.
- If given at the connection time, the connect string must be enclosed in double quotation marks.
- All components of the connect string are case insensitive.

For example:

```
Connect=tcp srv1.dom.acme.com 1315
```

4.3.1 Reconnecting to primary servers from applications

Preparing client applications for HotStandby

Client programs that have lost their connection to the Primary must be able to reconnect to the new Primary server (the old Secondary). You must code client applications to be able to:

1. Recognize that Primary is not available for write transactions any more.
2. Connect to the other server or switch to using previously created connection.
3. Take into account whether the current (interrupted) transaction was lost/aborted and must be re-executed on the new Primary server.

Getting the secondary server address

The easiest way to get the connection information for the Secondary database server is to use the **ADMIN COMMAND 'hotstandby cominfo'** command, which gives the connection information for the other server in the HSB pair.

About this task

Procedure

1. When your application first connects to Primary, the application can execute the **ADMIN COMMAND 'hotstandby cominfo'** command and store the result. Note that when the **cominfo** command returns a value, it does NOT imply that Primary and Secondary are currently connected. The "cominfo" command simply returns the value specified in the **Connect** parameter of the `solid.ini` configuration file, or the value most recently specified with the **hsb parameter connect** command. If you need to check the connect status between Primary and Secondary servers, you can use **ADMIN COMMAND 'hotstandby status connect'**.
2. Later, if Primary fails, the application can use the stored information to connect to Secondary (new Primary).

Detecting HotStandby server failure in client applications

To use the HotStandby (HSB) component, applications must know when to switch from the failed Primary to the Secondary (new Primary) server. There are a couple of possible ways to do this. The best way is to simply check the return codes from the functions that you call to see if you have received an error that indicates you should switch to the other server.

You may also monitor the states of the servers (for example, check the Primary server to see whether its state has changed to PRIMARY UNCERTAIN).

The errors that indicate you should try switching to another server include:

- 10013: Transaction is read only
- 10041: Database is read only
- 10047: Replication transaction aborted
- 11002: Disk full
- 11003: File write failed, configuration exceeded
- 14501: Operation failed
- 14502: Invalid rpc parameter
- 14503: Communication error
- 14506: Server is closed (for example, because it is currently the target of an HSB netcopy operation)
- 14510: Communication write operation failed

- 14511: Communication read operation failed
- 14518: Connection broken
- 14519: User thrown out (for example, because of some administrative operation)
- 14529: Operation timed out
- 20009: Session error, write operation failed
- 21306: Server not found, connect failed
- 21308: Connection is broken (write failed with code ...)
- 21318: Operation failed (unusual return code)

ODBC applications

The following error message is returned to ODBC applications that cannot establish a connection (for example, due to an inoperable database server):

- SQLState = 08001 - Client unable to establish connection

In addition, the following solidDB communication error message is produced:

- 21306 - Server *'server_name'* not found, connection failed.

If a connection fails (for example, due to a network failure) in between operations, such as executing queries and fetching results, the following error message is returned:

- SQLState = 08S01 - Communication link failure

JDBC applications

The following error message is returned to JDBC applications that cannot establish a connection (for example, due to an inoperable database):

- SQLState = 08001 - Unable to connect to data source.

If a connection fails (for example, due to a network failure) in between operations, such as executing queries and fetching results, the following error message is returned:

- SQLState = 08S01 - Communication link failure

Note:

ODBC and JDBC use different error messages for the same error code (08001).

Switching the application to the new primary

After the application detects that it cannot send transactions to the "old Primary" server, the application must poll the old Primary and old Secondary servers until it finds a server that is in PRIMARY ACTIVE, PRIMARY ALONE, or STANDALONE state.

Polling is accomplished by having the application attempt to connect to the servers and check the status of the servers when the connection is established. When the connect is successful, the client can request the server state by using SQL function HOTSTANDBY_STATE, which is described in section “Using the HOTSTANDBY_STATE function” on page 96.

CAUTION:
After the switch, all open database objects, such as prepared statements, open cursors and transactions, are no longer active. Thus, you must initialize these objects again. Also, if you were using Temporary Tables or Transient Tables (solidDB main memory engine features), the tables will be empty on the new Primary.

Using the HOTSTANDBY_CONNECTSTATUS function

To verify connect status information when reconnecting to a Primary server from an application, you can use the HOTSTANDBY_CONNECTSTATUS function. This function is equivalent to the administrative command **`hotstandby status connect`**.

The function has no arguments and returns one of the following status values:

Table 21. HOTSTANDBY_CONNECTSTATUS status values

Status	Description
CONNECTED	The connection is active. This status is returned from both the Primary and Secondary servers.
CONNECTING	The Primary server is connecting to the Secondary server. This status is returned from both the Primary and Secondary servers.
CATCHUP	The Primary server is connected to the Secondary server, but the transaction log is not yet fully copied. This status is returned from both the Primary and Secondary server.
BROKEN	The connection is broken. This status is returned from both the Primary and Secondary servers.

Using the HOTSTANDBY_STATE function

To implement application polling of the Primary and Secondary servers, you can use the HOTSTANDBY_STATE function. This function is equivalent to the administrative command **`hotstandby state`**. It allows the application request the current HotStandby state when it is connected to the server.

Note: This function has no arguments. For a description of each possible state that this function may return, see 3.4.8, “Verifying HotStandby server states,” on page 62.

Sample pseudo-code

An application, whether or not it is HSB-enabled, should have error handling that allows the application to replay a failed/aborted transaction.

In a non-HSB environment, a transaction may be aborted because of a concurrency conflict (optimistic tables) or deadlock (pessimistic tables). The application must catch these error situations and either automatically retry the transaction or ask interactive user to re-execute the transaction.

If your application already has code to handle failed or aborted transactions, then it is relatively easy to extend this code to make use of HSB.

In a very simplified example, the application pseudo-code with proper error handling for a non-HA-aware application handling looks something like this:

```
BEGIN TRANSACTION
EXECUTE APPLICATION LOGIC
PREPARE & EXECUTE STATEMENTS
COMMIT TRANSACTION
IF ERROR OCCURRED
      IF ERROR == concurrency conflict or deadlock
            GO TO BEGIN TRANSACTION
      END IF
      other error handling
END IF ;
```

Improving the above application to make it HA-aware is very simple. You must add code so that the application can:

- Connect to either of the two servers instead of only one; and
- In the case of an error, find the server that is currently in one of the following states: PRIMARY ACTIVE, PRIMARY ALONE or STANDALONE.

The pseudo-code should look similar to the following:

```
BEGIN TRANSACTION
EXECUTE APPLICATION LOGIC
PREPARE & EXECUTE STATEMENTS
COMMIT TRANSACTION
IF ERROR OCCURRED
      IF ERROR == server unavailable for write transactions
            FIND CURRENT PRIMARY SERVER
            GO TO BEGIN TRANSACTION
      END IF
      IF ERROR == concurrency conflict or deadlock
            GO TO BEGIN TRANSACTION
      END IF
      IF ERROR == something else
            other error handling
      END IF
END IF
```

The logic to find the current primary server is also very simple. Just check the current state of both servers (try to reconnect if necessary) and if either of them is PRIMARY ACTIVE, PRIMARY ALONE or STANDALONE, set that server as the current primary. If neither server meets that criterion, wait awhile and retry checking the current server states.

4.3.2 Reconnecting to secondary servers

In some cases, you may want to connect to the current Secondary (if it is up). Applications can submit read-only queries to the Secondary server; this can sometimes help you balance the workload across your servers.

An application can only connect to Secondary databases in the read-only mode. Note that a client can connect to the Secondary server (only in read-only mode) by using the following parameter values in the HotStandby section of the `solid.ini` configuration file in these servers:

- **`Connect`** parameter in the Primary server
- **`Listen`** parameter in the Secondary server

You can also use the following command to get the connection information for a server's partner:

```
ADMIN COMMAND 'hotstandby cominfo';
```

Thus, if you are connected to the current Primary server, you can get the address of the current Secondary by using the cominfo query.

4.4 Defining timeouts between applications and servers

This section describes how to configure application read timeout and connect timeout settings by using either the `solid.ini` **`Connect`** parameter or the connect string of the SQLConnect function for ODBC.

These timeout values apply to the server's connections with client applications, including solidDB SQL Editor (**`solsql`**), solidDB Remote Control (**`solcon`**), and HA Manager.

4.4.1 Application read timeout option

The application read timeout option helps detect failures in low level network RPC read operations. The timeout setting applies to the read in the physical network (TCP/IP protocol only).

This RPC read timeout (called connection timeout in ODBC and JDBC) can be configured (in milliseconds) in the following ways. The default timeout is 0 – infinite (operating system default timeout applies),

- Client-side **`Com.ClientReadTimeout`** parameter

  For example:

  ```
  [Com]
  ;Set RPC read timeout to 1000 milliseconds (one second)
  ClientReadTimeout=1000
  ```

- Client-side **`Com.Connect`** parameter with option **`-r`***milliseconds*

  For example:

  ```
  [Com]
  ;Set RPC read timeout to 1000 milliseconds (one second)
  Connect=TCP -r1000 1313
  ```

  Note: For client applications such as the watchdog, it is convenient to provide the RPC read timeout (called also connection timeout) in the **`Com.Connect`** parameter using the **`-r`** option. Otherwise certain network failure types may cause indefinite waits.

  Note:

  The **`Connect`** parameters in the `[Com]` section, `[Watchdog]` section, and `[Hotstandby]` section are for different purposes. Make sure that you edit the correct one.

- Connect string of the SQLConnect function (**`-r`** option)

  For example:

  ```
  SQLConnect (hdbc, "TCP -r1000 1313", SQL_NTS,
  "dba", SQL_NTS, "dba", SQL_NTS);
  ```

  In the example above, the constant SQL_NTS indicates that the previous string (servername, username, or password) was passed as a standard Null-Terminated String.

4.4.2 Specifying -C option in the connect parameters

You can specify the connect timeout (called also login timeout) value in the **Connect** parameter used in the [Com] and [Watchdog] sections of the solid.ini file. This connect timeout works only for the TCP/IP protocol.

The syntax is:

```
Parameter = tcp -cnumber-of-milliseconds [machine name] port_number
```

where *Parameter* is **Connect** or **Listen**.

If no value is provided for the connect timeout, the server uses the operating system-specific default value.

Note:

For client applications, such as the watchdog, it is convenient to provide the connect timeout value in the Connect parameter using the **-c** option. Otherwise certain network failure types may cause a long wait before the failure is detected.

For example:

Application node:

```
[Com] ;The server listens to port 1320, and the Connection timeout is 1000 ms.
Listen = tcpip -c1000 1320
```

4.5 Configuring SMA with HotStandby

When using SMA with Transparent Connectivity (TC), the applications on the Primary and the Secondary must connect to the databases using a SMA-specific TC connect info syntax.

About this task

With SMA TC, the application on each node must be able to connect to the local server with a SMA connection and to the remote server with a network-based connection.

The format of the TC connect target list for SMA with HotStandby is the following:

```
connect_target_list::=[SERVERS:]sma_connect_string, network_connect_string
```

where

```
sma_connect_string::= sma protocol_name port_number | pipe_name

network_connect_string::= protocol_name IP_address | host_computer_name
                                    port_number | pipe_name
```

Additionally, you need to set the load balancing method to LOCAL_READ (PREFERRED_ACCESS=LOCAL_READ).

Important: When using SMA with TC, if you set the load balancing method to READ_MOSTLY or WRITE_MOSTLY (default), a network connection is used instead of the SMA connection. Thus, when using SMA with TC, always set the load balancing method to LOCAL_READ.

Procedure

1. Set up the two HotStandby servers.
2. Set up SMA on both servers.
3. For both applications, define the TC connection using the SMA-specific connect target list syntax and the load balancing attribute `PREFERRED_ACCESS=LOCAL_READ`.
4. Compile and start the applications.

Example

Connect info of the application on `host1` where solidDB is listening at port 1964:

```
PREFERRED_ACCESS=LOCAL_READ SERVERS=sma tcp 1964, tcp host2 2315
```

Connect string of the application on `host2` where solidDB is listening at port 2315:

```
PREFERRED_ACCESS=LOCAL_READ SERVERS=sma tcp 2315, tcp host1 1964
```

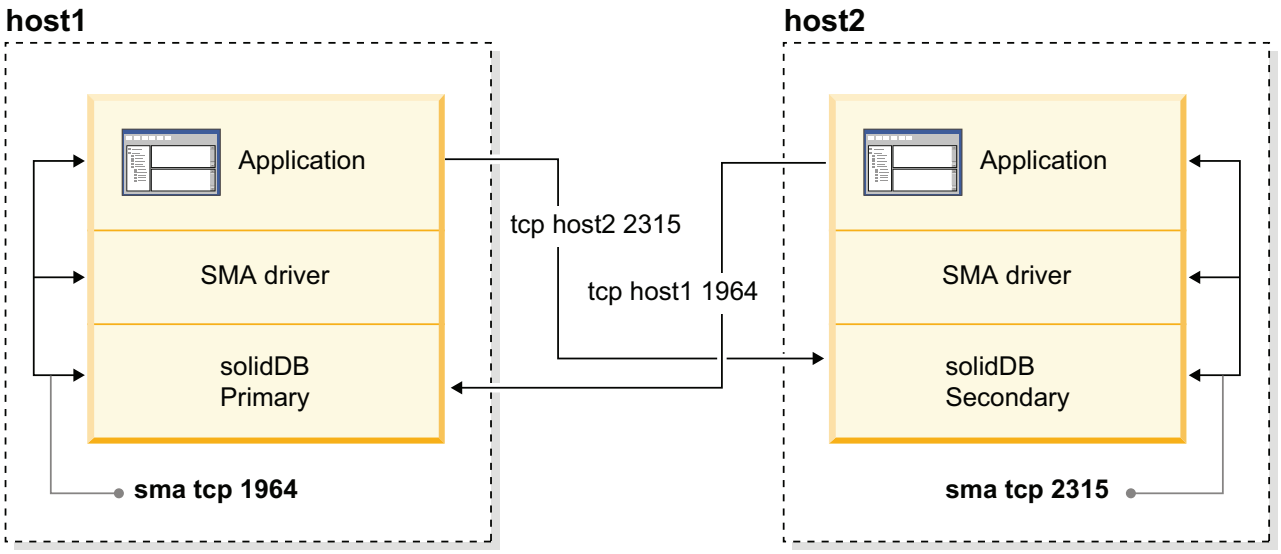


Figure 16. Example: HotStandby with SMA configuration

4.6 Configuring advanced replication with HotStandby

Any node of a advanced replication system can be made highly available with the solidDB HotStandby component.

When the master and replica databases of a advanced replication system are synchronizing data, the synchronization occurs between the Primary servers of the database server pairs. In other words, the Primary of the Master communicates with the Primary of the Replica. See Figure 7 on page 18.

A database server may fail over to its Secondary server at any point of time, including when the database server is synchronizing data with another server using advanced replication. If the failover occurs during synchronization, executing the synchronization message stops and the process must be resumed after the failover. For details about how to resume synchronization after an error has occurred, refer to the *IBM solidDB Advanced Replication User Guide*.

If a server containing a master database is made fault tolerant with solidDB HotStandby, the replicas of the master database must know the connect strings to both master servers. To do this, execute the following statement in each of the replica databases:

```
SET SYNC CONNECT 'connect_string_to_server_1, connect_string_to_server_2'
TO MASTER master_nodename
```

In the diagram below, the gray arrows represent the original connections to the original Primary server, while the black arrows represent the new connections to the new Primary (old Secondary) server. The alternate connection is used if the synchronization with the old Primary server fails.

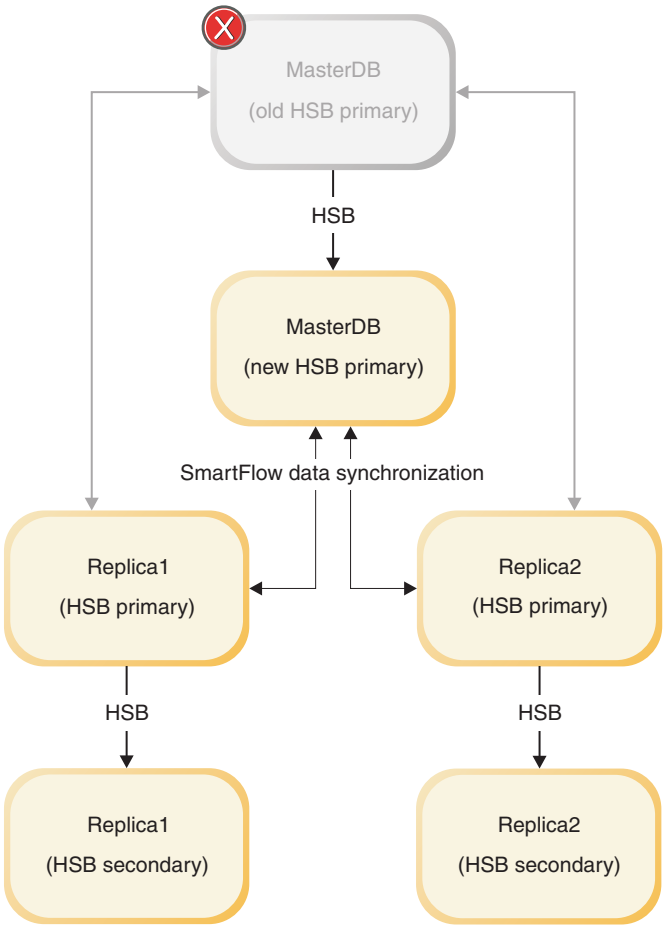


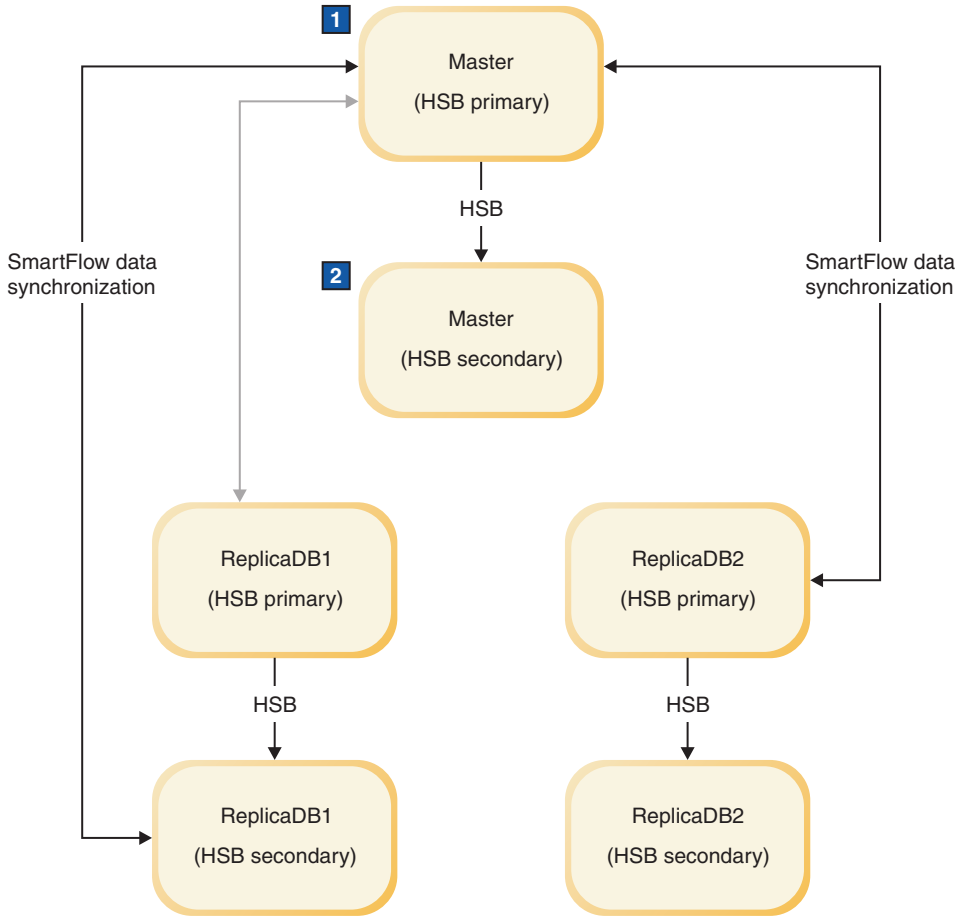
Figure 17. HotStandby and advanced replication: failover of Master database

If the server using solidDB HotStandby is a server containing a replica database and if the master server uses remote procedure calls (CALL *procedure_name* AT *node_name*) to run procedures at the replica, for example to initiate the synchronization, the master server must be informed about the connect strings to both of the replica servers. Typically a master server uses remote procedure calls to initiate synchronization with a replica database. To inform the master about the connect strings to the replica server pair, execute the following statement in the master database:

```
SET SYNC CONNECT 'connect_string_to_server_1,
connect_string_to_server_2' TO REPLICA replica_nodename
```

Alternatively, you can save the statement in the replica server and propagate it to master the next time that you synchronize. In that case, use the following statement:

```
SAVE SET SYNC CONNECT 'connect_string_to_server_1,
connect_string_to_server_2' TO REPLICA replica_nodename
```



1. Failover in ReplicaDB
2. SET SYNC CONNECT 'tcp machine4 1315' REPLICA TO replicaDB1

Figure 18. HotStandby and advanced replication: failover of Replica database

If the master server never executes remote procedure calls in the replica, the above statement is not needed.

5 Failure handling with High Availability Controller (HAC)

This section describes possible failure scenarios and typical recovery procedures for them if using the High Availability Controller (HAC). HAC handles various failure scenarios implicitly. However, different failure or initialization scenarios (administrative scenarios, for short) can be handled by a human administrator, or a *watchdog* type software program.

HAC is a watchdog type program that monitors Primary and Secondary servers, and gives commands to change those servers' states when necessary. For example, HAC can determine when the Primary or Secondary server itself has failed or when just the communication link between these servers is down.

The purpose of recovery is to bring the failed component back to operation. Occasionally, further failures happen during recovery. They usually lead to a situation where the system remains in a state of limited availability (only one server is up), awaiting human intervention. Typical recovery-time failures that are not automatically taken care of are:

- The failed database is corrupted to a point that it is impossible to restart it.
- There is not enough free disk space to perform a catchup.

5.1 Primary database fails

Scenario

The primary database (in the PRIMARY ACTIVE state) on node 1 fails.

The secondary database (in the SECONDARY ACTIVE state) on node 2 encounters connection failure to the primary database on node 1.

Recovery

In the recovery from the Primary database failure the Secondary server replaces the Primary server. The recovery proceeds automatically as follows:

1. Upon the connection failure, the Secondary database on node 2 moves automatically to the SECONDARY ALONE state.
2. The HAC instance on the Secondary database on node 2 concludes that the Primary database on node 1 has failed and sets the Secondary database on node 2 to the PRIMARY ALONE state.
3. In parallel with the above task, the HAC instance on the Primary database on node 1 restarts the Primary database, which enters the SECONDARY ALONE state.
4. The HAC instance on the Secondary database on node 2 initiates the process of connecting the Primary and Secondary database.
5. A catchup is made.

   Optionally, the connection process includes a **netcopy** operation from the Secondary database on node 2 to the Primary database on node 1.

5.2 Secondary database fails

Scenario

The secondary database (in the SECONDARY ACTIVE state) on node 2 fails.

The Primary database (in the PRIMARY ACTIVE state) on node 1 encounters connection failure to the Secondary database on node 2.

Recovery

In the recovery from the Secondary database failure the Secondary server is restarted. The recovery proceeds automatically as follows:

1. Upon the connection failure, the Primary database on node 1 moves automatically to the PRIMARY UNCERTAIN state, or if the **AutoPrimaryAlone** parameter is enabled, to the PRIMARY ALONE state.
2. The HAC instance on the Primary database on node 1 concludes that the Secondary database on node 2 has failed.
3. If the Primary database was set to the PRIMARY UNCERTAIN state in step 1, HAC sets it now to the PRIMARY ALONE state.
4. In parallel with the above task, the HAC instance on the Secondary database on node 2 restarts the Secondary database, which enters the SECONDARY ALONE state.
5. The HAC instance on the Primary database on node 1 initiates the process of connecting the Primary and Secondary database.
6. A catchup is made.

   Optionally, the connection process includes a **netcopy** operation from the Primary database to the Secondary database.

5.3 Primary node fails

Scenario

The primary node (node 1) fails.

The Secondary database (in the SECONDARY ACTIVE state) on node 2 encounters connection failure to the Primary database (in the PRIMARY ACTIVE state) on node 1.

Recovery

In the recovery from the Primary node failure the Primary server is restarted. The recovery proceeds automatically as follows:

1. Upon the connection failure, the Secondary database moves automatically to the SECONDARY ALONE state.
2. The HAC instance on the Secondary database on node 2 concludes that the Primary database on node 1 has failed.
3. The HAC instance on the Secondary database on node 2 sets the Secondary database to the PRIMARY ALONE state.
4. The Primary node (node 1) is restarted.
5. The HAC instance on the Primary database (node 1) is restarted.

6. The HAC instance on the Primary database (node 1) concludes that the Primary database is not running.
7. The HAC instance on the Primary database (node 1) restarts the Primary database and sets it to the SECONDARY ALONE state.
8. The HAC instance on the Secondary database on node 2 initiates the process of connecting the Primary and Secondary database.
9. A catchup is made.

   Optionally, the connection process includes a **netcopy** operation from the Primary database to the Secondary database.

5.4 Secondary node fails

Scenario

The secondary node (node 2) fails.

The Primary database (in the PRIMARY ACTIVE state) on node 1 encounters connection failure to the Secondary database (in the SECONDARY ACTIVE state) on node 2.

Recovery

In the recovery from the Secondary node failure the Secondary server is restarted. The recovery proceeds automatically as follows:

1. Upon the connection failure, the Primary database on node 1 moves automatically to the PRIMARY UNCERTAIN state.
2. The HAC instance on the Primary database on node 1 concludes that the Secondary database on node 2 has failed.
3. The HAC instance on the Primary database on node 1 sets the Primary database to the PRIMARY ALONE state.
4. The Secondary node (node 2) is restarted.
5. The HAC instance on the Secondary database (node 2) is restarted.
6. The HAC instance on the Secondary database (node 2) concludes that the Secondary database is not running.
7. The HAC instance on the Secondary database (node 2) restarts the Secondary database and sets it to the SECONDARY ALONE state.
8. The HAC instance on the Primary database on node 1 initiates the process of connecting the Primary and Secondary database.
9. A catchup is made.

   Optionally, the connection process includes a **netcopy** operation from the Primary database to the new Secondary database.

5.5 HotStandby link fails

Scenario

The HotStandby link fails.

The Primary database (in the PRIMARY ACTIVE state) on node 1 and the Secondary database (in the SECONDARY ACTIVE state) on node 2 both encounter connection failure to each other.

Recovery

In the recovery from the HotStandby link failure the HAC instances ping the External Reference Entity (ERE) to find out if it is the network or the opposite server that has failed. The recovery proceeds automatically as follows:

1. Upon the connection failure, both databases move automatically to the PRIMARY UNCERTAIN (node 1) and SECONDARY ALONE state (node 2), respectively.
2. The direct connections of both HAC instances to the remote server fail.
3. Both HAC instances ping the ERE by using the operating system's ping utility.
4. If the ping fails, the local server is retained or set to the SECONDARY ALONE state.
5. If the ping succeeds, the successful HAC tries to connect to the remote database server.
6. If the connect effort to the remote database server fails, the HAC concludes that its part of the network connection is operational and sets the local server to the PRIMARY ALONE state.
7. The HAC instance on the Primary database attempts to re-establish the connection to the Secondary database.
8. Once the network becomes operational and the connect succeeds, the Primary database and the Secondary database move automatically to the PRIMARY ACTIVE and SECONDARY ACTIVE state, respectively.

5.6 Server is unresponsive to external clients

Scenario

Connection to the server fails or hangs forever. Servers are in PRIMARY ACTIVE, and SECONDARY ACTIVE state, but clients cannot connect to them and they cannot execute transactions.

Recovery

Regardless of the state of unresponsive server, HAC executes a script as configured in the `solidhac.ini` configuration file with the **`LocalDB.UnresponsiveActionScript`** parameter. The script is started with single parameter including the process id (pid) of unresponsive solidDB process.

Typical solution is to terminate the process identified by the process id. In such a case, the recovery proceeds automatically as follows:

If the unresponsive server was in Primary role:

1. Upon the connection failure, the Secondary database (node 2) moves automatically to the SECONDARY ALONE state.
2. The HAC instance on the Secondary database (node 2) notices that the Primary database process (node 1) has terminated and sets the Secondary database (node 2) to the PRIMARY ALONE state.
3. In parallel with the above task, the HAC instance on the Primary database (node 1) restarts the old Primary database, which enters the SECONDARY ALONE state.
4. The HAC instance on the Secondary database (node 2) initiates the process of connecting the Primary and Secondary database.
5. A catchup is made.

Optionally, the connection process includes a **netcopy** operation from the Secondary database (node 2) to the Primary database (node 1).

If the unresponsive server was in Secondary role:

1. Upon the connection failure, the Primary database (node 1) moves automatically to the PRIMARY UNCERTAIN state, or if the AutoPrimaryAlone parameter is enabled, to the PRIMARY ALONE state.
2. The HAC instance on the Primary database (node 1) notices that the Secondary database process (node 2) has terminated.
3. If the Primary database was set to the PRIMARY UNCERTAIN state in step 1, HAC sets it now to the PRIMARY ALONE state.
4. In parallel with the above task, the HAC instance on the Secondary database (node 2) restarts the old Secondary database, which enters the SECONDARY ALONE state.
5. The HAC instance on the Primary database (node 1) initiates the process of connecting the Primary and Secondary database.
6. A catchup is made.

   Optionally, the connection process includes a **netcopy** operation from the Primary database to the Secondary database.

6 Upgrading (migrating) HotStandby servers

Migration involves updating the version of the software. For a highly-available system like solidDB HotStandby, the migration can be *cold* or *hot*.

Cold migration means that you shut down the whole system (both servers), upgrade the system, and restart with new software and configuration data.

Hot migration means that you upgrade your HotStandby server pair without taking your entire system offline for the amount of time required to upgrade the servers. One server can keep operating while the other server is being upgraded.

Important: With hot migration, although your entire system will not be down, users or applications might have to disconnect from one server and connect to the other server if you are using basic connectivity. If you are using transparent connectivity, you can perform the hot migration transparently to users and applications.

6.1 Cold migration procedure

About this task

The migration steps are:

Procedure

1. Disconnect the servers and shut them down.
2. Install the new version of the software.
3. If you are upgrading to a new version, copy the new license file (`solid.lic`) from the License Certificate image to both the Primary and Secondary servers.
4. If necessary, update the `solid.ini` files.
5. Start Primary with the command line parameter **`-x autoconvert`**, which instructs the server to convert existing database to the new format.
6. Set the Primary to the PRIMARY ALONE state.
7. Perform 'hsb netcopy' from Primary to Secondary.
8. Connect the servers.

6.2 Hot migration procedure

This hot migration procedure is supported as of solidDB server version level 7.0 Interim Fix 2 / 7.0 Fix Pack 3. For versions prior to Fix Pack 3, use the *hot migration procedure using netcopy*.

Before you begin

1. If your applications have not been designed to fail over automatically, notify users that they will lose their connections and will need to reconnect to the new Primary server.
2. Prepare your system and your software for upgrades:
   a. Since each of your solidDB servers will be operating alone (specifically, in PRIMARY ALONE state) during part of the upgrade operation, make sure

that both computers are in a healthy state. For example, both computers should have sufficient free disk space, reliable network connections, and a UPS device in case of power failure.

b. Each of the servers will operate in PRIMARY ALONE state for at least a short time (while the other server is being upgraded). While the server is in PRIMARY ALONE state, it will be storing transactions in the transaction log. You must have enough disk space available for the log file to store all the transactions that will occur while the other server is being upgraded, including the time it takes that other server to catch up after it is restarted.

c. Make sure that a copy of the new software version is available on each computer.

Note: solidDB is delivered as a single installation file. If you do not want to run the installer on your production environment node, install solidDB on a separate node and copy the executables, libraries, and drivers manually to your production node, as applicable for your setup.

Tip: The upgrade procedure differs depending on whether you are upgrading to a new version level (for example, from 6.5 to 7.0) or applying a fix pack (for example, 7.0 Fix Pack 2 to 7.0 Fix Pack 3). The differences are described in the procedure.

3. If you have a watchdog program, turn it off temporarily so that it does not issue commands that conflict with the commands that you issue during the upgrade process. For example, after you disconnect the Primary from the Secondary, you do not want the watchdog to try to reconnect them before you upgrade the Secondary.

About this task

In the steps below, S1 ("OP") and S2 ("OS") represent the *original* Primary and Secondary servers. Each server's state changes as you go through this process.

Procedure

1. S1: Disconnect the Primary from the Secondary:
   ```
   ADMIN COMMAND 'hsb disconnect';
   ```
2. S2 OS: Shut down server S2 (Secondary):
   ```
   ADMIN COMMAND 'shutdown force';
   ```
3. S1 OP: Tell server S1 (Primary) to operate in PRIMARY ALONE state if it has not already automatically switched to that state. Verify that server S1 (Primary) is in the PRIMARY ALONE state.
   ```
   ADMIN COMMAND 'hsb set primary alone';
   ADMIN COMMAND 'hsb state';
   ```
4. S2 OS: Upgrade server S2 (original Secondary) to the new solidDB version level.

   Note: solidDB is delivered as a single installation file. If you do not want to run the installer on your production environment node, install solidDB on a separate node and copy the executables, libraries, drivers, and license file manually to your production node, as applicable for your setup. If necessary, update also the `solid.ini` file.
5. S2 OS: Bring up server S2 (original Secondary).
   - If you are upgrading to a new version level (for example, 6.5 to 7.0), start the solidDB server using the **`-x autoconvert`** command-line option.

```
solid -x autoconvert
```

The **-x autoconvert** option instructs the server to accept and convert the existing database to the new version level.

- If you are applying a new fix pack level (for example, 7.0 Fix Pack 2 to 7.0 Fix Pack 3), you can start the solidDB server normally. Conversion of the database is not needed.

The S2 server comes up in SECONDARY ALONE state.

6. S1 OP: Connect the Primary server to the Secondary:

```
ADMIN COMMAND 'hsb connect';
```

Note: If the Secondary server is running a newer version of the server, you cannot issue the **hsb connect** command on the Secondary.

This step starts the process by which the Secondary *catches up* on data changes that occurred while the Secondary was down.

7. S1 OP: Wait for the *catchup* to complete before continuing.
8. S2: After the servers are connected and caught up, perform a role switch to prepare for shutting down the current Primary (S1 OP):

```
ADMIN COMMAND ' hsb switch primary';
```

9. S2: Disconnect the Primary from the Secondary:

```
ADMIN COMMAND 'hsb disconnect';
```

10. S1 OP: Shut down server S1 OP (now in Secondary mode):

```
ADMIN COMMAND 'shutdown force';
```

Important: The **force** option aborts all open transactions.

11. S2 OS: Set the new Primary server S2 (old Secondary) to operate in the PRIMARY ALONE state, if it has not already automatically switched to that state. Verify that server S2 is in the PRIMARY ALONE state.

```
ADMIN COMMAND 'hsb set primary alone';
ADMIN COMMAND 'hsb state';
```

12. S1 OP: Upgrade server S1 (your original Primary server) to the new solidDB version level.

Note: solidDB is delivered as a single installation file. If you do not want to run the installer on your production environment node, install solidDB on a separate node and copy the executables, libraries, drivers, and license file manually to your production node, as applicable for your setup. If necessary, update also the `solid.ini` file.

13. S1 OP: Bring up server S1.

- If you are upgrading to a new version level (for example, 6.5 to 7.0), start the solidDB server using the **-x autoconvert** command-line option.

```
solid -x autoconvert
```

The **-x autoconvert** option instructs the server to accept and convert the existing database to the new version level.

- If you are applying a new fix pack level (for example, 7.0 Fix Pack 2 to 7.0 Fix Pack 3), you can start the solidDB server normally. Conversion of the database is not needed.

The S1 OP server comes up in SECONDARY ALONE state.

14. S2 OS: Connect the servers:

```
ADMIN COMMAND 'hsb connect';
```

The **hsb connect** command connects the new Primary server to the new Secondary and starts the process by which the new Secondary catches up on data changes that occurred while it was down.

If this step fails, copy the entire database to the Secondary server using **hsb netcopy** and then resume from step 12.

Results

After the new Secondary server catches up to the new Primary, your system should be operating normally. Both the new Primary and the new Secondary server have been upgraded and have the most current data.

What to do next

After the upgrade, you might want to run some test queries to make sure that everything is operating properly.

1. Test that both your Primary server and your Secondary server are working correctly. For example, you might choose the following sequence of operations:

   On the Primary:

   ```
   ADMIN COMMAND 'hsb state';
   ADMIN COMMAND 'hsb status catchup';
   ```

   Issue any type of read-only query.

   On the Secondary:

   ```
   ADMIN COMMAND 'hsb state';
   ADMIN COMMAND 'hsb status catchup';
   ```

   Issue any type of read-only query.
2. If you had a watchdog program, restart it.

Related tasks:

6.3, “Hot migration procedure using netcopy”
The hot migration procedure using netcopy must be used when upgrading the solidDB server up to version level 7.0 Fix Pack 2.

6.3 Hot migration procedure using netcopy

The hot migration procedure using netcopy must be used when upgrading the solidDB server up to version level 7.0 Fix Pack 2.

Before you begin

1. If your applications have not been designed to fail over automatically, notify users that they will lose their connections and will need to reconnect to the new Primary server.
2. Prepare your system and your software for upgrades:
   a. Since each of your solidDB servers will be operating alone (specifically, in PRIMARY ALONE state) during part of the upgrade operation, make sure that both computers are in a healthy state. For example, both computers should have sufficient free disk space, reliable network connections, and a UPS device in case of power failure.
   b. Each of the servers will operate in PRIMARY ALONE state for at least a short time (while the other server is being upgraded). While the server is in PRIMARY ALONE state, it will be storing transactions in the transaction log. You must have enough disk space available for the log file to store all

the transactions that will occur while the other server is being upgraded, including the time it takes that other server to catch up after it is restarted.

c. Make sure that a copy of the new software version is available on each computer.

   Note: solidDB is delivered as a single installation file. If you do not want to run the installer on your production environment node, install solidDB on a separate node and copy the executables, libraries, and drivers manually to your production node, as applicable for your setup.

   Tip: The upgrade procedure differs depending on whether you are upgrading to a new version level (for example, from 6.5 to 7.0) or applying a fix pack (for example, 7.0 Fix Pack 2 to 7.0 Fix Pack 3). The differences are described in the procedure.

3. If you have a watchdog program, turn it off temporarily so that it does not issue commands that conflict with the commands that you issue during the upgrade process. For example, after you disconnect the Primary from the Secondary, you do not want the watchdog to try to reconnect them before you upgrade the Secondary.

About this task

In the steps below, S1 ("OP") and S2 ("OS") represent the *original* Primary and Secondary servers. Each server's state changes as you go through this process.

Procedure

1. S1: Disconnect the Primary from the Secondary:

   ```
   ADMIN COMMAND 'hsb disconnect';
   ```

2. S2 OS: Shut down server S2 (Secondary):

   ```
   ADMIN COMMAND 'shutdown force';
   ```

3. S1 OP: Tell server S1 (Primary) to operate in PRIMARY ALONE state if it has not already automatically switched to that state. Verify that server S1 (Primary) is in the PRIMARY ALONE state.

   ```
   ADMIN COMMAND 'hsb set primary alone';
   ADMIN COMMAND 'hsb state';
   ```

4. S2 OS: Upgrade server S2 (original Secondary) to the new solidDB version level.

   Note: solidDB is delivered as a single installation file. If you do not want to run the installer on your production environment node, install solidDB on a separate node and copy the executables, libraries, drivers, and license file manually to your production node, as applicable for your setup. If necessary, update also the `solid.ini` file.

5. S2 OS: Bring up server S2 (original Secondary).

   - If you are upgrading to a new version level (for example, 6.5 to 7.0), start the solidDB server using the **`-x autoconvert`** command-line option.

     ```
     solid -x autoconvert
     ```

     The **`-x autoconvert`** option instructs the server to accept and convert the existing database to the new version level.

   - If you are applying a new fix pack level (for example, 7.0 Fix Pack 2 to 7.0 Fix Pack 3), you can start the solidDB server normally. Conversion of the database is not needed.

The S2 server comes up in SECONDARY ALONE state.

6. S1 OP: Netcopy the database from the old Primary (S1) to the new Secondary (S2):

   ```
   ADMIN COMMAND 'hsb netcopy';
   ```

7. S1 OP: Verify that the netcopy succeeded:

   ```
   ADMIN COMMAND 'hsb status copy';
   ```

8. S1 OP: Connect the Primary server to the Secondary:

   ```
   ADMIN COMMAND 'hsb connect';
   ```

   Note: If the Secondary server is running a newer version of the server, you cannot issue the **hsb connect** command on the Secondary.

   This step starts the process by which the Secondary *catches up* on data changes that occurred while the Secondary was down.

9. S1 OP: Wait for the *catchup* to complete before continuing.
10. S2: After the servers are connected and caught up, perform a role switch to prepare for shutting down the current Primary (S1 OP):

    ```
    ADMIN COMMAND ' hsb switch primary';
    ```

11. S2: Disconnect the Primary from the Secondary:

    ```
    ADMIN COMMAND 'hsb disconnect';
    ```

12. S1 OP: Shut down server S1 OP (now in Secondary mode):

    ```
    ADMIN COMMAND 'shutdown force';
    ```

    Important: The **force** option aborts all open transactions.

13. S2 OS: Set the new Primary server S2 (old Secondary) to operate in the PRIMARY ALONE state, if it has not already automatically switched to that state. Verify that server S2 is in the PRIMARY ALONE state.

    ```
    ADMIN COMMAND 'hsb set primary alone';
    ADMIN COMMAND 'hsb state';
    ```

14. S1 OP: Upgrade server S1 (your original Primary server) to the new solidDB version level.

    Note: solidDB is delivered as a single installation file. If you do not want to run the installer on your production environment node, install solidDB on a separate node and copy the executables, libraries, drivers, and license file manually to your production node, as applicable for your setup. If necessary, update also the `solid.ini` file.

15. S1 OP: Bring up server S1.

    - If you are upgrading to a new version level (for example, 6.5 to 7.0), start the solidDB server using the **-x autoconvert** command-line option.

      ```
      solid -x autoconvert
      ```

      The **-x autoconvert** option instructs the server to accept and convert the existing database to the new version level.

    - If you are applying a new fix pack level (for example, 7.0 Fix Pack 2 to 7.0 Fix Pack 3), you can start the solidDB server normally. Conversion of the database is not needed.

    The S1 OP server comes up in SECONDARY ALONE state.

16. S2 OS: Netcopy the database from the new Primary (S2) to the new Secondary (S1):

    ```
    ADMIN COMMAND 'hsb netcopy';
    ```

17. S2 OS: Verify that the netcopy succeeded:

```
ADMIN COMMAND 'hsb status copy';
```

18. S2 OS: Connect the servers:

    ```
    ADMIN COMMAND 'hsb connect';
    ```

    The **hsb connect** command connects the new Primary server to the new Secondary and starts the process by which the new Secondary catches up on data changes that occurred while it was down.

    If this step fails, copy the entire database to the Secondary server using **hsb netcopy** and then resume from step 14.

Results

After the new Secondary server catches up to the new Primary, your system should be operating normally. Both the new Primary and the new Secondary server have been upgraded and have the most current data.

What to do next

After the upgrade, you might want to run some test queries to make sure that everything is operating properly.

1. Test that both your Primary server and your Secondary server are working correctly. For example, you might choose the following sequence of operations:

   On the Primary:

   ```
   ADMIN COMMAND 'hsb state';
   ADMIN COMMAND 'hsb status catchup';
   ```

   Issue any type of read-only query.

   On the Secondary:

   ```
   ADMIN COMMAND 'hsb state';
   ADMIN COMMAND 'hsb status catchup';
   ```

   Issue any type of read-only query.

2. If you had a watchdog program, restart it.

Appendix A. HotStandby configuration parameters

This section discusses the configuration parameters used with HotStandby.

There are four types of parameters:

- Server-side parameters in the server-side `solid.ini` configuration file
  - There are two HotStandby-specific sections in the server-side `solid.ini` configuration file: `[Cluster]` and `[HotStandby]`.
  - Various other parameters can also affect HotStandby functionality.
- Client-side parameters in the client-side `solid.ini` configuration file
- High Availability Controller (HAC) configuration parameters in the `solidhac.ini` configuration file
- High Availability Manager (HAM) configuration parameters in the `HAManager.ini` configuration file

Tip: If you use the Watchdog sample provided with the solidDB package, the `solid.ini` will also contain watchdog-specific parameters in a `[Watchdog]` section. For information, see the section Appendix F, “Watchdog sample,” on page 171.

Access mode

The access mode of a parameter defines whether the parameter can be changed dynamically, and when the change takes effect. When the value of a parameter is changed with an ADMIN COMMAND, the change might not apply immediately, nor the next time that the server is started. If a parameter value is written to the `solid.ini` file, it will take effect the next time that the server starts.

Access mode values

The possible access modes are:

- *RO* (read-only): the value cannot be changed; the current value is always identical to the startup value.
- *RW*: can be changed through an ADMIN COMMAND, and the change takes effect immediately.
- *RW/Startup*: can be changed through an ADMIN COMMAND, and the change takes effect the next time that the server starts.
- *RW/Create*: can be changed through an ADMIN COMMAND, and the change applies when a new database is created.

Related concepts:

3.2, “Configuring HotStandby,” on page 37
HotStandby is configured by using the `solid.ini` configuration files at both the Primary and Secondary nodes. The `[HotStandby]` section contains all the HotStandby-specific configuration parameters. Other sections and parameters, such as the **`Com.Listen`** parameter, must be set also.

A.1 Server-side parameters

The server-side configuration parameters define various performance, memory and disk usage, and other characteristics of the solidDB server. For HotStandby use, the most important server-side parameters are in the **[HotStandby]** and **[Cluster]** sections.

A.1.1 Cluster section

Table 22. Cluster parameters

[Cluster]	Description	Factory Value	Access Mode
ReadMostlyLoadPercentAtPrimary	Defines the percentage of read loads that are directed to the Primary when load balancing is set to READ_MOSTLY.	50	RW/Startup

A.1.2 HotStandby section

Table 23. HotStandby parameters

HotStandby	Description	Factory value	Access mode
1SafeMaxDelay	In 1-Safe replication, the maximum delay before a committed transaction is sent to the Secondary (in milliseconds).	5000	RW
2SafeAckPolicy	This specifies the timing of the Secondary's acknowledgement when it receives a transaction from the Primary. Valid values are: • 1 = 2-safe received. The Secondary server acknowledges when it receives the data. • 2 = 2-safe visible. The Secondary server acknowledges when the data is "visible", that is, when the Secondary has executed the transaction. • 3 = 2-safe durable. The Secondary server acknowledges when it has made the data durable, that is, when it has committed the data and written the data to the disk. 2-safe durable is the safest approach, and 2-safe received has the fastest response time. However, in practice, the 2-safe received mode provides in most cases sufficient guarantees for data safety hence providing the best compromise between safety and speed. This parameter applies only if the server is using 2-safe replication. **Note:** Although this parameter controls the Secondary server's behavior, this parameter is set on the Primary. The value in the Secondary's `solid.ini` value is ignored.	1	RW
AutoPrimaryAlone	If this parameter is set to `yes`, the server is automatically put in PRIMARY ALONE state (rather than PRIMARY UNCERTAIN state) when the connection to the Secondary is broken.	`no`	RW

Table 23. HotStandby parameters (continued)

HotStandby	Description	Factory value	Access mode
CatchupSpeedRate	While the server is performing catchup, it also continues to service database requests from clients. You may use the **CatchupSpeedRate** parameter to give greater importance to responding to application requests and lower priority to catchup, or vice versa. The speed rate is expressed as a percentage of the maximum available speed dictated by the link and Secondary throughput. Larger numbers mean more emphasis on catchup and less on servicing client requests. Valid values are 1-99.	50	RW
Connect	The **Connect** parameter indicates the address of the other HotStandby server in the pair. The value of this parameter is a standard solidDB connect string (Basic Connectivity) or a TC-specific connect string (Transparent Connectivity). The connect string defined with this parameter must match the server listening name of the other HotStandby server (defined with **Com.Listen** parameter). If you omit this parameter in a server that you intend for HotStandby, you can set this parameter dynamically by using an ADMIN COMMAND. Until the server has a connect string, the server can only be in the states that do not involve a HotStandby connection, that is, PRIMARY ALONE, SECONDARY ALONE, and STANDALONE. If **HSBEnabled** is set to no, this parameter is ignored. For Transparent Connectivity (TC) connections with multi-home servers, the **Connect** parameter can be overridden with the **HotStandby.TCConnect** parameter.	No factory value.	RW

ConnectTimeout	By specifying a connect timeout value, you can set the maximum time in seconds that a HotStandby connect operation waits for a connection to a remote machine.
	
	The **ConnectTimeout** parameter is used with the following administration commands:
	• **hotstandby connect**
	• **hotstandby switch primary**
	• **hotstandby switch secondary**
	
	For example, to set the timeout to 30 seconds (30000 milliseconds):
	`[HotStandby]`
	`ConnectTimeout=30000`
	
	See also **PingTimeout**.
		0 (no timeout)
		
		Unit: 1 ms
			RW

Table 23. HotStandby parameters (continued)

HotStandby	Description	Factory value	Access mode

CopyDirectory	The **CopyDirectory** parameter in the [HotStandby] section defines a name and location for the HotStandby copy operation that is performed when the user executes the command:
	`ADMIN COMMAND 'hotstandby copy';`
	
	For example, the parameter may look like:
	`[HotStandby]`
	`CopyDirectory=C:\solidDB\secondary\dbfiles`
	
	If you provide a relative path for the **CopyDirectory** parameter, the path will be relative to the directory that holds the Primary server's `solid.ini` file.
	
	This parameter has no factory value, so if the directory is not specified in the `solid.ini` file, it must be provided in the copy command.
	
	The **ADMIN COMMAND 'hotstandby netcopy'** is the recommended way to copy the database because it is a more flexible solution.
		No factory value	RW

HSBEnabled	If this parameter is set to `yes`, the server may act as a HotStandby Primary or Secondary server. If this parameter is set to `no`, then the server may not act as a HotStandby server. Setting this parameter to `yes` will implicitly define the default initial state of the server to be SECONDARY ALONE when the server first starts. Valid values are `yes`and `no`. To use HotStandby, you must also specify the **Connect** parameter, either by setting it in the `solid.ini` file or by using an ADMIN COMMAND to set it.	`no`	RO
MaxLogSize	Defines the maximum size of the disk-based HSB log. The factory value: unlimited	`0` Unit: 1 byte k=KB m=MB	
MaxMemLogSize	When the file-based logging is disabled (**Logging.LogEnabled**=`no`), the size of the in-memory log holding transactions before they are sent to the Secondary. The value affects the time the server may stay in the PRIMARY ALONE state, before the in-memory log becomes full.	`8M` Unit: 1 byte k=KB m=MB	RO
NetcopyReceiveBufferSize	Defines the buffer size at Secondary server for storing data during netcopy. When the buffer is full, writes of netcopy data are throttled.	`32M` Unit: 1 byte k=KB m=MB	RW
NetcopyRpcCompress	Controls whether data compression is used for a **netcopy** connection.	`no`	RW
NetcopyRpcTimeout	Data transmission acknowledgment timeout for netcopy operation (in milliseconds)	`30000` Unit: 1 ms	RW
PingInterval	The Primary and Secondary send "ping" messages to each other at regular intervals to make sure that they are still connected. (These pings are independent of the transaction information that the Primary sends to the Secondary.) The value is equal to the interval (in milliseconds) between two consecutive pings sent by a server.	`1000` (one second) Unit: 1 ms	RW

Table 23. HotStandby parameters (continued)

HotStandby	Description	Factory value	Access mode
PingTimeout	The parameter specifies how long a server should wait before concluding that the other server is down or inaccessible. After the time specified (in milliseconds) has passed the server concludes that a connection is broken and changes the state accordingly. See also **ConnectTimeout**.	4000 (four seconds) Unit: 1 ms	RW
PrimaryAlone	This parameter is deprecated. Use the **AutoPrimaryAlone** parameter.	no	RW
SafenessLevel	This parameter sets the safeness level of the replication protocol. Possible values are: 1safe, 2safe and auto By using the auto value, you can allow the safeness level to dynamically change in relation to the durability level. If you set **SafenessLevel** to auto and set the durability to relaxed by using the SET DURABILITY command or the **DurabilityLevel** parameter, the safeness level is set to 1-safe, and when you set the durability level to strict, the safeness level is set to 2-safe. However, if **DurabilityLevel** is set to 2 (Adaptive Durability), the auto setting has no effect; the safeness level will always be 2-safe.	2-safe	RW
SecondaryThreads	This parameter defines the number of threads that the Secondary server uses for processing write operations. The optimal number of threads depends on the environment. In principle, Valid values are 1–256.	4	RW/Startup
TCConnect	This parameter defines the address of the other HotStandby server in the pair for a Transparent Connectivity (TC) connection, if the applications and servers need to use different networks to connect to each other (for example, when using multi-home servers). From the application connection perspective, the address specified with this parameter precedes the address defined with the **HotStandby.Connect** parameter. The TC connection will thus use the server addresses specified with this parameter, while the HotStandby connection between the servers uses the server addresses defined with the **HotStandby.Connect** parameter.	No factory value.	RW

A.2 Client-side parameters

The client-side configuration parameters define various characteristics for usage of the solidDB ODBC client and solidDB tools such as solidDB SQL Editor (**solsql**). For HotStandby use, the most important client-side parameters are in the **[Com]** and **[TransparentFailover]** sections. The client-side parameters are stored in the client-side solid.ini configuration file and are read when the client starts.

A.2.1 Com section

The [Com] section in the client-side solid.ini file contains the **Com.Connect** parameter that can be used with HotStandby.

Table 24. Client-side communication parameters

[Com]	Description	Factory Value
ClientReadTimeout	This parameter defines the connection (or read) timeout in milliseconds. A network request fails if no response is received during the time specified. The value 0 sets the timeout to infinite. This value can be overridden with the connect string option -r and, further on, with the ODBC attribute SQL_ATTR_CONNECTION_TIMEOUT. **Note:** This parameter applies only to the TCP protocol.	0 (infinite)
Connect	The **Connect** parameter defines the default network name (connect string) that the client uses to connect to the solidDB server, if the connect string is not specified in the connection parameters explicitly. This value is used also when the SQLConnect() call is issued with an empty data source name. The format of the standard solidDB connect string is: `protocol_name [options] [host_comput er_name] server_name` where options and server_name depend on the communication protocol. **Important:** In HotStandby and SMA setups, additional connect string attributes are used to specify further functionality, such as Transparent Connectivity (TC). For more details, see Network name and connect string syntax.	tcp localhost 1964 (Windows) upipe SOLID (Linux and UNIX)
ConnectTimeout	The **ConnectTimeout** parameter defines the login timeout in milliseconds. This value can be overridden with the connect string option -c and, further on, with the ODBC attribute SQL_ATTR_LOGIN_TIMEOUT. **Note:** This parameter applies only to the TCP protocol.	OS-specific
SocketLinger	This parameter controls the TCP socket linger (SO_LINGER) behavior after a close on the socket connection is issued. It indicates if the system attempts to deliver any buffered data (yes), or if the system discards it (no), when a close() is issued.	no
SocketLingerTime	This parameter defines the length of the time interval (in seconds) the socket lingers after a close is issued. If the time interval expires before the graceful shutdown sequence completes, an abortive shutdown sequence occurs (the data is discarded). The default value zero indicates that the system default is used (typically, 1 second)	0
Trace	If this parameter is set to yes, trace information about network messages for the established network connection is written to a file specified with the **TraceFile** parameter.	no
TraceFile	If the **Trace** parameter is set to yes, trace information about network messages is written to a file specified with this parameter. The trace file is output to the current working directory of the server or client, depending on which end the tracing is started.	soltrace.out

A.2.2 TransparentFailover section

The TransparentFailover parameters are client-side parameters.

Table 25. TransparentFailover parameters

[TransparentFailover]	Description	Factory value
ReconnectTimeout	This parameter specifies how long (in milliseconds) the driver should wait until it tries to reconnect to the primary in case of TF switchover or failover. If the driver cannot find the new primary (reconnect), an error is returned and the TF connection becomes broken.	10000
WaitTimeout	This parameter specifies how long (in milliseconds) the driver should wait for the server to switch state. When the driver tries to reconnect to the servers, it might connect to the server being in an intermediate (switching or uncertain) state.	10000

A.3 High Availability Controller (HAC) parameters

This section describes the High Availability Controller (HAC) configuration parameters in the `solidhac.ini` configuration file.

The HAC configuration file `solidhac.ini` is divided into different sections; the sections are described in the following sections. The parameters are presented in the same order as they are in the configuration file.

For an example of the `solidhac.ini`, see section A.5.2, "The solidhac.ini configuration file," on page 129.

The format of parameter names, values, and section headings in the `solidhac.ini` configuration file follows the same conventions as the `solid.ini` configuration file.

To change the HAC parameter settings, edit the `solidhac.ini` file and restart HAC.

[HAController] section

Table 26. HAC configuration parameters: [HAController] section

Parameter name	Description	Mandatory	Factory value
Listen	The value of the **Listen** parameter specifies the protocol and the port that the HAC uses for communication with the High Availability Manager (HAM) or the solidDB utilities such as **solsql** and **solcon**. For example, Listen=tcp 3135. • The only supported protocol is TCP/IP ('tcp'). • If listening cannot be started, for example, because the port is used by another process, the information is written to the log file (`hacmsg.out`), followed by the termination of HAC.	X	
NetcopyErrorLevel	This parameter defines the number of times a **hsb netcopy** operation followed by a **hsb connect** operation can fail before an error message is printed to the `hacmsg.out` file and HAC is switched to ADMINISTRATIVE mode. After you have fixed the problem, switch HAC back to the AUTOMATIC mode by issuing the command **admin command 'hacontroller setautomatic'**. The value of the parameter must be a positive 64-bit integer.		10

Table 26. HAC configuration parameters: [HAController] section (continued)

Parameter name	Description	Mandatory	Factory value
NetcopyWarningLevel	This parameter defines the number of times a **netcopy** operation followed by a **hsb connect** can fail before a warning message is printed to the hacmsg.out file. The value of the parameter must be a positive 64-bit integer.		3
StartInAutomaticMode	This parameter specifies the initial mode for HAC; it defines whether the HAC starts execution in the automatic mode. Once the HAC is running, it can be in one of the following modes: AUTOMATIC, or ADMINISTRATIVE. • In the AUTOMATIC mode (yes), HAC automatically tries to maximize the availability by changing the HSB states of the server, and restarting the server processes when necessary. • In the ADMINISTRATIVE mode (no), HAC only monitors the health of the servers. This parameter can be changed dynamically. The possible values are yes and no. **Note:** See also the **PreferredPrimary** parameter in the LocalDB section; the **PreferredPrimary** parameter is effective only if the **StartInAutomaticMode** parameter has value yes.		yes
EnableDBProcessControl	Setting **EnableDBProcessControl**=yes allows HAC to manage local server process by automatically starting the server, and by providing the user with commands to shutdown and restart the database process. This value is only valid if the HAC is in the AUTOMATIC mode. The possible values are yes and no. **Note:** Setting **EnableDBProcessControl**=yes makes the **StartScript** parameter in the [LocalDB] section mandatory.		no
EnableAutoNetcopy	Setting **EnableAutoNetcopy**=yes allows HAC to initiate netcopy when a HSB link cannot be established with the **hsb connect** command. The possible values are yes and no.		yes
RequiredConnectFailures	This parameter defines the number of consecutive failed connect attempts that are required before the server is considered to have failed. When a server state is unknown, or HAC needs for some other reason to ensure the state of the server, the non-blocking **SQLConnect** (**check**) command is used. If the execution of non-blocking **SQLConnects** in such a case fails, it is repeated multiple times before the server in question is considered non-responsive. The possible values are numerical values from 1 to unlimited.		1
CheckTimeout	This parameter defines the timeout in milliseconds between consecutive non-blocking **SQLConnect** commands in CHECK mode. Very small values tend to cause 'false positives'. That is, a server seems to be failed, although it is running, but was not able to respond within the timeout period. The possible values are milliseconds from 1 to unlimited.		150

Table 26. HAC configuration parameters: [HAController] section (continued)

Parameter name	Description	Mandatory	Factory value
CheckInterval	This parameter defines the interval between consecutive non-blocking **SQLConnect** commands. This value does not affect the failover time. Checking (polling) takes place typically after failure or during system startup. The possible values are milliseconds from 1 to unlimited.		1000
Username	Username for HAC. The username must begin with a letter or an underscore. Use lowercase letters from a to z, uppercase letters from A to Z, the underscore character "_", and numbers from 0 to 9.	X	
Password	Password for HAC for the user identified by the **Username** parameter. The password can begin with any letter, underscore, or number. Use lowercase letters from a to z, uppercase letters from A to Z, the underscore character "_", and numbers from 0 to 9.	X	
DBUsername	Username for the local HotStandby server to which the HAC connects. The database user should have either SYS_ADMIN_ROLE, or SYS_CONSOLE_ROLE.	X	
DBPassword	Password for the local HotStandby server for the user identified by the **DBUsername** parameter.	X	
ApplicationConnTestUsername	Defines the username for the connections used in application connection tests (**EnableApplicationConnCheck** is set to yes). If **EnableApplicationConnCheck** is set to yes and the value for this parameter is not set, the value of **DBUsername** is used.		
ApplicationConnTestPassword	Defines the password for the connections used in application connection tests (**EnableApplicationConnCheck** is set to yes). If **EnableApplicationConnCheck** is set to yes and the value for this parameter is not set, the value of **DBPassword** is used.		

[LocalDB] section

Table 27. HAC configuration parameters: [LocalDB] section

Parameter name	Description	Mandatory	Factory value
Connect	This parameter defines connect information of the local database server. HAC uses this information when it connects to the local server. The connect information consists of a communication protocol (tcp) and the server port, for example: tcp 2125.	X	

Table 27. HAC configuration parameters: [LocalDB] section (continued)

Parameter name	Description	Mandatory	Factory value
EnableApplicationConnTest	If set to yes, periodical testing for the application connection is enabled. The application connection test checks whether the connection the application uses to connect to the server is working. To test the connection, HAC connects to the server using the same connect information as the application and executes simple commands to ensure the server is responsive. The application connection test enables HAC to monitor the external availability of solidDB server. If connection to server cannot be established or the server does not respond to simple queries, HAC concludes that the service provided by server is not available. If the server process exist in the system, HAC calls the script specified with the parameter **UnresponsiveActionScript**. **Important:** Before enabling application connection test, ensure that you have created the user account the test uses. If the application connection test cannot connect to solidDB due to an invalid user name or password, it waits until the time specified with **ApplicationConnTestInterval** expires. An error is also output to hacmsg.out. Also, if **EnableApplicationConnTest** is set to yes and the user account the Application Connection Tester uses is missing from either server, solidDB might slow down considerably. To recover: 1. Suspend HAC operations with ADMIN COMMAND 'hac suspend'. 2. Create the user accounts on both servers. 3. Resume HAC operations with ADMIN COMMAND 'hac resume'.		no
ApplicationConnTestConnect	This parameter defines the connect information for application connection test connections (**EnableApplicationConnCheck** is set to yes). The value of this parameter needs to be the same as the connect information the application uses to connect to the server. The value consists of a communication protocol (tcp), server address, and port number, for example: tcp -i10.0.0.101 2125. If **ApplicationConnTestConnect** is not specified, the value defined with the parameter **Connect** is used.		
EnableUnresponsiveActions	If set to yes, a user-provided script is executed if an application connection test fails. The script is defined with the parameter **UnresponsiveActionScript**.		no
RequiredAppConnTestFailures	This parameter defines the number of times the application connection test commands are executed before the server is considered unresponsive.		3
ApplicationConnTestTimeout	This parameter defines timeout in milliseconds for consecutive commands used in application connection test.		5000
ApplicationConnTestInterval	This parameter defines the interval in milliseconds between consecutive non-blocking commands used in application connection test.		3000

Table 27. HAC configuration parameters: [LocalDB] section (continued)

Parameter name	Description	Mandatory	Factory value
StartScript	This parameter declares the name of the script which is used to initiate the database process, for example, /home/soliddb/start_solid.sh. This parameter is mandatory if HAC is configured to control the database process with the **EnableDBProcessControl** parameter: • If **EnableDBProcessControl** is set to yes, this parameter is mandatory. • If **EnableDBProcessControl** is set to no, this parameter is not effective. **Tip:** In Linux and UNIX environments, you can use the -x option to sh to print commands and their arguments as they are executed. For example: `#!/bin/sh -x` The log of the start script (start_solid.sh) is output by default into /tmp/hac_start_solid.err. The log of the starting solidDB process is output by default into /tmp/hac_start_solid.out.	See description	

UnresponsiveActionScript	This parameter defines the name and location of the script that contains the actions to be taken if application connection test fails, for example: /home/solid/terminate_solid.sh.
	
	When calling the script, HAC needs to specify the solidDB process identifier as a parameter. If HAC does not know the solidDB process id, the script cannot be executed.
	
	This parameter is mandatory if both **EnableApplicationConnTest** and **EnableUnresponsiveActions** are set to yes.
	
	Example:
	
	The following terminate_solid.sh script terminates the solidDB process with id 1 in Linux and UNIX operating systems:
	`#!/bin/sh`
	`#terminate_solid.sh`
	
	`ulimit -c unlimited`
	`kill -6 $1`
	`sleep 30`
	`kill -9 $1`
	
	Tip: In Linux and UNIX environments, you can use the -x option to sh to print commands and their arguments as they are executed.
	
	For example:
	`#!/bin/sh -x`
	
	The log of the action script (terminate_solid.sh) is output into /tmp/hac_terminate_solid.err. The log of the terminating solidDB process is output into /tmp/hac_terminate_solid.out.
		See description	

PreferredPrimary	This parameter defines whether the local server becomes the Primary when the *logpos* values of both servers are equal. If both servers have the same value in **PreferredPrimary**, the first server becomes the new Primary. This parameter is effective only if the **StartInAutomaticMode** parameter has value yes. The possible values are yes and no.		no

[RemoteDB] section

Table 28. HAC configuration parameters: [RemoteDB] section

Parameter name	Description	Mandatory	Factory value
Connect	This parameter defines connect information of the remote database server. HAC uses this information when it connects to the remote server. The remote database is defined by giving the communication protocol, the remote server IP address, and its port, for example, `tcp 192.168.3.123 2125`.	X	

[ERE] section

Table 29. HAC configuration parameters: [ERE] section

Parameter name	Description	Mandatory	Factory value
EREIP	This parameter identifies the IP address of an External Reference Entity, for example, `192.168.3.1`. See also the **RequiredPingFailures** parameter.		
RequiredPingFailures	This parameter defines the maximum number of consecutive Ping calls that must fail before HAC concludes that the server is disconnected from the ERE and isolated from the client network. This parameter can only be used with ERE. The possible values are numerical values from 1 to `unlimited`.		3

A.4 High Availability Manager (HAM) configuration parameters

This section describes the High Availability Manager configuration parameters in the `HAManager.ini` configuration file.

The format of parameter names, values, and section headings in the `HAManager.ini` configuration file follows the same conventions as the `solid.ini` configuration file.

Table 30. High Availability Manager configuration parameters

Parameter name	Description
Header_text	This parameter defines the header text for the HA manager. This value is shown in the user interface. This is a mandatory parameter.
Server1_host Server2_host	These two parameters define the host names of the HAC instances.
Server1_name Server2_name	These two parameters define the names of the HAC instances.
Server1_pass Server2_pass	These two parameters define the passwords of the HAC instances.

Table 30. High Availability Manager configuration parameters (continued)

Parameter name	Description
Server1_port Server2_port	These two parameters define the ports of the HAC instances.
Server1_user Server2_user	These two parameters define the usernames of the HAC instances.
Window_title	This parameter defines the window title for the HA manager. This value is shown in the user interface. This is a mandatory parameter.

A.5 Configuration file examples

The following sections give examples of different configuration files related to HotStandby.

A.5.1 The solid.ini configuration file

Below is a sample excerpt of the solidDB configuration file (`solid.ini`) for the first HotStandby server:

```
[Com]
; The first server listens to the network with this
; name
Listen = tcp 1320
[HotStandby]
HSBEnabled=yes
; The first server connects to the second server
; using the following connect string.
Connect = tcp 188.177.166.12 1321
AutoPrimaryAlone=No
[Logging]
LogEnabled=yes
```

Below is a sample excerpt of the solidDB configuration file (`solid.ini`) for the second HotStandby server:

```
[Com]
; The second server listens to the network using the following
; connect string.
Listen = tcp 1321
[HotStandby]
HSBEnabled=yes
; The second server connects to the first server
; using the following connect string.
Connect = tcp  188.177.166.11 1320
AutoPrimaryAlone=No
[Logging]
LogEnabled=yes
```

A.5.2 The solidhac.ini configuration file

A sample excerpt of the High Availability Controller (HAC) configuration file (`solidhac.ini`).

```
;=============================================
; NOTE : Copy this file as solidhac.ini
;        to solidhac working directory
```

```
;
; solidDB High Availability Controller inifile
;=============================================

[HAController]
;** HAC connect info
;** HAC clients, HA Manager, for example, use this information.
;** Mandatory
;** Listen=tcp 3135
Listen=

;** Setting StartInAutomaticMode=Yes starts HAC in AUTOMATIC mode.
;** In AUTOMATIC mode, solidhac automatically tries
;** to maximize the availability by changing the HSB states of the
;** server, and restarting the server processes when necessary.
;** In contrast, it can be in ADMINISTRATIVE mode
;** in which HAC only monitors the health of the servers.
;**
;** This is dynamically changeable parameter.
;** Optional
;** Values : Yes/No, default = Yes
StartInAutomaticMode=

;** Setting EnableDBProcessControl=Yes allows solidhac
;** manage local db process by automatically starting
;** the db, and by providing the user with commands to
;** shutdown and restart db process.
;**
;** Optional
;** Effective only when HAC is in AUTOMATIC mode.
;** Values : Yes/No, default = No
EnableDBProcessControl=

;** Setting EnableAutoNetcopy=Yes allows solidhac to initiate
;** netcopy when HSB link cannot be established with 'hsb connect'.
;**
;** Optional
;** Effective only when HAC is in AUTOMATIC mode.
;** Values : Yes/No, default = Yes
EnableAutoNetcopy=

;** When server state is unknown, or HAC needs, for some other reason, to
;** ensure the state of server, non-blocking SQLConnect command is used.
;** If the execution of non-blocking SQLConnect in such a case fails,
;** it is repeated multiple (RequiredConnectFailures) times before
;** the server in question is considered as non-responsive.
;**
;** Optional
;** Values : 1..n, default=2
RequiredConnectFailures=

;** Timeout in milliseconds for non-blocking SQLConnect commands.
;** Too short interval can cause 'false positives', server seems
;** to be failed because it wasn't able to respond within the timeout period.
;**
;** Optional
;** Values : 1..n, default=150 (milliseconds)
CheckTimeout=

;** Interval between consecutive non-blocking SQLConnect commands.
;** The value doesn't affect on failover time. Checking (polling)
;** takes place typically after failure, or during system startup.
;**
;** Optional
;** default = 1000 (milliseconds)
CheckInterval=
```

```
;** HAC user identification
;** Mandatory
Username=
Password=

;** HSB server user identification
;** Mandatory
DBUsername=
DBPassword=

;** Identification for application connection test
;** These values are used when ApplicationConnectionTest
;** thread monitors the connection, and availability of
;** the server.
;** If values are not set, and
;** LocalDB.EnableApplicationConnCheck=Yes, then DBUsername, and
;** DBPassword are used.
;**
;** Optional
ApplicationConnTestUsername=
ApplicationConnTestPassword=


[LocalDB]
;** soliddb connect info
;** Mandatory
;** Connect=tcp 2125
Connect=

;** Enable periodical connection testing in the server.
;** In practice, HAC connects to the server, and executes
;** simple command(s) to ensure the responsiveness.
;**
;** Optional
;** default = No
EnableApplicationConnTest=

;** Connect info for applications, used in application connection test,
;** if it is enabled.
;**
;** Optional, if not specified, LocalDB.Connect is used.
;** ApplicationConnect=tcp 10.0.0.101 2125
ApplicationConnTestConnect=

;** Enables execution of the user-provided script when application
;** connection test fails. The script is defined with
;** UnresponsiveActionScript.
;**
;** Optional
;** default = No
EnableUnresponsiveActions=

;** Number of times the application connection test commands
;** are executed before the server is considered unresponsive.
;**
;** Optional
;** default = 3
RequiredAppConnTestFailures=

;** Timeout in milliseconds for consecutive application connection
;** test commands.
;**
;** Optional
;** default = 5000 (milliseconds)
ApplicationConnTestTimeout=

;** Interval between consecutive non-blocking application connection
```

```
;** test commands.
;**
;** Optional
;** default = 30000 (milliseconds)
ApplicationConnTestInterval=

;** The name of the script, which is used to initiate the db process.
;**
;** Optional, except if HAC controls db process (EnableDBProcessControl=Yes).
;** Value is not effective if EnableHACActions=No or EnableDBProcessControl=No
;** StartScript=/home/solid/start_solid.sh
StartScript=

;** The name and location of the script that contains the intended actions that
;** take place if application connection test fails.
;** When calling the script, HAC needs to specify the solidDB® process
;** identifier as a parameter. If HAC does not know the solidDB process id,
;** the script cannot be executed.
;**
;** Optional, except if ApplicationConnectionTest=Yes, and
;** EnableUnresponsiveAction=Yes
;**
;** UnresponsiveActionScript=/home/solid/terminate_solid.sh
UnresponsiveActionScript=

;** Setting PreferredPrimary=Yes moves local HSB server to as Primary
;** in the case where either of the servers could start as Primary.
;** If both servers have PreferredPrimary=No, or no value, first
;** (new Primary) server wins.
;**
;** Optional
;** Value is not effective if EnableHACActions=No.
;** Values : Yes/No, default no
PreferredPrimary=


[RemoteDB]
;** soliddb connect info
;** Mandatory
;** Connect=tcp 192.168.3.123 2125
Connect=


[ERE]
;** IP address of an ERE
;** Optional
;** Connect=192.168.3.1
EREIP=

;** The number of consecutive ping calls that must
;** fail before HAC concludes that the server is
;** disconnected from ERE.
;**
;** Optional
;** Values : 1..n, default=3
;** RequiredPingFailures=10
RequiredPingFailures=
```

A.5.3 The HAManager.ini configuration file

Below is a sample excerpt of the High Availability Manager configuration file (HAManager.ini):

```
;================================================
 ;solidDB High-Availability Manager
; Configuration file HAManager.ini
; V. 0.3
```

```
; 2008-21-01
;================================================
;** HA Controller connect info, for example
;Server1_name = Server 1
;Server1_host = node1.acme.com
;Server1_port = 2220
;Server2_name = Server 2
;Server2_host = node2.acme.com
;Server2_port = 2220

; All the following lines are mandatory.
Window_title = HA Manager
Header_text  = SolidDB HA Manager

; Display names, host addresses and port numbers
; of the SolidHAC (HA Controllers) instances
;Server 1 HA Controller
;----------------------
Server1_name = Server 1
Server1_host = localhost
Server1_port = 1234
Server1_user = foo
Server1_pass = bar
;
;Server 2 HA Controller
;----------------------
Server2_name = Server 2
Server2_host = 192.168.0.1
Server2_port = 1234
Server2_user = foo
Server2_pass = bar
```

Appendix B. Error codes for HotStandby

This section documents error codes that are related to HotStandby.

A full list of solidDB error codes is available in the appendix *Error codes* in *IBM solidDB Administrator Guide*.

Some of the errors documented in this section are values of the RC (Return Code) column of the ADMIN COMMAND result set, whereas some other errors are returned as the error code of the ODBC or JDBC driver. For example, most errors in sections B.1, "HotStandby errors and status codes" and B.2, "High Availability Controller errors and status codes," on page 144 are ADMIN COMMAND result set values, whereas all B.5, "solidDB communication errors," on page 148 are returned by the driver.

B.1 HotStandby errors and status codes

solidDB HotStandby errors (14009 - 147xx, 307xx) occur when using the HotStandby commands.

solidDB server errors for HotStandby

This section lists the solidDB server errors that are related to HotStandby. A full list of the errors in the Server class is available in section *solidDB server errors* in the *IBM solidDB Administrator Guide*.

Table 31. solidDB server errors for HotStandby

Code	Class	Type	Description
14003	Server	Return Code	ACTIVE ADMIN COMMANDs that may return this status in the result set of the command: • **`ADMIN COMMAND 'hotstandby status switch'`** • **`ADMIN COMMAND 'hotstandby status catchup'`** • **`ADMIN COMMAND 'hotstandby status copy'`** Meaning: The switch process, catchup process, copy or netcopy process is still active.
14007	Server	Return Code	CONNECTING ADMIN COMMANDs that may return this status in the result set of the command: • **`ADMIN COMMAND 'hotstandby status connect'`** Meaning: The Primary and Secondary servers are in the process of connecting.
14008	Server	Return Code	CATCHUP ADMIN COMMANDs that may return this status in the result set of the command: • **`ADMIN COMMAND 'hotstandby status connect'`** Meaning: The Primary server is connected to the Secondary server, but the transaction log is not yet fully copied. This message is returned only from the Primary server.

Table 31. solidDB server errors for HotStandby (continued)

Code	Class	Type	Description
14009	Server	Return Code	No server switch occurred before. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hotstandby status switch'** Meaning: The switch process has never happened between the servers.
14501	Server	Error	Operation failed. Meaning: The operation failed and the server is shutting down. Failure may be due to issuing the command to a non-HotStandby server, or to either a Primary or Secondary server in which the command does not apply. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hotstandby switch primary'** • **ADMIN COMMAND 'hotstandby switch secondary'** • **ADMIN COMMAND 'hotstandby cominfo'** • **ADMIN COMMAND 'hotstandby connect'** • **ADMIN COMMAND 'hotstandby status switch'** • **ADMIN COMMAND 'hotstandby set standalone'** • **ADMIN COMMAND 'hotstandby copy'** • **ADMIN COMMAND 'hotstandby netcopy'**
14502	Server	Error	RPC parameter is invalid Meaning: some of the connection info provided in the HSB connect string is erroneous and the connection to another server failed. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hotstandby status connect'**
14503	Server	Error	Communication error, connection lost. Meaning: There was a communication error and the other server was not found. There was a failure to connect to the other server. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hotstandby status connect'**
14520	Server	Error	Server is HotStandby secondary server, no connections are allowed.
14522	Server	Error	HotStandby copy directory not specified. Meaning: No copy directory is specified. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hotstandby copy'** To solve this problem, either specify the directory as part of the command, for example: `ADMIN COMMAND 'hotstandby copy \Secondary\dbfiles\'` or else set the **CopyDirectory** parameter in the `solid.ini` configuration file.

Table 31. solidDB server errors for HotStandby (continued)

Code	Class	Type	Description
14523	Server	Error	Switch process is already active. Meaning: The switch process is already active in the HotStandby server. If you only need to complete the current switch, then wait. If you are trying to switch a second time (that is, switch back to the original configuration), then you must wait for the first switch to complete before you can start the second switch. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hotstandby switch primary'** • **ADMIN COMMAND 'hotstandby switch secondary'** • **ADMIN COMMAND 'hotstandby status switch'**
14524	Server	Error	HotStandby databases have a different base database, database time stamps are different. Meaning: Databases are from a different seed database. You must synchronize databases. You may need to perform netcopy of the Primary's database to the Secondary. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hotstandby connect'** • **ADMIN COMMAND 'hotstandby status switch'**
14525	Server	Error	HotStandby databases are not properly synchronized. Meaning: Databases are not properly synchronized. You must synchronize the databases. You might need to start one of the database servers (the one that you intend to become the Secondary) with the command-line option **-x backupserver** and then netcopy the Primary's database to the Secondary. For more information about how to resynchronize the primary and secondary servers after receiving error 14525, see How to resync HotStandby primary and secondary servers if hsb netcopy takes such a long time that hsb connect fails with error 14525?. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hotstandby connect'** • **ADMIN COMMAND 'hotstandby status switch'**
14526	Server	Error	Invalid argument. Meaning: An argument used in the HotStandby ADMIN COMMAND is unknown or invalid. All HotStandby commands can return this error in the result set of the ADMIN COMMAND. Note: In the following HotStandby commands, the invalid argument error is a syntax error when the specified Primary or Secondary server can not apply to the switch: • **ADMIN COMMAND 'hotstandby switch primary'** • **ADMIN COMMAND 'hotstandby switch secondary'**

Table 31. solidDB server errors for HotStandby (continued)

Code	Class	Type	Description
14527	Server	Error	This is a non-HotStandby server. Meaning: The command was executed on a server that is not configured for HotStandby. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hotstandby connect'** • **ADMIN COMMAND 'hotstandby status switch'** • **ADMIN COMMAND 'hotstandby switch primary'** • **ADMIN COMMAND 'hotstandby switch secondary'** • **ADMIN COMMAND 'hotstandby state'**
14528	Server	Error	Both HotStandby databases are primary databases. Meaning: Both databases are Primary. This is a fatal error because there may be conflicting changes. Both databases are automatically dropped to Secondary state by the system. You must decide which database is the real Primary database and then synchronize the databases. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hotstandby connect'** • **ADMIN COMMAND 'hotstandby status switch'**
14535	Server	Error	Server is already a primary server. Meaning: The server you are trying to switch to Primary is already in one of the PRIMARY states. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hotstandby switch primary'**
14536	Server	Error	Server is already a secondary server. Meaning: The server you are trying to switch to Secondary is already in one of the SECONDARY states. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hotstandby switch secondary'**
14537	Server	Error	HotStandby connection is broken. Meaning: This command is returned from both the Primary and Secondary server. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hotstandby status connect'** • **ADMIN COMMAND 'hotstandby connect'** One possible cause of this problem is an incorrect Connect string in the Secondary's `solid.ini` file. If the netcopy operation succeeds but the connect command fails, check the Connect string. Netcopy does not require the Secondary to open a separate connection to the Primary, and thus may succeed even if the Connect string on the Secondary is wrong.

Table 31. solidDB server errors for HotStandby (continued)

Code	Class	Type	Description
14538	Server	Error	Server is not HotStandby primary server. Meaning: To issue this command, the server must be a HotStandby Primary server. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hotstandby copy copy_directory'** • **ADMIN COMMAND 'hotstandby netcopy'** • **ADMIN COMMAND 'hotstandby connect'** • **ADMIN COMMAND 'hotstandby set primary alone'** • **ADMIN COMMAND 'hotstandby set standalone'**
14539	Server	Error	Operation Refused. This error code is given when one of the following situations occurs: • The user issued a **netcopy** command to a Primary server, but the server that should be Secondary is not actually in a Secondary state, or is not in "netcopy listening mode". (Both the Primary and the "Secondary" server are probably in PRIMARY ALONE state.) To solve the problem, restart the "Secondary" with the **-x backupserver** command-line option, then try again to issue the **netcopy** command to the Primary. **Attention:** If both servers were in PRIMARY ALONE state, and if both servers executed transactions while they were in PRIMARY ALONE state, then they probably each have data that the other one does not. This is a serious error, and doing a **netcopy** to put them back in sync would result in writing over some transactions that have already been committed in the "Secondary" server. • This message can be generated when you use a callback function and the callback function refuses to shut down or accept a backup or netcopy command. When you use linked library access, you can provide "callback" functions by using the SSCSetNotifier function. Your callback functions will be notified when the server has been commanded to shut down or to do a netcopy operation. If for some reason your application doesn't want the command to be followed, then your callback can return a value that cancels the command. In this situation, you will see error 14539. To solve the problem, wait until the client code finishes the operation that it does not want to interrupt, then retry the command (for example, the shutdown or netcopy).
14540	Server	Error	Server is already a non-HotStandby server.
14541	Server	Error	HotStandby configuration in `solid.ini` conflicts with **ADMIN COMMAND 'HSB SET STANDALONE'**.
14542	Server	Error	Server in backupserver mode. Operation refused.
14543	Server	Error	Invalid command. The database is a HotStandby database but, HotStandby section not found in `solid.ini` configuration file.
14544	Server	Error	Operation failed. This command is not supported on diskless server.
14545	Server	Error	Primary can only be set to primary alone when its role is primary broken.

Table 31. solidDB server errors for HotStandby (continued)

Code	Class	Type	Description
14546	Server	Error	Switch failed. The server or the remote server cannot switch from primary alone to secondary server. Catchup should be done first before switch. Meaning: This command is returned when a state switch to SECONDARY is executed from a local or remote Primary server that is in the PRIMARY ALONE state and it is detected that the Primary and Secondary server are not in sync. You must connect the Primary server to the Secondary server and wait for the catchup process to complete before switching the Secondary to the Primary. HotStandby commands that return this error: • **ADMIN COMMAND 'hotstandby switch secondary'**
14547	Server	Error	The value for the **-R** option (Read Timeout) was missing or invalid.
14548	Server	Error	Switch failed. The server in Standalone cannot be switched to a secondary. Meaning: This command is returned when a state switch to SECONDARY is executed from a local or remote Primary server that is in the STANDALONE state and it is detected that the Primary and Secondary server are not in sync. You must connect the Primary server to the Secondary server and wait for the catchup to complete before switching the Secondary to the Primary. HotStandby commands that return this error: • **ADMIN COMMAND 'hotstandby switch secondary'**
14549	Server	Error	HotStandby transaction is active. Meaning: If the HotStandby connection is broken, Primary server must be set to alone mode or switched to secondary mode before shutdown.
14550	Server	Error	Hotstandby connect parameter can be changed only when the primary is not connected to secondary.
14551	Server	Error	Maximum number of START AFTER COMMIT statements reached.
14552	Server	Error	Server is in backup server mode, no connections are allowed. Error 14552 is returned when a client attempts to establish a connection to a solidDB server which is in a backup server mode (also called *netcopy listening mode*). The backup server mode is a special server mode where the solidDB instance has been started with the command line option `-xbackupserver`. This mode indicates that the solidDB instance is a Secondary server that is either waiting for or in the process of receiving the database file from the Primary server due to a **netcopy** command issued at the Primary server.

solidDB HotStandby errors

Table 32. solidDB HotStandby errors

Code	Class	Type	Description
14700	HotStandby	Error	Rejected connection, both servers in PRIMARY role. Meaning: Command **'hsb connect'** returns this error if both nodes are in same role.
14701	HotStandby	Error	Rejected connection, both servers in SECONDARY role. Meaning: Command **'hsb connect'** returns this error if both nodes are in same role.

Table 32. solidDB HotStandby errors (continued)

Code	Class	Type	Description
14702	HotStandby	Error	Operation failed, catchup is active. Meaning: While the servers are performing catchup, you will get this error if you issue any of the following commands on the Primary: **'hsb switch secondary'**, **'hsb set secondary alone'**, **'hsb set standalone'**, **'hsb connect'**, **'hsb copy'** or **'hsb netcopy'**. While the servers are performing catchup, you will get this error if you issue any of the following commands on the Secondary: **'hsb switch primary'**, **'hsb set secondary alone'**, **'hsb set primary alone'**, **'hsb set standalone'**, or **'hsb connect'**.
14703	HotStandby	Error	Operation failed, copy is active. Meaning: While the Primary is doing copy or netcopy, the following commands returns this error: **'hsb switch secondary'**, **'hsb set secondary alone'**, **'hsb set standalone'**, **'hsb connect'**, **'hsb disconnect'**, **'hsb copy'** or **'hsb netcopy'**.
14704	HotStandby	Error	HotStandby copy or netcopy is only allowed when primary is in alone state. Meaning: This error is returned if the server is in PRIMARY ACTIVE state and the command **'hsb copy'** or **'hsb netcopy'** is issued.
14705	HotStandby	Error	Setting to STANDALONE is not allowed in this state. Meaning: If the server is in PRIMARY ACTIVE state and you issue the command **'hsb set standalone'**, then you will get this message.
14706	HotStandby	Error	Invalid read thread mode for HotStandby, only mode 2 is supported.
14707	HotStandby	Error	Operation not allowed in the STANDALONE state.
14708	HotStandby	Error	Catchup failed, catchup position was not found from log files.
14709	HotStandby	Error	Hot Standby enabled, but connection string is not defined.
14710	HotStandby	Error	Hot Standby admin command conflict with an incoming admin command.
14711	HotStandby	Error	Failed because server is shutting down.
14712	HotStandby	Error	Server is secondary. Use primary server for this operation.

solidDB HSB errors and messages

Table 33. solidDB HSB errors and messages

Code	Class	Type	Description
14007	HSB	Message	CONNECTING
14008	HSB	Message	CATCHUP
14009	HSB	Message	No role switches since the server startup
14010	HSB	Message	DISCONNECTING
14522	HSB	Message	HotStandby copy directory not specified.
14537	HSB	Message	BROKEN
14704	HSB	Error	HotStandby copy or netcopy is only allowed when primary is in alone state
14712	HSB	Error	Server is secondary. Use primary server for this operation
30500	HSB	Message	Started as a HotStandby primary
30501	HSB	Message	Started as a HotStandby secondary
30502	HSB	Message	The database was not shut down properly the last time that it was used starting as a HotStandby secondary
30503	HSB	Message	Forcing HotStandby primary to start as a secondary
30504	HSB	Message	HotStandby role switched to secondary
30505	HSB	Message	HotStandby role switched to primary

Table 33. solidDB HSB errors and messages (continued)

Code	Class	Type	Description
30506	HSB	Message	Primary server must be set to PRIMARY ALONE or switched to the secondary role.
30507	HSB	Message	HotStandby server set to PRIMARY ALONE.
30508	HSB	Message	HotStandby server set to SECONDARY ALONE
30509	HSB	Message	HotStandby switch to primary failed, error *error_code*
30510	HSB	Message	HotStandby switch to secondary failed, error *error_code*
30511	HSB	Message	Failed to start HotStandby to *server_name*, error *error_code*
30512	HSB	Message	Failed to switch HotStandby role to primary, error *error_code*
30513	HSB	Message	Failed to switch HotStandby role to secondary, error *error_code*
30514	HSB	Message	Both databases are primary servers starting as a secondary
30515	HSB	Message	Both HotStandby databases are primaries
30516	HSB	Message	Failed to start HotStandby to *server_name*, other server rejected with error *error_code*
30517	HSB	Message	HotStandby role in secondary switched
30518	HSB	Message	HotStandby role switched to standalone
30530	HSB	Message	Starting to send HotStandby catchup data to secondary server
30531	HSB	Message	HotStandby catchup completed successfully
30532	HSB	Message	HotStandby catchup ended abnormally
30533	HSB	Message	HotStandby catchup can not be started. Secondary is not properly synchronized with primary full synchronization is required
30534	HSB	Message	HotStandby catchup ended abnormally, status *error_code*
30535	HSB	Message	HotStandby catchup ended abnormally, error *error_code*
30536	HSB	Message	HotStandby catchup ended abnormally due to a communication error
30537	HSB	Message	HotStandby catchup ended abnormally, secondary returned error *error_code*
30538	HSB	Message	HotStandby catchup size <value> greater than configured maximum size *value*, stopping HotStandby
30539	HSB	Message	File error in HotStandby catchup, stopping HotStandby
30540	HSB	Message	Starting to receive HotStandby catchup data from primary server
30541	HSB	Message	Secondary is not properly synchronized with primary due to a log file corruption. Restart secondary and execute a HSB netcopy.
30550	HSB	Message	Connection broken to HotStandby secondary server
30551	HSB	Message	Connected to HotStandby
30552	HSB	Message	HotStandby secondary connected
30553	HSB	Message	HotStandby primary connected
30554	HSB	Message	Hot Standby connection broken to Secondary server with an open transaction waiting for the operator to resolve transaction status. Primary server must be set to alone mode or switched to secondary mode.
30555	HSB	Message	HotStandby ping timeout
30556	HSB	Message	Connection broken to HotStandby secondary
30557	HSB	Message	HotStandby databases are not properly synchronized
30558	HSB	Message	HotStandby connection to secondary timed out
30559	HSB	Message	HotStandby connection broken
30560	HSB	Message	HotStandby: *HotStandby_error_message*
30561	HSB	Message	Started connecting to HotStandby
30562	HSB	Message	Connection broken to HotStandby primary server
30570	HSB	Message	Network backup completed.
30571	HSB	Message	Started to receive network backup.
30572	HSB	Message	Database started using a HotStandby copy/netcopy.

Table 33. solidDB HSB errors and messages (continued)

Code	Class	Type	Description
30573	HSB	Message	Network backup failed.
30574	HSB	Message	Hot Standby forcing threads to 1
30575	HSB	Message	Hot Standby replication configured but no active license found replication not started
30577	HSB	Message	HotStandby connect operation failed
30579	HSB	Message	HotStandby connection is already active.
30581	HSB	Message	Invalid event *event*
30582	HSB	Message	HotStandby cannot set the server to PRIMARY ALONE.
30583	HSB	Message	HotStandby copy failed.
30585	HSB	Message	Database starts to listen for netcopy.
30586	HSB	Message	HotStandby catchup, *catchup_phase* logpos: *log_position* *catchup_phase* can be: • HSB waitdurable • HSB catchup start • HSB write catchup • HSB write switch
30750	HSB	Message	HotStandby connection is already active.
30752	HSB	Message	Operation failed disconnect is active.
30757	HSB	Message	CONNECTED
30758	HSB	Message	Bad Hot Standby command.
30759	HSB	Message	HotStandby server is set to STANDALONE.
30760	HSB	Message	Started the process of disconnecting the servers.
30761	HSB	Message	Started the process of switching the role to primary.
30762	HSB	Message	Started the process of switching the role to secondary.
30763	HSB	Message	Started the process of connecting the servers.
30764	HSB	Message	Copy started.
30765	HSB	Message	Parameter **AutoPrimaryAlone** is set to Yes.
30766	HSB	Message	Parameter **AutoPrimaryAlone** is set to No.
30767	HSB	Message	Parameter **Connect** is set to *value*.
30768	HSB	Message	HotStandby connection is already broken.
30769	HSB	Message	Operation failed because connection between the servers is active.
30772	HSB	Message	Hot Standby node identifier must be defined in the ini file.
30774	HSB	Message	Server is already STANDALONE.
30775	HSB	Message	Parameter **CopyDirectory** is set to *value*.
30776	HSB	Message	Parameter **ConnectTimeout** is set to *value*.
30777	HSB	Message	Parameter **PingTimeout** is set to *value* milliseconds.
30779	HSB	Message	Hot Standby migration is active
30782	HSB	Message	Server is already set to primary alone.
30783	HSB	Message	Server is already set to secondary alone.
30784	HSB	Message	Parameter *parameter_name* is set to *value*.
30785	HSB	Message	Parameter *parameter_name* is set to *value*.
30786	HSB	Message	Parameter *parameter_name* is set to *value*.
30787	HSB	Fatal Error	pri_dologskip:bad type, log pos, log size This error refers to a failed operation on the HSB primary server. The error returns the failed operation and its location in the log, and the log size. Operations in the replication log are skipped.

Table 33. solidDB HSB errors and messages (continued)

Code	Class	Type	Description
30788	HSB	Fatal Error	pri_hsblogcopy_write:bad type, log pos, log size This error refers to a failed operation on the HSB primary server. The write to the replication log file fails. The error returns the failed operation and its location in the log, and the log size.
30789	HSB	Fatal Error	Failed to open hot standby replication log file.
30790	HSB	Fatal Error	Failed to allocate memory for HotStandby log. Max Log size is *logsize*. This error concerns a diskless database using HotStandby. In these systems, the HotStandby log is written to memory. This error is given if allocating more memory for the log file fails.
30791	HSB	Fatal Error	HotStandby:solhsby:bad type *type*, log pos *log_pos*, log size *log_size*
30792	HSB	Message	Both servers are secondary.
30793	HSB	Message	Maximum number of secondary tasks *value* reached. The queue at the secondary server for incoming log operations is growing faster than the operations can be executed and acknowledged to the primary server. The queue can be monitored with the performance counter **HSB secondary queues**.
30794	HSB	Message	Invalid **HotStandby.Connect** option **-z**. **-z** option is not supported.

B.2 High Availability Controller errors and status codes

solidDB High Availability Controller errors (17xxx) occur when using specific High Availability Controller commands.

Table 34. High Availability Controller errors and status codes

Error or status code	Description
17501	HAC is shutting down. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hac shutdown'** Meaning: The High Availability Controller is shutting down.
17502	Command failed, HAC is suspended. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hac suspend'** • **ADMIN COMMAND 'hac gethsbdstate'** • **ADMIN COMMAND 'hac getdbstate'** • **ADMIN COMMAND 'hac shutdowndb'** • **ADMIN COMMAND 'hac restartdb'** • **ADMIN COMMAND 'hac switchdb'** • **ADMIN COMMAND 'hac statemachinestate'** • **ADMIN COMMAND 'hac getereip'** • **ADMIN COMMAND 'hac pingere'** Meaning: The command execution failed and the High Availability Controller is suspended.

Table 34. High Availability Controller errors and status codes (continued)

Error or status code	Description
17503	Unknown command. Enter 'hac commands' for help. Meaning: Incorrect admin command syntax.
17504	HAC is already running. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hac resume'**
17506	HSB state does not allow for switchover. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hac switchdb'**
17507	Cannot execute command. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hac shutdowndb'**
17509	Restarting database server failed, see `solmsg.out` for details. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hac restartdb'**
17510	Cannot connect to database server. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hac switchdb'**
17511	Database server was not shut down. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hac restartdb'**
17513	Switchover failed. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hac switchdb'**
17514	ERE IP is not specified in the configuration file. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hac getereip'** • **ADMIN COMMAND 'hac pingere'**

Table 34. High Availability Controller errors and status codes (continued)

Error or status code	Description
17516	HAC is already active. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hac setactive'** • **ADMIN COMMAND 'hac pingere'**
17517	HAC is already passive. ADMIN COMMANDs that may return this status in the result set of the command: • **ADMIN COMMAND 'hac setpassive'**

B.3 solidDB database errors for HotStandby

solidDB database errors (10002 - 10050) are detected by solidDB and are sent to the client application. They may demand administrative actions.

This section lists the solidDB database errors that are related to HotStandby. A full list of the errors in the Database class is available in section *solidDB database errors* in the *IBM solidDB Administrator Guide*.

Table 35. solidDB database errors

Code	Class	Type	Description
10002	Database	Error	Operation failed. Meaning: The connect operation failed with an unexpected error. Most likely, the servers are not properly synchronized.
10013	Database	Error	Transaction is read-only. Meaning: You have tried to write inside a transaction that is set read-only, or the server is temporarily set to read-only mode, for example during the state switch. Updatable transactions are not allowed.
10019	Database	Error	Backup is already active. Meaning: You have tried to start a backup or copy when one is already in progress.
10024	Database	Error	Illegal backup directory "*directory_name*". Meaning: The backup or copy directory is either an empty string or a dot indicating that the backup or copy will be created in the current directory.
10030	Database	Error	Backup or copy directory '*directory_name*' does not exist. Meaning: Backup or copy directory is not found. Check the name of the backup or copy directory.

Table 35. solidDB database errors (continued)

Code	Class	Type	Description
10045	Database	Error	This operation cannot be executed on a HotStandby Secondary server. Meaning: This operation cannot be executed on a HotStandby Secondary server. In order for the requested operation to succeed, the server must be a Primary.
10046	Database	Error	Operation failed, data dictionary operation is active. Meaning: A data dictionary operation is currently in progress.
10047	Database	Error	Replicated transaction is aborted. Meaning: Transactions are aborted, for example, in a state switch. When the server state is switched from Primary to Secondary, all active transactions are aborted.
10048	Database	Error	Replicated transaction contains data dictionary changes, normal update operations are not allowed. Meaning: HotStandby mode restricts data dictionary operations; for example, CREATE TABLE cannot be mixed with normal update operations. This message is obsolete in version 4.1 and later, which allow you to mix DML and DDL operations within a transaction while using HSB.
10049	Database	Error	The remote server is not a secondary server. Meaning: The server that you specified in the command is not in a SECONDARY state.
10050	Database	Error	Replicated operation updated BLOB columns. Meaning: BLOB columns cannot be replicated to the Secondary server.
10078	Database	Error	User rolled back the transaction.
10079	Database	Error	Cannot remove filespec. File is already in use.
10080	Database	Error	HotStandby Secondary server can not execute operation received from Primary server. Meaning: A possible cause for this error is that the database did not originate from the Primary server using HotStandby **copy** or **netcopy** command.
10081	Database	Error	The database file is incomplete or corrupt. Meaning: If the file is on a hot standby secondary server, use the **hotstandby copy** or **hotstandby netcopy** command to send the file from the primary server again.
10082	Database	Error	Backup aborted.
10083	Database	Error	Failed to abort HSB transaction because commit is already sent to secondary.
10084	Database	Error	Table is not locked.
10085	Database	Error	Checkpointing is disabled.
10087	Database	Error	HotStandby not allowed for main memory tables.

Table 35. solidDB database errors (continued)

Code	Class	Type	Description
10088	Database	Error	Specified lock timeout is too large.
10089	Database	Error	Operation failed, server is in HSB primary uncertain mode.

B.4 solidDB table errors

solidDB database table errors are caused by erroneous SQL statements and are detected by solidDB. Administrative actions are not needed.

This section lists the solidDB table errors that are related to HotStandby. A full list of the errors in the Table class is available in section *solidDB table errors* in the *IBM solidDB Administrator Guide*.

Table 36. solidDB table errors

Code	Class	Type	Description
13068	Table	Error	Server shutdown in progress Meaning: You are unable to complete this operation because server shutdown is in progress.
13123	Table	Error	Table '*table_name*' is not empty Meaning: This operation can only be executed when a table is empty. For example, you can only change a table from disk-based to in-memory (or vice-versa) when the table is empty.
13167	Table	Error	Only M-tables can be transient Meaning: You cannot create a transient table that is disk-based. For example, the following SQL statement will get this error message: `CREATE TRANSIENT TABLE t1 (i INT) STORE DISK;`
13170	Table	Error	Only M-tables can be temporary Meaning: You cannot create a temporary table that is disk-based. For example, the following SQL statement will get this error message: `CREATE TEMPORARY TABLE t1 (i INT) STORE DISK;`

B.5 solidDB communication errors

solidDB communication errors (21306, 21308) are caused by network errors. These errors demand administrative actions.

This section lists the solidDB communication errors that are related to HotStandby. A full list of the errors in the Server class is available in section *solidDB communication errors* in the *IBM solidDB Administrator Guide*.

Table 37. solidDB communication errors

Code	Class	Type	Description
21306	Communication	Error	Server "*server_name*" not found, connection failed. Meaning: The Secondary server was not found. • Check that the server is running. • Check that the network name is valid. • Check that the server is listening to the specified network name.
21308	Communication	Error	"Connection is broken (<operation> operation failed with code <code>)". For example, `"Connection is broken (TCP/IP 'Write' operation failed with code 7)."` Recommended actions: • Check that the server is running. • Check that the network name is valid. • Check that the server is listening to the specified network name.

Appendix C. HotStandby and HAC ADMIN COMMANDs

This section describes the ADMIN COMMANDs available with HotStandby.

Depending on the tool you are using, the ADMIN COMMAND syntax differs:

- **solidDB SQL Editor (solsql)**

  When used with **solsql**, the command name must be given with single quotation marks. For example:

  ```
  ADMIN COMMAND 'hotstandby switch primary';
  ADMIN COMMAND 'hacontroller shutdown';
  ```

- **solidDB Remote Control (solcon)**

  When used with **solcon**, the command name must be given without the ADMIN COMMAND prefix and quotation marks. For example:

  ```
  hotstandby switch primary
  hacontroller shutdown
  ```

Tip: You can abbreviate 'hotstandby' to 'hsb' and 'hacontroller' to 'hac'. For example:

```
ADMIN COMMAND hsb switch primary
ADMIN COMMAND hac shutdown
```

In the syntax descriptions, the shortest possible form is used; the ADMIN COMMAND and the quotation marks are omitted, and the abbreviations 'hsb' and 'hac' are used.

Note: ADMIN COMMANDs always return a success return code (0) if there is no syntax error in the command. The actual result code of the command is included in the "RC" field of the result set.

For more information about the ADMIN COMMAND, see section *ADMIN COMMAND* in the *IBM solidDB SQL Guide*.

C.1 HotStandby commands (ADMIN COMMAND)

Table 38. HotStandby commands (ADMIN COMMAND)

Command	Explanation
hsb cominfo	Returns the connect string used to connect to the other server. This is usually the value of the **HotStandby.Connect** parameter, but it might also have been set with the command: `ADMIN COMMAND 'hsb parameter connect connect_string';` You can use this information in an application to connect to other servers.

Table 38. HotStandby commands (ADMIN COMMAND) (continued)

Command	Explanation
hsb connect	If the connection between the Primary and Secondary servers is broken or has not yet been established, this command connects the Primary server to the Secondary server and starts HotStandby replication. This command is always needed to connect the servers since there is no automatic mechanism for connecting between servers. After a successful connect, the state of the Primary server is automatically set from PRIMARY ALONE to PRIMARY ACTIVE. If unsuccessful, the state remains PRIMARY ALONE. This command can be executed on either the Primary or the Secondary. **Note:** When you execute this command, if the Primary server and Secondary server are connected, but the transaction log is not yet fully copied to the Secondary, the following message is displayed: `Catchup is active`.
hsb copy [*directory_name*]	**Important:** This command is deprecated. Use the **hsb netcopy** command instead. You can use the **hsb copy** command to initially create the Secondary database from the Primary. This command copies the database into a directory that is local to the Primary node (and also local to the Secondary node). After the copy is completed, you may start the Secondary server. After you connect the Primary to the Secondary, the Primary automatically brings the Secondary server up-to-date by copying the transaction log to the Secondary server. You can also use this command to synchronize the Primary database with a Secondary database (when it has been offline for a considerable period of time) that is in a directory local to the Primary node. Read 3.4.5, "Synchronizing primary and secondary servers," on page 50. If the optional *directory_name* is specified, the database files are copied to that directory; otherwise they are copied to the directory specified with the **copydirectory** parameter in the `[Hotstandby]` section of the `solid.ini` configuration file. Because the **hsb copy** command does not copy the `solid.ini` configuration file or log files, it is recommended that you make this directory different from the normal backup directory. The Primary can execute the **hsb copy** command only if the Primary is in PRIMARY ALONE state. During and after the command, the server remains in PRIMARY ALONE state. After the command has been completed, you may start the Secondary server and connect the two servers.
hsb disconnect	This tells the server to disconnect gracefully from the other member of the HSB pair. This command is valid on either the Primary or the Secondary server. A typical reason to use this command is to disconnect the servers before upgrading one of them. (The other server can be set to PRIMARY ALONE state so that it can continue responding to client requests.) This command normally causes both servers to go into an "Alone" mode; that is, the Primary server switches from PRIMARY ACTIVE to PRIMARY ALONE, while the Secondary server switches from SECONDARY ACTIVE to SECONDARY ALONE. This command is valid on both the Primary and the Secondary. **Note:** Using the shutdown command `ADMIN COMMAND 'shutdown'` causes the server to do a controlled disconnect before it shuts down. If the Secondary is shut down (and disconnects), the Primary knows that it is safe to go to PRIMARY ALONE state, and will do so.

Table 38. HotStandby commands (ADMIN COMMAND) (continued)

Command	Explanation

hsb logpos	This commands returns the log operation ID and the server's role (PRIMARY, SECONDARY, STANDALONE) at the time of the last operation.
	
	A typical output is shown below:
	`ADMIN COMMAND 'hsb logpos';`
	`RC TEXT`
	`-- ----`
	`0 00000000000000000871:PRIMARY`
	
	This command can be used to determine which of two servers should become Primary, for example, after a failure where both databases have failed and you do not know which server should be made the Primary. (The server that was the Primary before the connection was broken is not necessarily the server that should become the Primary now.)
	
	In principle, the server that has the greater value for the log operation ID has accepted more transactions, and thus should become the Primary. However, if you have made updates to both servers after the HSB connection has failed, the log operation ID values can no longer be compared reliably.
	
	For more information about how to use this command, see 3.4.9, "Choosing which server to make primary," on page 63.

hsb netcopy	This command can be used copy the Primary database or diskless in-memory data to a Secondary server. The database files are copied through the network link, using the connection defined with the **HotStandby.Connect** parameter in the Primary's `solid.ini` file. You might want to use this command, for example, to synchronize a Primary database with a Secondary database that has been offline for a long time. You can also use **hsb netcopy** to create a new Secondary database, for example, to replace a corrupt Secondary database, to set up a Secondary database for a new HotStandby configuration, or add a new Secondary to an existing configuration. The Primary server must be in PRIMARY ALONE state to issue this command. After the command has completed (successfully or unsuccessfully), the Primary server remains in the PRIMARY ALONE state. If the copy is completed successfully, the Secondary server is automatically switched to SECONDARY ALONE state. **Tip:** The **hsb netcopy** command is usually followed by the **hsb connect** command to connect the Primary and Secondary servers. After the Primary server is connected to the Secondary, the Primary automatically brings the Secondary up-to-date by copying the transaction log. For more details on using **hsb netcopy**, see 3.4.5, "Synchronizing primary and secondary servers," on page 50 and "Copying a primary database to a secondary over the network" on page 54.

Table 38. HotStandby commands (ADMIN COMMAND) (continued)

Command	Explanation

hsb parameter	**Important:** This command is deprecated.
	
	This command allows you to set HSB-specific parameters such as **AutoPrimaryAlone**, **Connect**, and **PingTimeout**. For a complete description of each of these parameters, see Appendix A, "HotStandby configuration parameters," on page 117.
	
	When you set the value of one of some parameters, the command takes effect immediately, but is not written to the `solid.ini` configuration file before a shutdown is executed.
	
	The syntax for this command is:
	`ADMIN COMMAND 'hsb parameter`
	`param_name param_value';`
	
	This command does not use an equals sign. Thus it differs from the otherwise similar command (recommended):
	`ADMIN COMMAND 'parameter`
	`hotstandby.param_name = param_value';`

Table 38. HotStandby commands (ADMIN COMMAND) (continued)

Command	Explanation
hsb role	**Important:** This command is deprecated. Use the **hsb state** command instead. Returns one of the following roles in the result set: • PRIMARY, if the connected server is a normal Primary server. In this role, the transactions at the Primary server are sent to the Secondary server. • PRIMARY NOHSBLOG, indicating that the Primary server accepts transactions and stores them in the database. However, it does not store the transactions in a log so that it could later send them to the Secondary. To resynchronize the Secondary with the Primary, the entire database at the Primary must be copied to the Secondary server. • PRIMARY BROKEN, if the Primary server has a broken connection to the Secondary server. Only read-only transactions can be executed in the Primary server. • PRIMARY ALONE, if the Primary server is working by itself. The connection to the Secondary is broken, but transactions are accepted and added to the transaction log at the Primary so that later they can be sent to the Secondary. • PRIMARY CATCHUP, if the catchup is in progress. During catchup, the Primary automatically sends the transaction log changes to the Secondary server after the 'hsb connect' command has been issued at the Primary. After the catchup process is completed, the role of the server is switched automatically to PRIMARY. The Primary can continue to accept transactions if its role was PRIMARY ALONE before the connect. • SECONDARY, if the connected server is a normal Secondary server. This means the server receives and applies transactions from the Primary. • SECONDARY BROKEN, if the Secondary server has a broken connection to the Primary server. • SECONDARY CATCHUP, if the Secondary server is catching up with the changes from the Primary server after the '**hsb connect**' command was issued at the Primary server. After the catchup process is completed, the role of the Secondary is switched automatically to SECONDARY. If you issue **hsb role** on a server that is not configured for HotStandby, the following error message is returned: `14527: This is a non-HotStandby Server`. This command returns the same information as the SQL function: HOTSTANDBY_ROLE.

Table 38. HotStandby commands (ADMIN COMMAND) (continued)

Command	Explanation
hsb set primary alone	This command sets the Primary server to the PRIMARY ALONE state unconditionally. This command is available if the server is currently in one of the following states: PRIMARY ACTIVE, SECONDARY ACTIVE, SECONDARY ALONE and STANDALONE. This command can be used to implement fast failovers. When the Secondary is in the SECONDARY ACTIVE state, the server will not make any attempt to communicate with the Primary, having received this command. Instead, it will immediately switch to the PRIMARY ALONE state. This behavior may be utilized in cases when the information about the Primary failure reaches HAC before the Secondary has detected the failure (the delay is dictated by the **PingTimeout** and **PingInterval** parameters). However, if it happens (for example, because of incorrect failure detection) that the Primary is "alive" and in the PRIMARY ACTIVE state when this command is executed in the Secondary, the Primary will be automatically forced to PRIMARY UNCERTAIN state. It can then be moved to the SECONDARY ALONE state and reconnected without any loss of transactions. **Note:** The alternative way of executing failovers is to use the **hsb switch primary** command. In the PRIMARY ALONE state, the connection to the Secondary server is broken, but this state allows the Primary server to run with continuous updates to the transaction log. The PRIMARY ALONE state persists until the Primary server is shut down, a connection is successfully made to the Secondary server, or the server runs out of space for the transaction log. When you set a server to PRIMARY ALONE state, it does not automatically make any attempt to re-establish connections with the other server. **Important:** Before executing this command on a server, make sure that the other server in the pair is not already in PRIMARY ALONE state (or STANDALONE state). This is to avoid *dual primaries* – see 3.6.2, “Network partitions and dual primaries,” on page 66 for more details. See also the command **'hsb switch primary'**.
hsb set secondary alone	This command sets the server state to SECONDARY ALONE. This command is available if the server is currently in one of the following states: PRIMARY ALONE, PRIMARY UNCERTAIN, STANDALONE.
hsb set standalone	When this command is issued, the state of the Primary server becomes STANDALONE. The server stops storing transactions for the Secondary server. The Primary (STANDALONE) can continue accepting read/write transactions. This option is useful in the Primary server when the Secondary server is offline for a significant period of time and the transaction log may grow too large. This command is available if the server is currently in one of the following states: PRIMARY ALONE or SECONDARY ALONE.

Table 38. HotStandby commands (ADMIN COMMAND) (continued)

Command	Explanation
hsb state	Returns the state of the server: • PRIMARY ACTIVE, if the connected server is a normal Primary server. In this state, transactions on the Primary server are sent to the Secondary server. • STANDALONE, indicating that the Primary server accepts transactions and stores them in the database, but it does not store the transactions to forward them to the Secondary. • PRIMARY UNCERTAIN, if the Primary server has a broken connection to the Secondary server and has not yet been switched to another state, such as PRIMARY ALONE. Only read-only transactions can be executed in the Primary server. • PRIMARY ALONE, if the Primary server is working by itself. The connection to the Secondary is broken, but transactions are accepted and stored in the Primary's transaction log so that they can be forwarded to the Secondary. • SECONDARY ACTIVE, if the connected server is a normal Secondary server. This means the server receives and applies transactions from the Primary. • SECONDARY ALONE, if the Secondary server has a broken connection to the Primary server. If **ADMIN COMMAND 'hsb state'** is issued on a server that is not configured for HotStandby, the following error message is returned: `14527: This is a non-HotStandby Server`. This command returns the same information as the SQL function: HOTSTANDBY_STATE. See section *Using Function HOTSTANDBY_STATE* in "Switching the application to the new primary" on page 95 for details on this function. See Appendix D, "Server state transitions," on page 161 for an overview of HotStandby state transitions that occur while performing administrative and troubleshooting operations.
hsb status *option*	This command returns HotStandby status information of the last successfully started operation. The option may be any of the following: • `catchup` • `connect` • `copy` • `switch` For more details, see the descriptions of the individual commands/options below, for example, **hsb status catchup**. The status command gives information about the outcome of operations that take a prolonged time, after they have started successfully. If the starting of operation fails (for example, because of incorrect state) the status command will not return the status of that operation but the one executed previously.
hsb status catchup	This command indicates whether or not the server is doing catchup, that is, when the Secondary reads the Primary's transaction log and applies the changes. Possible return values are: • ACTIVE • NOT ACTIVE

Table 38. HotStandby commands (ADMIN COMMAND) (continued)

Command	Explanation
hsb status connect	This command returns status information: • CONNECTED - Connect active. This information is returned from both the Primary and Secondary servers. • CONNECTING - The Primary server and Secondary server are connecting to each other. This information is returned from both the Primary and Secondary servers. • CATCHUP - The Primary server is connected to the Secondary server, but the Primary HotStandby database log is not fully copied to the Secondary server. This information is returned from both the Primary and Secondary servers. • BROKEN - Connection between the Primary and Secondary server is broken. This information is returned from both the Primary and Secondary servers. **Note:** This command returns the same information as the SQL function HOTSTANDBY_CONNECTSTATUS. See section *Using function HOTSTANDBY_CONNECTSTATUS* in "Switching the application to the new primary" on page 95 for details on this function.
hsb status copy	This command allows you to check the result of the last **hsb copy** or **hsb netcopy** command. This status command always uses the keyword `copy`, even if you are checking the result of a **hsb netcopy** rather than a **hsb copy**. Status information returned: • SUCCESS - Copy completed successfully. • ACTIVE - Copy process is still active. • ERROR *number* - Copy failed with error code *number*.
hsb status switch	This command returns HotStandby switch status information: • ACTIVE - Copy process is still active. • SUCCESS - Copy completed successfully. • ERROR *number* - Copy failed with error code *number*. • NO SERVER SWITCH OCCURRED BEFORE - No switch has happened before.
hsb switch primary	This command switches the database server to PRIMARY. The command starts a switch process, which can be monitored using command **hsb status switch**. If the servers are connected at the time that you execute this command, the servers simply reverse states — that is, the old Primary changes from PRIMARY ACTIVE to SECONDARY ACTIVE, and the Secondary server switches from SECONDARY ACTIVE to PRIMARY ACTIVE. If the servers are not connected and the server is in SECONDARY ALONE state, when you switch the server to Primary, it will end up in PRIMARY ALONE state. The new Primary server will not automatically try to connect to the other server and switch to PRIMARY ACTIVE state. Because the command is available both in the SECONDARY ACTIVE and SECONDARY ALONE states, it can be used to perform failovers. However, because the server will always make attempt to communicate with the Primary, the network timeout may be involved. Thus, this method is slower than using the **hsb set primary alone** command. On the other hand, this method secures better against a possibility of *dual primaries*. See also the command **hsb set primary alone**.

Table 38. HotStandby commands (ADMIN COMMAND) (continued)

Command	Explanation
hsb switch secondary	This command switches the database to SECONDARY state. All active write transactions are terminated. **Note:** If the connected database server is a Primary server, it becomes a Secondary server. If the old Secondary server is available, the old Secondary server is switched to Primary (see the **hsb switch primary** command). **Note:** If the **hsb switch secondary** command is issued inside an open transaction (in Windows environments, after the transaction has started and before you execute the COMMIT statement), when you issue the COMMIT statement, it fails with an error: `replicated transaction is aborted`. All transactions are rolled back during the switch, including the transaction in which the switch statement is executed. The switch itself is successful (that is, it is not rolled back) because ADMIN COMMANDs are not transactional commands. However, administrative commands force the start of a new transaction if one is not already open.

C.2 High Availability Controller commands (ADMIN COMMAND)

To issue the HAC commands, you must first connect to HAC using the port defined with the **HAController.Listen** parameter in the `solidhac.ini`. You can use, for example, **solsql** or the ODBC interface to connect to HAC.

Table 39. High Availability Controller commands (ADMIN COMMAND)

Command	Explanation
hac restartdb Abbreviation: **hac rsd**	If the solidDB server process has been terminated using the **hac shutdowndb** command, this command instructs the HAC to restart the solidDB server process.
hac resume Abbreviation: **hac rs**	This command resumes the HAC operations when HAC is in suspended mode (set with **hac suspend**). The `solidhac.ini` is not reread, that is, any changes to `solidhac.ini` are not effective.
hac setadministrative Abbreviation: **hac sad**	This command sets HAC to ADMINISTRATIVE mode.
hac setautomatic Abbreviation: **hac sam**	This command sets HAC to AUTOMATIC mode.
hac shutdown Abbreviation: **hac sd**	This command terminates the HAC process.
hac shutdowndb Abbreviation: **hac sdd**	This command instructs the HAC to shutdown the solidDB server process. When the server process is terminated using this command, HAC does not try to restart the server process. To restart the server process, use the **hac restartdb** command.
hac suspend Abbreviation: **hac sp**	This command shuts down all HAC operations (threads), expect for the thread that HAC uses to listen for ADMIN COMMANDs. When HAC has been suspended, the operations can be resumed with ADMIN COMMAND **hac resume**.

Table 39. High Availability Controller commands (ADMIN COMMAND) (continued)

Command	Explanation
trace { on | off } hac Abbreviation: **tra { on | off } hac**	This command controls tracing of HAC operations. The trace information is output to `hactrace.out` in the HAC working directory. In general, IBM Software Support and development teams use HAC traces facility for troubleshooting. You can also generate the trace to gain information about a problem that you are investigating, but its use is rather limited without knowledge of the solidDB source code. Using the HAC trace facility has minimal impact on performance.

Appendix D. Server state transitions

This chapter describes the possible state transitions (for example, the transition from OFFLINE to SECONDARY ALONE).

A description of each of the server states is in “Description of server states” on page 8.

D.1 HotStandby state transition diagram

The diagram in this section shows the state transitions that can occur, and the circumstances under which they may occur.

For example, you can change the state of a server from PRIMARY UNCERTAIN to PRIMARY ALONE by executing the command **'hsb set primary alone'**:

```
ADMIN COMMAND 'hsb Set Primary Alone';
```

As you use this diagram, remember the following:

1. The complete syntax of the commands is not shown. For example, it shows:

   ```
   'hsb set primary alone'
   ```

   rather than

   ```
   ADMIN COMMAND 'hsb Set Primary Alone';
   ```
2. The state transition paths shown for **'hsb copy'** also apply to **'hsb netcopy'**.
3. Some commands may fail when they are executed. When a command might succeed or fail, both possibilities are shown. If the branch is intended to describe what happens if the command fails, it will have the word 'failed': `'Disconnect' failed.`
4. In some situations, the behavior depends upon the setting of the `solid.ini` configuration parameter named **AutoPrimaryAlone**. The abbreviation "APA" is often used to represent this parameter.
5. When the diagram refers to "events", it refers to internally-generated notifications. These are not the same as the "events" that users can post and wait on, as described in the SQL commands for CREATE EVENT, for example.
6. Near the top left of the diagram, you will see the text `"Start with '-x backupserver'"`. If you want to start a new Secondary server and you want it to get a copy of the database from the Primary via the "**netcopy**" command, then you start the server (from the operating system command line) with the command-line option **-x backupserver**. This tells the server to wait for a **netcopy** from the Primary. Note that while the server is waiting to receive the **netcopy**, the server will not respond to queries about its state (or role). For example, if you issue the command:

   ```
   ADMIN COMMAND 'hsb state';
   ```

   the server will not respond and therefore you will not actually see it return the state "OFFLINE".
7. "rpc" stands for "Remote Procedure Call". "rpc broken" means that the Primary and Secondary lost connection with each other without doing an explicit Disconnect. The connection may be lost if the network fails, or if one server crashes, for example.

8. When an arrow loops back to the same state that it started from, it means that the state does not change. For example, if a server is in the state PRIMARY ALONE, and if it tries to connect to the other server but fails, then the state remains PRIMARY ALONE.

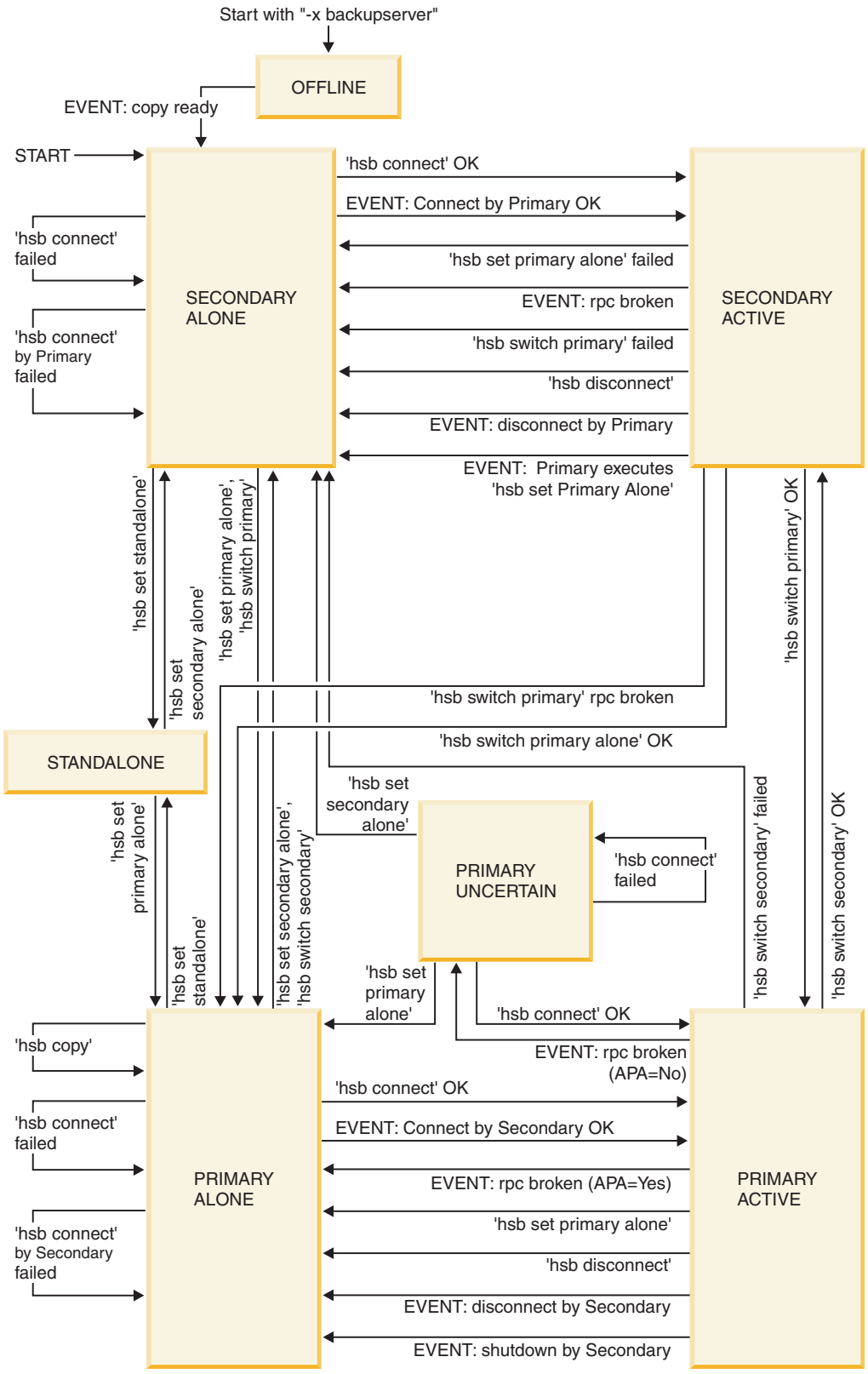


Figure 19. HotStandby server state transitions

The following table shows server states and the ways in which a HotStandby command can change the server state.

Table 40. Server state transition table

Server state	If this condition occurs, or if this HSB command is issued...	Then server state becomes...	If command is unsuccessful, then the state is...
OFFLINE	If the Primary server executes **ADMIN COMMAND 'hotstandby netcopy'** then the Secondary's state will change to SECONDARY ALONE after the database has been copied.	SECONDARY ALONE	Unchanged
PRIMARY ACTIVE	HotStandby timeout (automatic) when **AutoPrimaryAlone** = Yes. NOTE: The HSB timeout occurs automatically when the Secondary server is down or a connection between the Primary and Secondary is broken.	PRIMARY ALONE	(Not applicable)
PRIMARY ACTIVE	HotStandby timeout (automatic) when **AutoPrimaryAlone** = No. NOTE: The HSB timeout occurs automatically when the Secondary server is down or a connection between the Primary and Secondary is broken.	PRIMARY UNCERTAIN	(Not applicable)
PRIMARY ACTIVE	**ADMIN COMMAND 'hotstandby set standalone'** at the Primary	STANDALONE	Unchanged
PRIMARY ACTIVE	**ADMIN COMMAND 'hotstandby switch secondary'** at the Primary or **ADMIN COMMAND 'hotstandby switch primary'** at the Secondary.	SECONDARY ACTIVE	SECONDARY ALONE
PRIMARY ACTIVE	**ADMIN COMMAND 'hotstandby disconnect'** at the Primary.	PRIMARY ALONE	PRIMARY ALONE

Table 40. Server state transition table (continued)

Server state	If this condition occurs, or if this HSB command is issued...	Then server state becomes...	If command is unsuccessful, then the state is...
PRIMARY ALONE	**ADMIN COMMAND 'hotstandby copy'** or **ADMIN COMMAND 'hotstandby netcopy'** at the Primary. Note that the state of the Primary server does not change. The server stays in PRIMARY ALONE state. To change the state to PRIMARY ACTIVE, you must issue the "connect" command: **ADMIN COMMAND 'hotstandby connect';** NOTE: If you are using a diskless server without file access to the Secondary server, you must use netcopy, not copy.	PRIMARY ALONE	PRIMARY ALONE
PRIMARY ALONE	**ADMIN COMMAND 'hotstandby connect'** at the Primary NOTE: The above command is used to connect to the Secondary server, which is now fixed, or a server other than the failed Secondary.	PRIMARY ACTIVE (after the catchup is completed)	Unchanged
PRIMARY ALONE	**ADMIN COMMAND 'hotstandby set standalone'** at the Primary or the transaction log is full.	STANDALONE	Unchanged
PRIMARY ALONE	**ADMIN COMMAND 'hotstandby set secondary alone'** or **ADMIN COMMAND 'hotstandby switch secondary'** at the Primary.	SECONDARY ALONE	SECONDARY ALONE

Table 40. Server state transition table (continued)

Server state	If this condition occurs, or if this HSB command is issued...	Then server state becomes...	If command is unsuccessful, then the state is...
PRIMARY UNCERTAIN	**`ADMIN COMMAND 'hotstandby set primary alone'`** at the Primary server	PRIMARY ALONE	Unchanged
PRIMARY UNCERTAIN	**`ADMIN COMMAND 'hotstandby connect'`** at the Primary. **Note:** The above command is used to connect to the Secondary server (which is now fixed) or to connect to a server other than the failed Secondary.	PRIMARY ACTIVE	Unchanged
PRIMARY UNCERTAIN (HSB timeout has occurred for connecting to the Secondary)	**`ADMIN COMMAND 'hotstandby set standalone'`** at the Primary	STANDALONE	Unchanged
PRIMARY UNCERTAIN	**`ADMIN COMMAND 'hotstandby set secondary alone'`** or **`ADMIN COMMAND 'hotstandby switch secondary'`** at the Primary.	SECONDARY ALONE	Unchanged
SECONDARY ACTIVE	HotStandby timeout (automatic) **Note:** The HSB timeout occurs automatically when the Secondary server is down or a connection between the Primary and Secondary is broken.	SECONDARY ALONE	(not applicable)
SECONDARY ACTIVE	**`ADMIN COMMAND 'hotstandby switch secondary'`** at the Primary or **`ADMIN COMMAND 'hotstandby switch primary'`** at the Secondary.	PRIMARY ACTIVE	Unchanged
SECONDARY ACTIVE	**`ADMIN COMMAND 'hotstandby set primary alone'`** at the Secondary.	PRIMARY ALONE	Unchanged

Table 40. Server state transition table (continued)

Server state	If this condition occurs, or if this HSB command is issued...	Then server state becomes...	If command is unsuccessful, then the state is...
SECONDARY ACTIVE	**ADMIN COMMAND 'hotstandby disconnect'** at the Secondary or Primary.	SECONDARY ALONE	SECONDARY ALONE
SECONDARY ALONE	**ADMIN COMMAND 'hotstandby connect'** at the Secondary or Primary	SECONDARY ACTIVE	Unchanged
SECONDARY ALONE	**ADMIN COMMAND 'hotstandby set standalone'** at the Secondary.	STANDALONE	Unchanged
SECONDARY ALONE	**ADMIN COMMAND 'hotstandby set primary alone'** or **ADMIN COMMAND 'hotstandby switch primary'** at the Secondary	PRIMARY ALONE	Unchanged

Appendix E. HotStandby system events

This appendix covers only HSB-specific events. For a discussion of other types of events, see other manuals, such as *IBM solidDB SQL Guide*.

Each HotStandby operation generates an event. To monitor these events you can use an application, such as a watchdog application.

Events are objects with a name that signal that a specific action occurred in the server. Special statements in stored procedures are required to receive events. HotStandby events are no different from other events created and supported bysolidDB. They are sent to those users who are registered to receive the event in a stored procedure. For details on posting, registering, and waiting for events, read "Stored Procedures, Events, Triggers, and Sequences", in *IBM solidDB SQL Guide*, and solidDB SQL Syntax, also in *IBM solidDB SQL Guide*.

The following table lists the events that are currently available for HotStandby. Note that most events include five parameters, but not all of those parameters are necessarily used.

Table 41. HotStandby events

HSB Event	Event parameters	Cause of event

SYS_EVENT_HSBCONNECTSTATUS	ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER, TEXTDATA WVARCHAR
	
	For TEXTDATA, the possible valid values are:
	`TEXTDATA = {`
	`CONNECTED |`
	`CONNECTING |`
	`CATCHUP |`
	`BROKEN}`
		Change in connect status between the Primary and Secondary server

SYS_EVENT_HSBSTATESWITCH	ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER, TEXTDATA WVARCHAR
	
	For TEXTDATA, the possible valid values are:
	`TEXTDATA = {`
	`PRIMARY ACTIVE |`
	`PRIMARY ALONE |`
	`PRIMARY UNCERTAIN |`
	`SECONDARY ACTIVE |`
	`SECONDARY ALONE |`
	`STANDALONE`
	`}`
		Each state switch sends a state switch event.

Table 41. HotStandby events (continued)

HSB Event	Event parameters	Cause of event
SYS_EVENT_NETCOPYEND	ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER, TEXTDATA WVARCHAR None of the parameters are used.	HotStandby NETCOPY operation ended. This event can be caught by the user *only* if the user is using shared memory access or linked library access.
SYS_EVENT_NETCOPYREQ	ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER, TEXTDATA WVARCHAR None of the parameters are used.	A HotStandby NETCOPY was requested. If the user application's callback function returns non-zero, then netcopy is not performed. This event can be caught by the user *only* if the user is using shared memory access or linked library access.

Appendix F. Watchdog sample

This section discusses the Watchdog sample application available in the samples included in your solidDB installation.

A watchdog is a separate program for monitoring and controlling Primary and Secondary servers. The watchdog monitors both hot standby servers and switches their states when necessary. This alleviates the need for a database administrator to monitor the servers.

solidDB provides a sample watchdog that you can use as a basis for building a custom watchdog that meets your needs. This sample application is called the Watchdog. Pay attention to the following features of the Watchdog sample before you start programming.

- The Watchdog is meant to be used as an example of a watchdog
- The Watchdog uses polling to keep itself up-to-date with server states
- The Watchdog is a one-thread program
- The Watchdog uses the HSB API through ODBC. This API implementation can be used as a model for your own watchdog application.
- The Watchdog has no user interface.

If you are using the Watchdog, you need to configure a `[WatchDog]` section in the solidDB configuration file (`solid.ini`), which resides in the current working directory of the Watchdog. If the Watchdog is running in the same directory as the Primary or Secondary server, then you will have only one `solid.ini` file, which will be shared by the server and the Watchdog. If the Watchdog is running in a separate directory, then the Watchdog will have its own `solid.ini` file.

Furthermore, this appendix contains explanations of `solid.ini` configuration parameters that are specific to the Watchdog. These parameters are set in the `[WatchDog]` section of the `solid.ini` configuration file. If you write your own watchdog program, you do not need to use any of these parameters.

For a discussion of other `solid.ini` parameters, see *IBM solidDB Administrator Guide*.

F.1 HotStandby configuration using Watchdog

A HotStandby configuration allows for a Primary server, Secondary server, and the Watchdog to reside in different machines and use different operating systems and APIs as shown in the example in F.1.2, "System design issues," on page 173. For details on implementing heterogeneous configurations, read F.1.2, "System design issues," on page 173.

All communication between the Primary and Secondary database (including putting a failed system back in service and re-synchronizing Primary and Secondary databases) occurs within existing communication layers, such as TCP/IP. HotStandby requires no auxiliary storage or transfer methods, such as shared disks or FTP transfers.

Important: If you are running the Watchdog on the same machine where the Secondary server resides, be sure to set the parameter **`AutoPrimaryAlone`** to `no`. In

this situation, setting **AutoPrimaryAlone** to no is crucial because it prevents the potential error of having two primary servers. Primary may be in the PRIMARY ALONE state, and the Watchdog at server failure could switch Secondary to a PRIMARY ALONE state. This same error can also occur if a user happens to set the old Secondary server to become the new Primary. For more information about dual primaries, see 3.6.2, "Network partitions and dual primaries," on page 66.

F.1.1 How the Watchdog application works

The Watchdog sample application notifies you when the Primary server is down. In normal mode, the Watchdog checks the connection status of servers using the **hotstandby status connect** command in both Primary and Secondary servers.

The Watchdog performs this check between servers at regular intervals. The interval time is set with the **PingInterval** parameter in the Watchdog's solid.ini configuration file.

The Watchdog reaches the conclusion that there is a problem in the HotStandby system when it receives no response from the Primary or Secondary node or both nodes after a given number of polling attempts. The number of attempts is set in the **NumRetry** parameter in the Watchdog configuration file (the [Watchdog] section in the solid.ini).

The Watchdog also observes whether the Primary server and the Secondary server are connected to each other. If the Primary or Secondary server returns a successful connect status to the Watchdog, this means the Primary and Secondary are still connected. If it returns an error, on the other hand, then the Primary and Secondary are no longer connected.

If the **AutoSwitch** parameter in the Watchdog configuration file is set to YES, then the Watchdog is also responsible for automatically switching server states in the event of a Primary failure. For example, when the Primary server is down, the Watchdog switches the Secondary server to make it the new Primary and put it in PRIMARY ALONE state. If the **AutoSwitch** parameter is set to NO, the Watchdog does not change the server state itself, but instead writes a message to the Watchdog log to notify the user to switch server states.

To continue monitoring, the Watchdog switches to failure mode, which means it continuously keeps checking failed servers for a working connection.

Failure mode

When the Watchdog sample application knows that HotStandby Primary and Secondary servers are connected, the Watchdog stays in normal mode. If one of the servers fails, or if the communication link between these servers fails, the Watchdog will take some course of action. If the action fails to connect the servers, the Watchdog goes into failure mode.

After the Watchdog enters failure mode, the Watchdog waits for the system administrator to fix the problem with the Primary and Secondary servers. If, in the meantime, a second failure occurs, the Watchdog does not handle the failure. This limitation in the Watchdog is deliberate. There are situations where a series of failures and even seemingly appropriate responses can cause the error of having two Primary servers (either in PRIMARY ALONE or STANDALONE states). This is especially true if there are brief failures in the network, but no failures in the database servers themselves. An example that produces two Primary servers is provided in "Coding the Watchdog for multiple failures" on page 173.

During failure mode, the Watchdog polls both the Primary and Secondary servers. When it is able to connect to both servers, it sends the **hotstandby state** command to both servers to see whether it can communicate with them and to see which state each of them is in.

Once the Watchdog is able to communicate with both servers, it will decide what to do next based on the `solid.ini` parameter **DualSecAutoSwitch**. If **DualSecAutoSwitch** = `Yes` and both servers are secondary, then the Watchdog will automatically select one of the two secondaries to be a new primary and switch it to primary. If **DualSecAutoSwitch** = `No` then the system administrator must switch one server to be the primary. Note that **DualSecAutoSwitch** applies whether the Watchdog is in "normal" mode or "failure" mode.

Coding the Watchdog for multiple failures

There are two ways to handle multiple failures in the Watchdog. You can:

- After each failure (and automatic response by the Watchdog), require manual (human) intervention to check the situation. Manual intervention may require actions, such as restarting a server, or fixing a network problem. This is the approach that the Watchdog uses because it reduces the risk of having two Primary servers.
- Write a watchdog application that can handle multiple failures over time.

  This method does run the risk of having two Primary servers, as shown in the following example.

Dual primaries

In this example, Server1 is initially the Primary and Server2 is initially the Secondary.

1. A network failure occurs and Server1 becomes inaccessible.
2. The Watchdog switches Server2 from SECONDARY to PRIMARY ALONE.
3. A second network failure occurs, and Server2 becomes inaccessible.
4. The first network failure is repaired, and Server1 becomes accessible again.
5. The Watchdog, seeing that Server1 is accessible and Server2 is not, switches Server1 to PRIMARY ALONE.
6. The second network failure is fixed and Server2 becomes accessible again.
7. At this point, both Server1 and Server2 are in the PRIMARY ALONE state.

F.1.2 System design issues

How you configure HotStandby (locally or remotely, at one or more different locations, over the Internet, and with the Watchdog program) can affect the reliability and efficiency of your system. This section addresses these issues.

The illustration below shows one example of a heterogeneous system, in which the Primary and Secondary servers do not even use the same type of hardware and operating system.

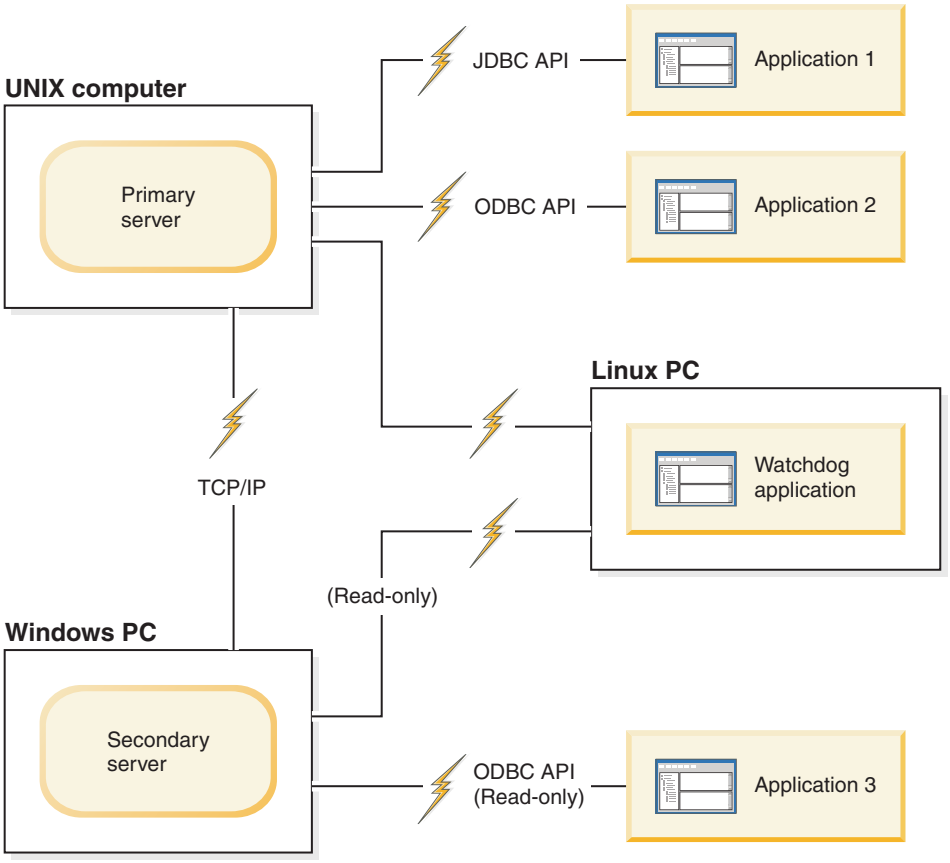


Figure 20. Heterogeneous HotStandby configuration with Watchdog

F.1.3 Watchdog configuration

For better efficiency and more precision in monitoring the state of the servers, the Watchdog is recommended as a separate component of any HotStandby configuration.

If only two machines are available, making it impossible to run the Watchdog program in a separate machine, run the Watchdog on the same machine where the Secondary server resides and set the parameter **AutoPrimaryAlone** to no in the configuration file (solid.ini) of both the Primary and Secondary server. Note that setting this parameter to no is extremely important, as it prevents the potential error of having two Primary servers.

CAUTION:
If both servers are in a state that allows writing (PRIMARY ALONE or STANDALONE), and if the databases of both servers are independently updated, then it will not be possible to resynchronize the two databases. Make sure that the Watchdog does not allow both servers to be put in the PRIMARY ALONE or STANDALONE state at the same time. See 3.6.2, "Network partitions and dual primaries," on page 66.

If the Primary server does fail, then the Watchdog is able to switch the Secondary to become the new Primary.

There are some disadvantages to putting the Watchdog in the same machine as the Secondary. The disadvantages include:

- If only the communication link between the Watchdog and the Primary is down, this configuration may result in a false switchover between the Primary and the Secondary.
- The communication link becomes a "single point of failure", that is, a single failure that may disable the entire system. (In most HotStandby configurations, the entire system is not disabled unless there are at least two failures.)
- If there is a network failure and the Secondary machine cannot communicate with the Primary machine, the users and applications are still able to access the Primary server and theoretically could continue operating with the Primary server. However, the Primary server stops accepting transactions because the watchdog cannot notify the Primary server to continue operating, for example by switching to PRIMARY ALONE state.

F.1.4 Using the sample Watchdog application

About this task

Initially, you should start the Watchdog after both servers are up and connected.

Procedure

To start the Watchdog, go to the current working directory of the Watchdog and at the prompt, issue the command:

```
watchdog
```

If you have not specified the usernames and passwords for connect1 and connect2 servers (capable of serving as Primary and Secondary) in the `solid.ini` file, the Watchdog prompts you for them.

Results

When started, the Watchdog pings both servers to check which one is Primary. The Watchdog remains in normal mode unless it detects a server failure after the number of retry attempts is exceeded. If a failure occurs after the Watchdog sends the last retry attempt to the server, the Watchdog switches to failure mode. When both the Primary and Secondary servers are up and reconnected, the Watchdog switches to normal mode.

F.2 Failure situations and Watchdog actions

This section describes how a typical watchdog program should work in specific failure scenarios that are commonly encountered.

The scenarios are in the context of either a server failure or a broken communication link between the Primary and Secondary server, or between one of the servers and the watchdog.

Although these commands may be issued by either a human administrator or a software program, for simplicity it is assumed that the commands are issued by the Watchdog sample.

F.2.1 Primary is down

Scenario

All connections to the Primary server are broken.

Remedy

When the Primary is down, switch the Secondary to be the new Primary and set the new Primary to the PRIMARY ALONE state. Later, the old Primary can become a new Secondary.

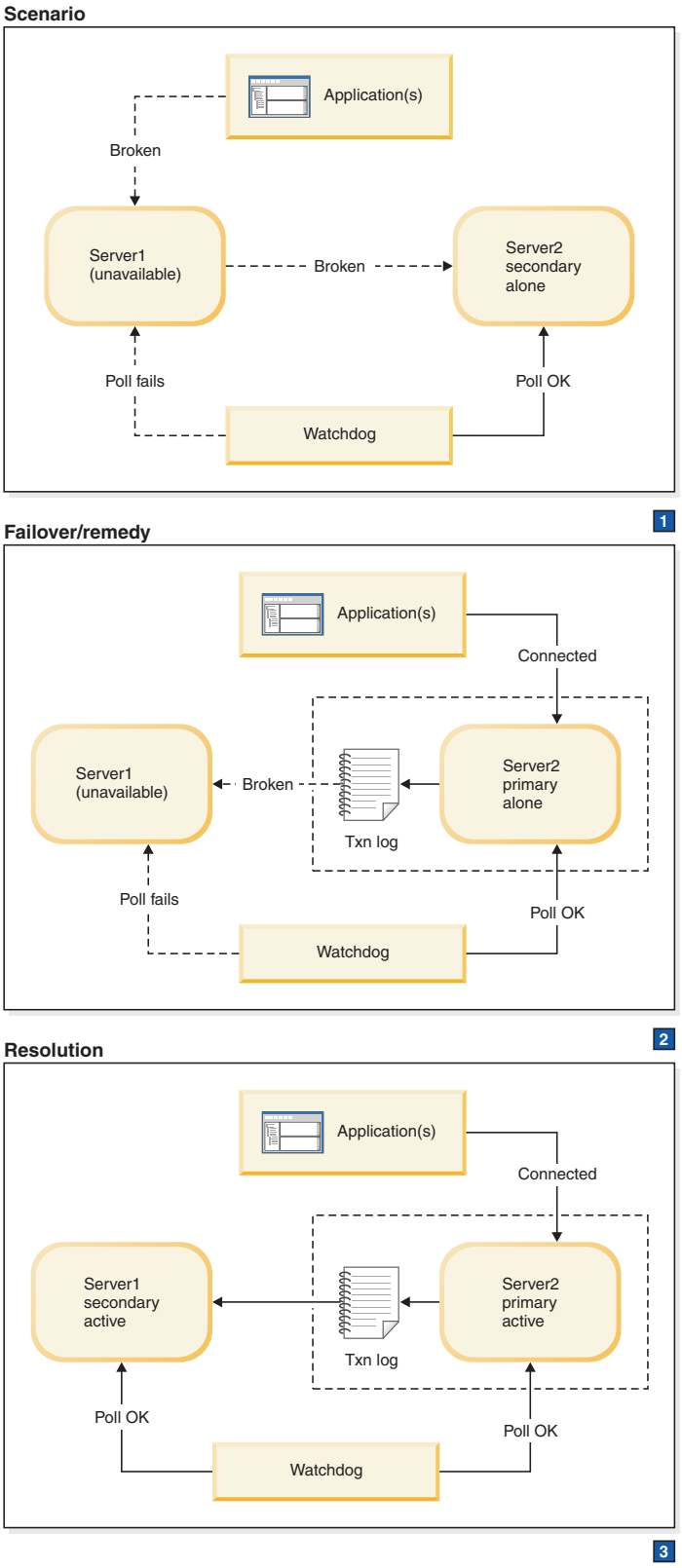


Figure 21. Primary is down scenario and remedy

1. Watchdog instructs Server2:
   HSB SET PRIMARY ALONE

Applications switch from Server1 to Server 2.

HSB SET STANDALONE

2. After Server1 is fixed, Server 1 is brought back up as Secondary.

   Watchdog instructs Server 2:

   - HSB NETCOPY
   - HSB CONNECT

3. If the transaction log fills up, you may have to switch the PRIMARY ALONE server to STANDALONE. In that case, you will also need to execute HSB NETCOPY before you reconnect the servers. If the transaction log does not fill up, then you must skip the NETCOPY command.

Symptoms

Applications cannot connect to the Primary. Also, the watchdog poll fails at the Primary. The HSB state of the secondary server is SECONDARY ALONE.

How to recover when the primary is down

About this task

To allow the "HotStandby" (Secondary server) to replace the Primary, do the following:

Procedure

1. Set the new Primary server to PRIMARY ALONE state by using the command:

   ```
   ADMIN COMMAND 'hotstandby set primary alone';
   ```

2. Reconnect applications to the new Primary.
3. Start using applications.
4. Fix and start the old Primary server as new Secondary server.
5. If necessary, copy the database from the new Primary to the new Secondary using command:

   ```
   ADMIN COMMAND 'hotstandby netcopy';
   ```

   Read 3.4.5, “Synchronizing primary and secondary servers,” on page 50 for details.

6. Reconnect the new Primary to the new Secondary using the command:

   ```
   ADMIN COMMAND 'hotstandby connect';
   ```

F.2.2 Secondary is down

Scenario

All connections to the Secondary server are broken. This may be caused either by a failure in the Secondary, or by a failure in the network that makes it impossible for either the Primary or the Watchdog to communicate with the Secondary server. In this section, the Secondary is referred to as failing, but in fact the problem may be with either the Secondary or the network.

Remedy

The standard remedy is to switch the Primary server to the PRIMARY ALONE state. After the Secondary is up again, synchronize it with the Primary.

Upon finding a problem with the connection to the Secondary server, the Primary server:

1. Suspends any open transactions, neither committing them nor rolling them back (the Primary does not send an error message — or a "success" message — to the client); and
2. Automatically switches its own state from PRIMARY ACTIVE to PRIMARY UNCERTAIN.

Typically, after making sure that the secondary server is unavailable, the watchdog will switch the Primary from PRIMARY UNCERTAIN to PRIMARY ALONE. After the Primary is switched to PRIMARY ALONE state, it can continue accepting transactions and saving them to send to the Secondary. Later, when the Secondary is brought back up, the Secondary can be sent the transaction log so that it can "catch up" to the Primary.

The Primary commits the open transactions after the Primary is set to PRIMARY ALONE state. To avoid the possibility that the Primary will commit the transactions when the Secondary has not, the transactions are kept in the transaction log, as though they had never been sent to the Secondary. When the Secondary is brought back up and starts catching up, the Primary sends that transaction log, and the Secondary checks each of the transactions. If any of the transactions are duplicates (that is, if the Secondary already committed that transaction before the Secondary failed), then the duplicate transactions are not re-executed on the Secondary.

The watchdog or system administrator must be careful in choosing whether to bring the Primary to PRIMARY ALONE state, or choose an alternative action. If the watchdog or system administrator chooses a different action than switching the Primary to PRIMARY ALONE state, she must take into account that the Secondary and Primary may not have the same data i.e. they may not both have rolled back the transaction. It is possible that the failed Secondary actually committed the data and crashed after committing the data but before sending the confirmation to the Primary, while the Primary never committed. In this situation, the secondary may actually be "ahead" of the Primary rather than behind it.

As always, the watchdog or administrator also must be careful not to allow both servers to go into PRIMARY ALONE state at the same time.

The diagram below is divided into three frames. The first frame shows the scenario, which is that the Primary and watchdog have lost contact with the Secondary. The next frame shows how to respond to keep your system working until the problem can be completely solved. The third frame shows the final state after the problem has been solved — that is, after the broken server has been fixed, or after communications have been restored.

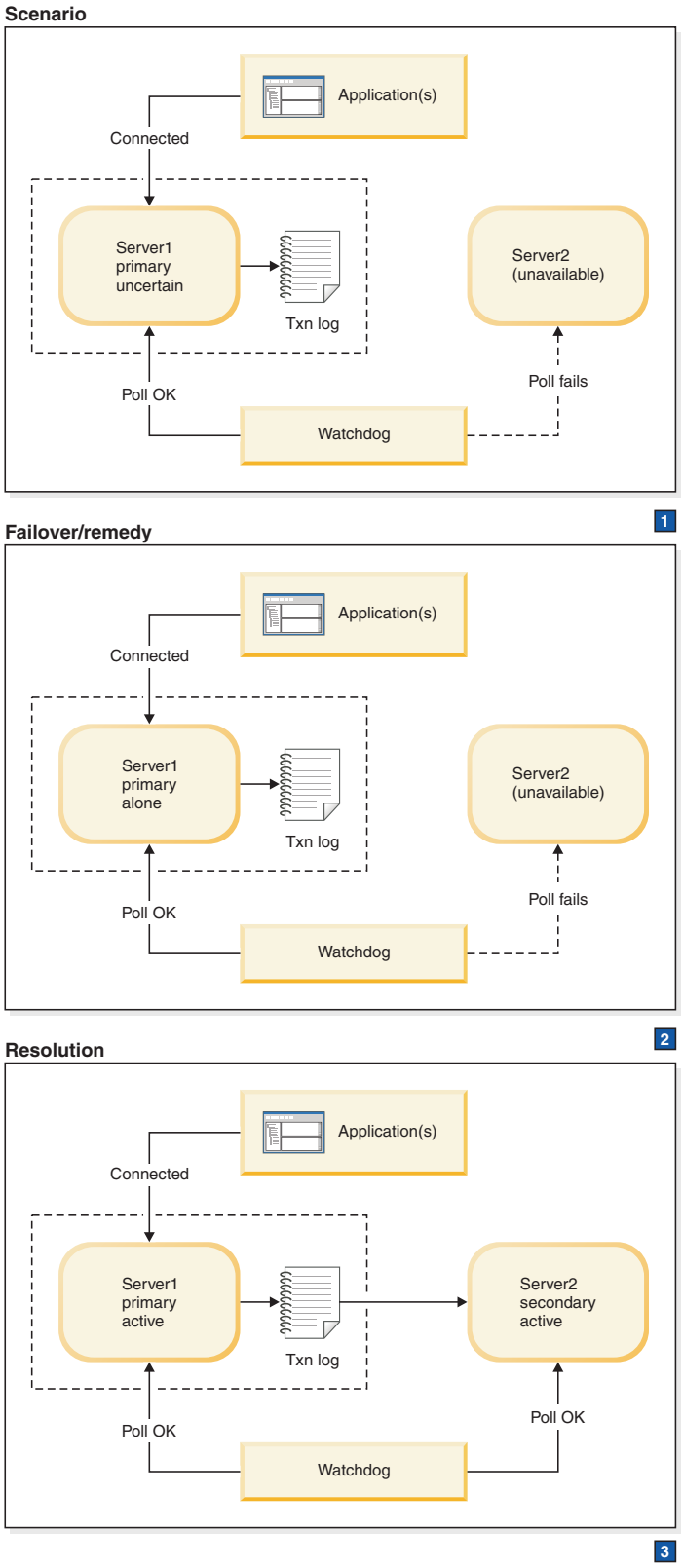


Figure 22. Secondary is down scenario and remedy

1. Watchdog instructs Server 1:
   HSB SET PRIMARY ALONE

HSB SET STANDALONE

2. After Server 2 is brought back up, Watchdog instructs Server 1:

   HSB NETCOPY

   HSB CONNECT

3. If the transaction log fills up, you may have to switch the PRIMARY ALONE server to STANDALONE. In that case, you will also need to execute HSB NETCOPY before you reconnect the servers. If the transaction log does not fill up, then you must skip the NETCOPY command.

Symptoms

The watchdog poll fails at the Secondary. The state of the primary server is either PRIMARY ALONE or PRIMARY UNCERTAIN.

How to recover when the secondary is down

About this task

To allow the Primary server to continue to receive transactions, operating independently of the Secondary server, do the following:

Procedure

1. If the Primary server is in the PRIMARY UNCERTAIN state, then set the Primary server to PRIMARY ALONE using the command:

   ```
   ADMIN COMMAND 'hotstandby set primary alone';
   ```

2. After the Secondary server has been repaired and restarted and/or the Secondary's network connections have been reestablished, check the state of the Primary server using the command:

   ```
   ADMIN COMMAND 'hotstandby state';
   ```

3. If the state of the Primary server is PRIMARY ALONE, then reconnect the Primary to the Secondary using the command:

   ```
   ADMIN COMMAND 'hotstandby connect';
   ```

4. If the state of the Primary server has earlier been changed to STANDALONE, then:

   a. Copy the database from the new Primary to the Secondary using the command:

      ```
      ADMIN COMMAND 'hotstandby netcopy';
      ```

   b. Read 3.4.5, “Synchronizing primary and secondary servers,” on page 50 for details.

5. Reconnect the Primary to the Secondary using the command:

   ```
   ADMIN COMMAND 'hotstandby connect';
   ```

Further scenarios when the secondary is down

If an application receives error message 10047 or 14537 from the Primary:

- Try to connect to the Secondary to check if its state was switched to new Primary.
- If its state is not one of the Primary states (PRIMARY ACTIVE or PRIMARY ALONE), see the scenario in F.2.1, “Primary is down,” on page 176.

F.2.3 Watchdog is down

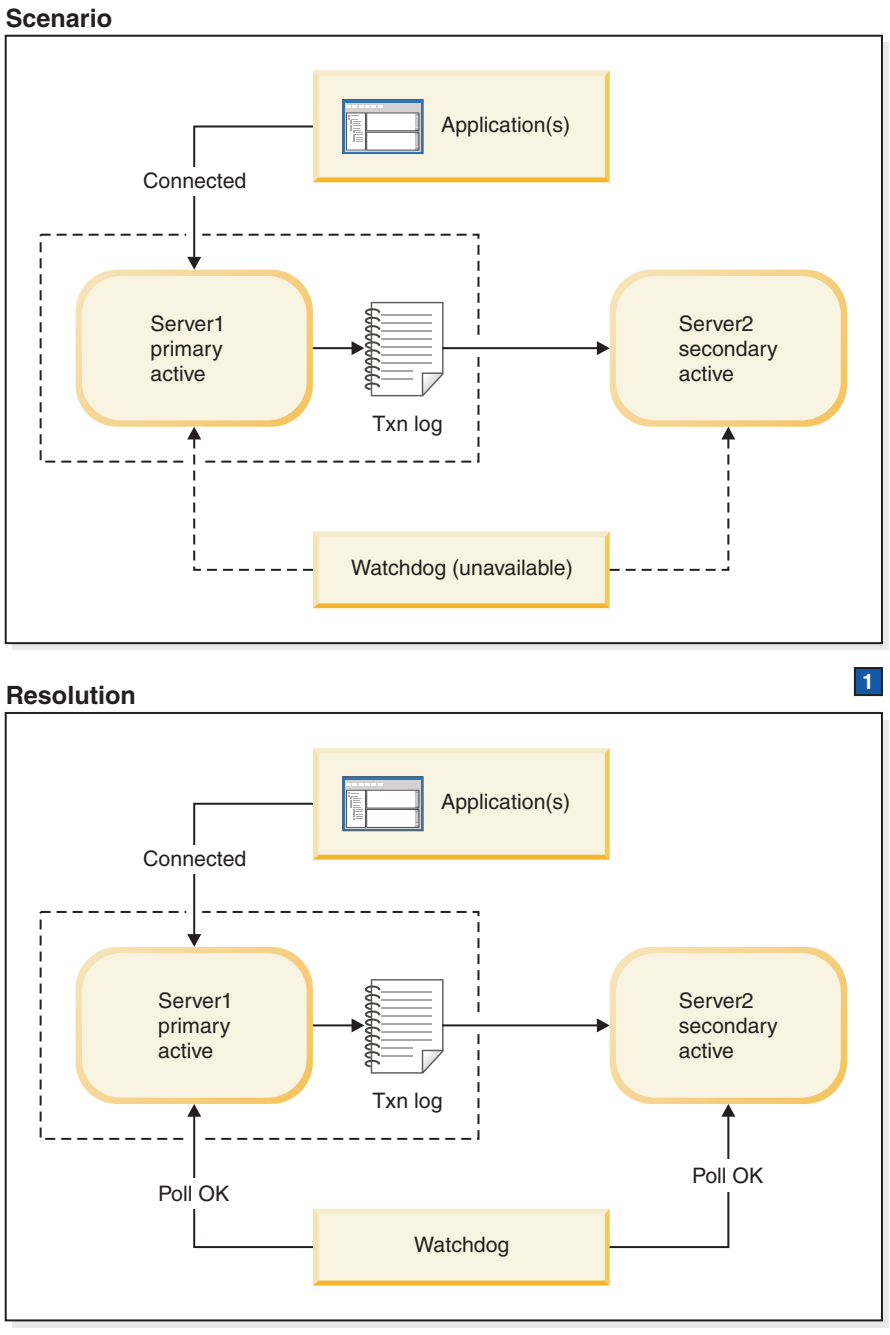
This section explains what happens if the Watchdog fails.

Scenario

All connections to the Watchdog are broken.

Remedy

Manual intervention is required. When the Watchdog is brought up, be sure to check the Primary and Secondary servers to confirm their states.



1. Bring the Watchdog back up or fix the network.

Figure 23. Watchdog is down scenario and remedy

Symptoms

The Watchdog process is down or network connections from the Watchdog to both servers are unavailable.

Further scenarios

If the servers have changed states and one server is no longer in service, refer to the applicable scenario in this section for instructions.

How to recover when the watchdog is down

About this task

To recover from the scenario where all connections to the Watchdog are broken:

Procedure

1. Allow the Primary and Secondary servers to continue normal operations.
2. Once the Watchdog is brought up, have it check the state of each server with the command:

   ```
   ADMIN COMMAND 'hotstandby state';
   ```

F.2.4 Communication link between Primary and Secondary is down

Scenario

The connection between the Primary and Secondary server is broken.

The Primary will switch itself to PRIMARY UNCERTAIN state. (If **AutoPrimaryAlone** is set to `Yes`, then the server will switch itself to PRIMARY ALONE state.)

Note: If the Primary server sends a commit message to the Secondary and then detects the failure of the Secondary, the Primary server relies on the Watchdog or the administrator to indicate how the Primary server is to proceed. This is because the Primary server is unable to detect whether the transaction was committed or rolled back in the Secondary before the Secondary server failed.

Until the Primary server receives a command from the Watchdog or the administrator, it no longer accepts transactions. At this stage, in order for the Primary server to continue operations, the Watchdog or administrator can set the Primary server to PRIMARY ALONE state.

Remedy

The Primary server can continue operations even when its link to the Secondary server is down. If the Primary is not already in PRIMARY ALONE state, then switch the Primary to the PRIMARY ALONE state. Once the link between the Primary and Secondary is restored, synchronize the databases.

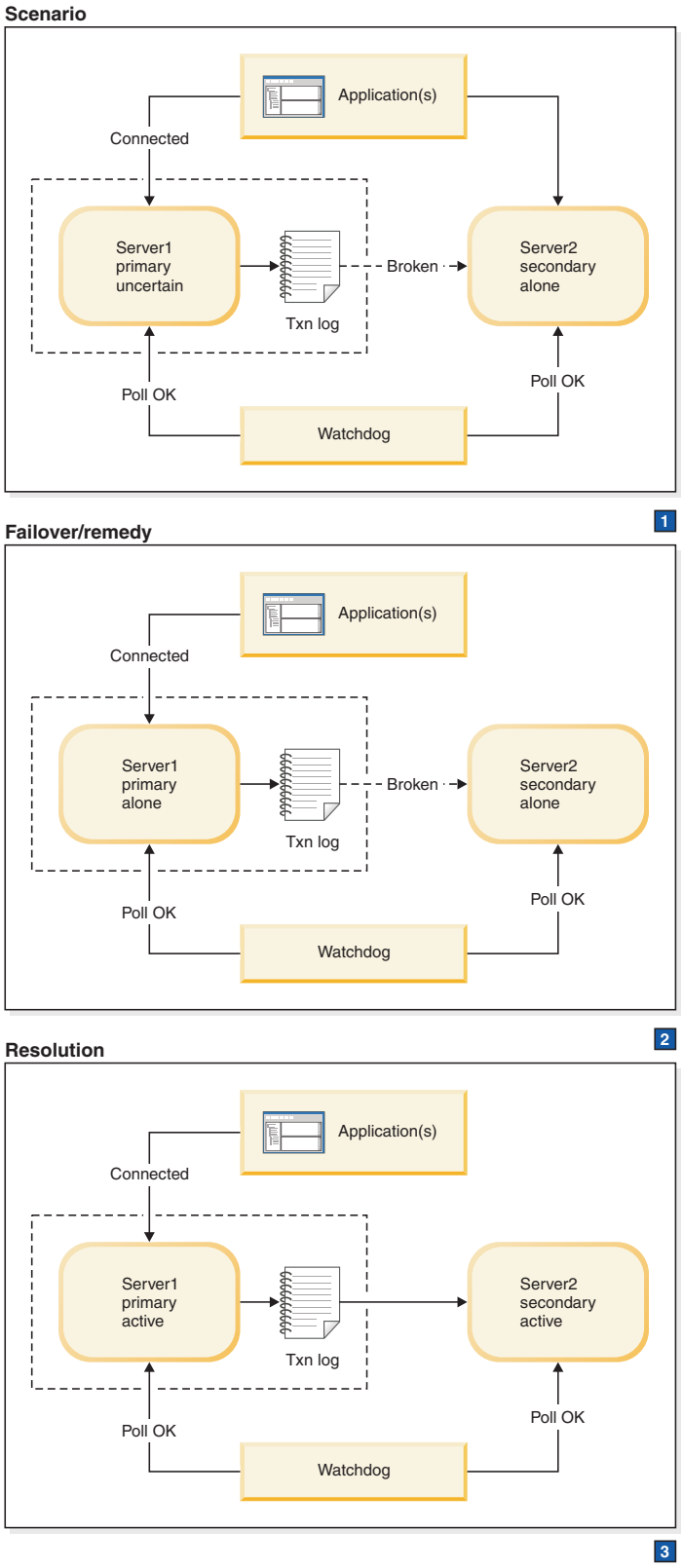


Figure 24. Broken link between Primary and Secondary scenario and remedy

1. Watchdog instructs Server 1:
   HSB SET PRIMARY ALONE

HSB SET STANDALONE

2. After the connection between primary and secondary is fixed, Watchdog instructs Server1:

   HSB NETCOPY

   HSB CONNECT

3. If the transaction log fills up, you may have to switch the PRIMARY ALONE server to STANDALONE. In that case, you will also need to execute HSB NETCOPY before you reconnect the servers. If the transaction log does not fill up, then you must skip the NETCOPY command.

Symptoms

The Primary server has no Secondary connected and the state is PRIMARY UNCERTAIN or PRIMARY ALONE.

How to recover when the communication link between the Primary and Secondary is down

About this task

To recover from the scenario where the connection between the Primary and Secondary server is broken:

Procedure

1. Fix the network connection between the Primary and Secondary servers.
2. Check the state of the Primary server using the command:

   ```
   ADMIN COMMAND 'hotstandby state';
   ```

3. If the state of the Primary server is PRIMARY ALONE, reconnect the Primary to the Secondary using the command:

   ```
   ADMIN COMMAND 'hotstandby connect';
   ```

4. If the state of the Primary server is STANDALONE, then:
   a. Copy the database from the Primary to the Secondary. Read 3.4.5, "Synchronizing primary and secondary servers," on page 50 for details.

      Before using the command **ADMIN COMMAND 'hotstandby netcopy';** be sure that the Secondary is up and running and is ready to receive the **netcopy**. Also, make sure that you set the Primary server's state to PRIMARY ALONE.

   b. Reconnect the Primary to the Secondary using the command:

      ```
      ADMIN COMMAND 'hotstandby connect';
      ```

Further scenarios when the communication link between the Primary and Secondary is down

If an application receives error message 10047 or 14537 from the Primary:

- Try to connect to the Secondary to check if it is switched as the new Primary.
- If the old Secondary is not switched as the new Primary, see scenario in F.2.1, "Primary is down," on page 176.

F.2.5 Communication link between the Watchdog and Primary is down

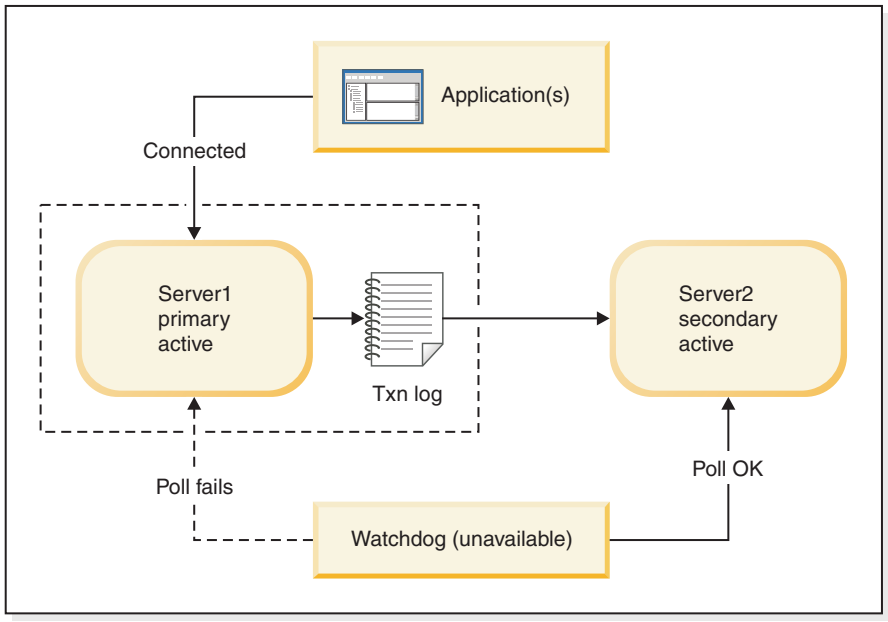
Scenario

The connection between the Watchdog and the Primary server is broken.

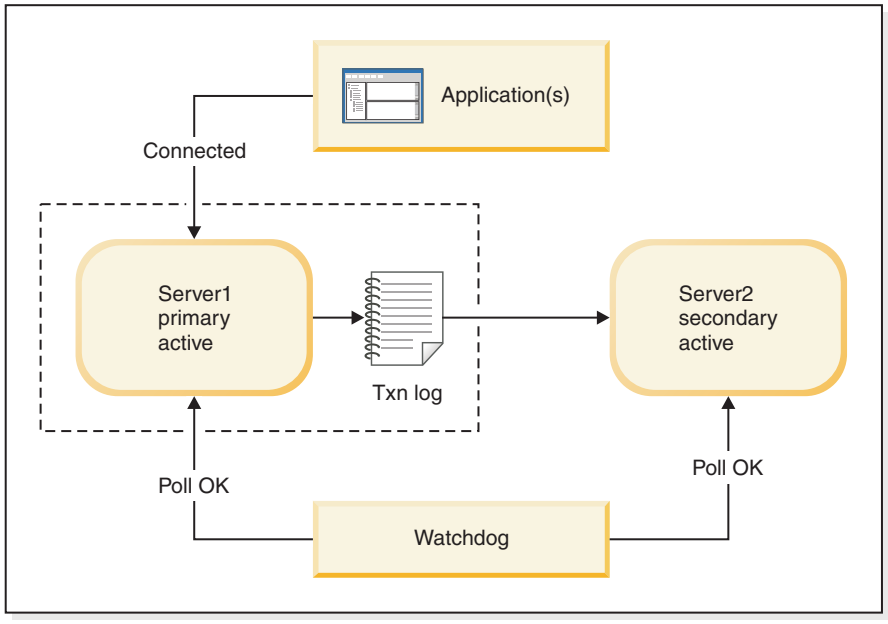
Remedy

The Primary and Secondary servers can continue operations even when the Watchdog link to the Primary server is down. When the Watchdog link to the Primary is fixed, be sure to check the states of the Primary and Secondary servers.

Scenario



Resolution 1



1. Fix Watchdog's network connection to Server1.

Figure 25. Broken link between Watchdog and Primary scenario and remedy

Symptoms

The Watchdog poll fails at the Primary server. However, the secondary server state is reported to be SECONDARY ACTIVE. This means that the primary server is very probably okay and that the Watchdog has merely lost contact with the Primary.

Further scenarios

If the states of the servers have changed and one server is no longer in service, refer to the applicable scenario in this section for instructions.

How to recover when the communication link between the Watchdog and Primary is down

About this task

To recover from the scenario where connection between the Watchdog and the Primary server is broken:

Procedure

1. Allow the Primary and Secondary servers to continue normal operations.
2. Fix the network connection between the Watchdog and the Primary server.
3. Once the network is connected, have the Watchdog check the states of each server with the command:

   ```
   ADMIN COMMAND 'hotstandby state';
   ```

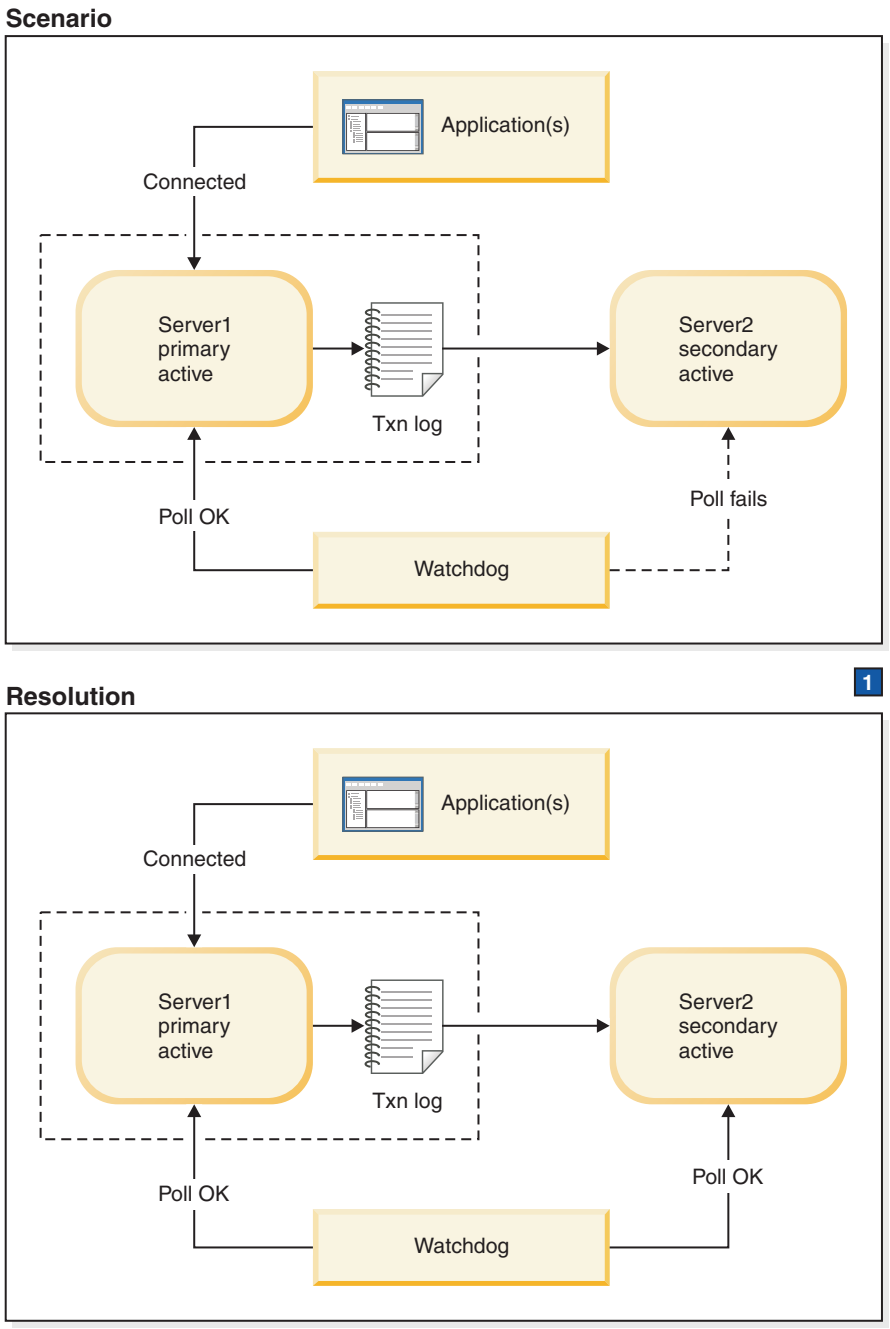
F.2.6 Communication link between the Watchdog and Secondary is down

Scenario

The connection between the Watchdog and the Secondary server is broken.

Remedy

The Primary and Secondary servers can continue operations even when the Watchdog link to the Secondary server is down. When the Watchdog link to the Secondary is fixed, be sure to check the Primary and Secondary servers to confirm their states.



1. Fix Watchdog's network connection to Server2.

Figure 26. Broken link between Watchdog and Secondary scenario and remedy

Symptoms

The Watchdog poll fails at the Secondary server.

Further scenarios

If the servers states have changed and one server is no longer in service, refer to the applicable scenario in this section for instructions.

How to recover when the communication link between the Watchdog and Secondary is down

About this task

To recover from the scenario where the connection between the Watchdog and the Secondary server is broken:

Procedure

1. Allow the Primary and Secondary servers to continue normal operations.
2. Fix the network connection between the Watchdog and the Secondary server.
3. Once the network is connected, have the Watchdog check the state of each server with the command:

   ```
   ADMIN COMMAND 'hotstandby state';
   ```

F.2.7 Communication links between the Watchdog and Primary, and between the Primary and Secondary, are down

Scenario

The connections between the Watchdog and the Primary server, and between the Primary server and Secondary server, are broken.

Remedy

For the Watchdog to continue monitoring the Primary server, switch the Secondary server to be the new Primary and set this new Primary to the PRIMARY ALONE state. Later, set up a new Secondary server and synchronize it with the Primary.

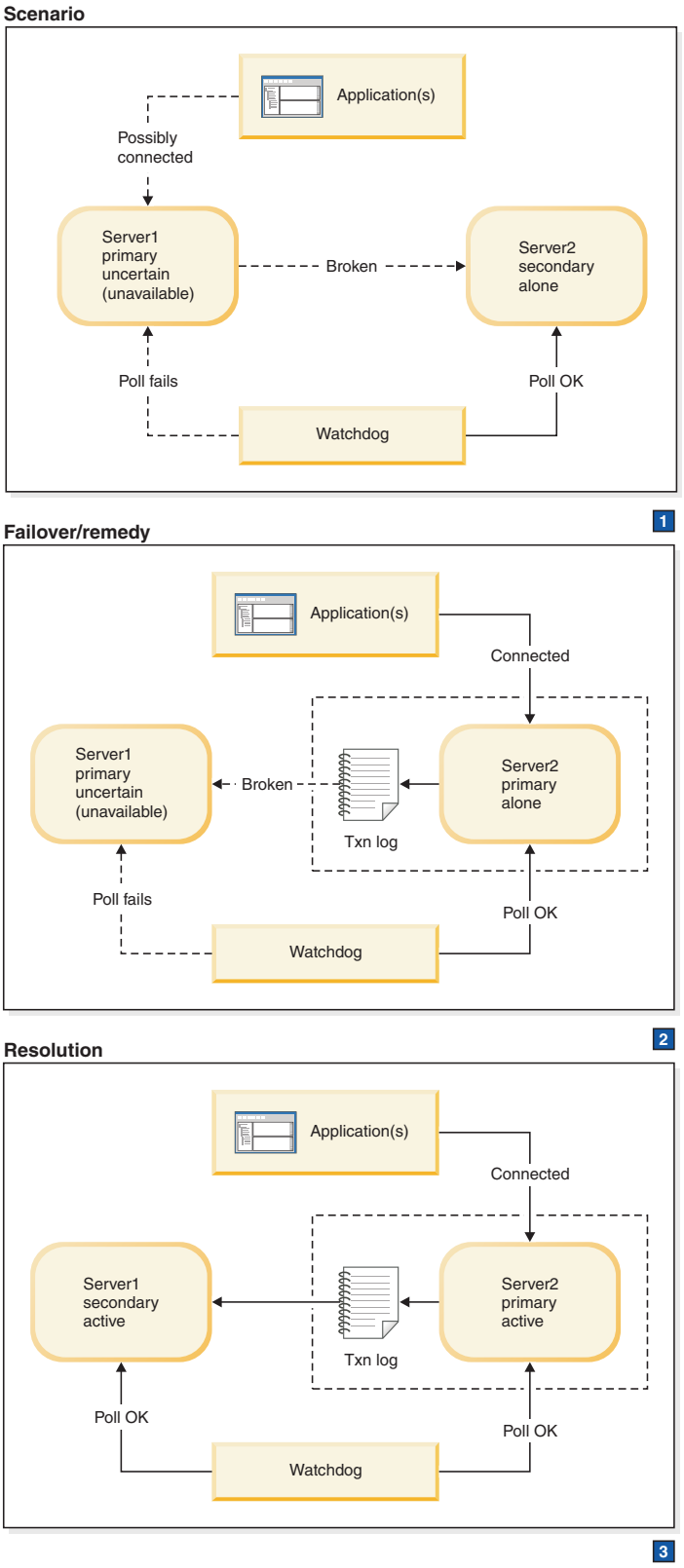


Figure 27. Broken link between Watchdog and Primary, and between Primary and Secondary, scenario and remedy

1. Server1's role is Primary Uncertain. However, from the watchdog's point of view, Server1 is unavailable, not Primary Uncertain.

   Watchdog instructs Server2:

HSB SET PRIMARY ALONE

Applications switch from Server1 to Server 2.

2. Both servers believe they are primary. If a program or an administrator (manual intervention) switches Server1 from Primary Uncertain to Primary Alone, then there are two active primaries, both of which could be updating data, and the differences would not be resolvable.

   After network connections are fixed, Watchdog instructs Server 1:

   HSB SWITCH SECONDARY

   Watchdog instructs Server2:

   HSB NETCOPY

   HSB CONNECT

3. If the transaction log fills up, you may have to switch the PRIMARY ALONE server to STANDALONE. In that case, you will also need to execute HSB NETCOPY before you reconnect the servers. If the transaction log does not fill up, then you must skip the NETCOPY command.

Symptoms

The Watchdog poll fails at the Primary server. The Secondary server and Primary server have lost their connections to each other; therefore Server2 is in the state SECONDARY ALONE, and the Primary (if it can be contacted) will report that its state is PRIMARY UNCERTAIN or PRIMARY ALONE.

The beginning of this scenario assumes that applications are possibly connected to the old Primary. However, since the old Primary is in the PRIMARY UNCERTAIN state, the applications are unable to perform updates. Note that it is also possible that the applications connected to Server1 may have lost their communication link and no longer know that the old Primary exists.

How to recover when communication links between the Watchdog and Primary, and between the Primary and Secondary, are down

To recover from the scenario where the connections between the Watchdog and the Primary server, and between the Primary server and Secondary server, are broken, perform the steps necessary to make the hot standby server (the Secondary server) replace the Primary.

About this task

To allow the Secondary server to replace the Primary, do the following:

Procedure

1. If the old Primary is in the PRIMARY UNCERTAIN state or is cut off from the applications as well as the Secondary, then set the Secondary server to PRIMARY ALONE state using the command:

   ```
   ADMIN COMMAND 'hotstandby set primary alone';
   ```

2. Reconnect applications to the new Primary.
3. Fix the network or the broken connections to the old Primary.
4. Check the server states. Both servers must now be running.
5. If the new Primary is in STANDALONE state (for example, because the new Primary's transaction log filled up while the connections were being fixed):
   a. Set the new primary to PRIMARY ALONE state using the command:

```
ADMIN COMMAND 'hotstandby set primary alone';
```

b. Copy the database from the new Primary to the new Secondary. Read 3.4.5, “Synchronizing primary and secondary servers,” on page 50 for details.

6. If the new Primary is in PRIMARY ALONE state:
   a. Switch the old Primary to be the new Secondary server using the command:
   ```
   ADMIN COMMAND 'hotstandby switch secondary';
   ```
7. Reconnect the new Primary to the new Secondary using the command:
   ```
   ADMIN COMMAND 'hotstandby connect';
   ```

Further scenarios where communication links between the Watchdog and Primary, and between the Primary and Secondary, are down

If an application receives error message 10047 or 14537 from the new Primary:

- Try to connect to the old Secondary to check if it has switched to be the new Primary.
- If the old Secondary is not switched to be the new Primary, re-execute the transaction with the original Primary in PRIMARY ALONE state.

F.2.8 Communication links between the Watchdog and Secondary, and between the Primary and Secondary, are down

Scenario

The connection between the Watchdog and the Secondary server, and the connection between the Primary server and Secondary, server are broken.

Remedy

The Primary server can continue operations even when its links to the Secondary server and the Watchdog are down. Switch the Primary server to the PRIMARY ALONE state, if it is not already in PRIMARY ALONE state. Later, when the Secondary is up again, synchronize it with the Primary.

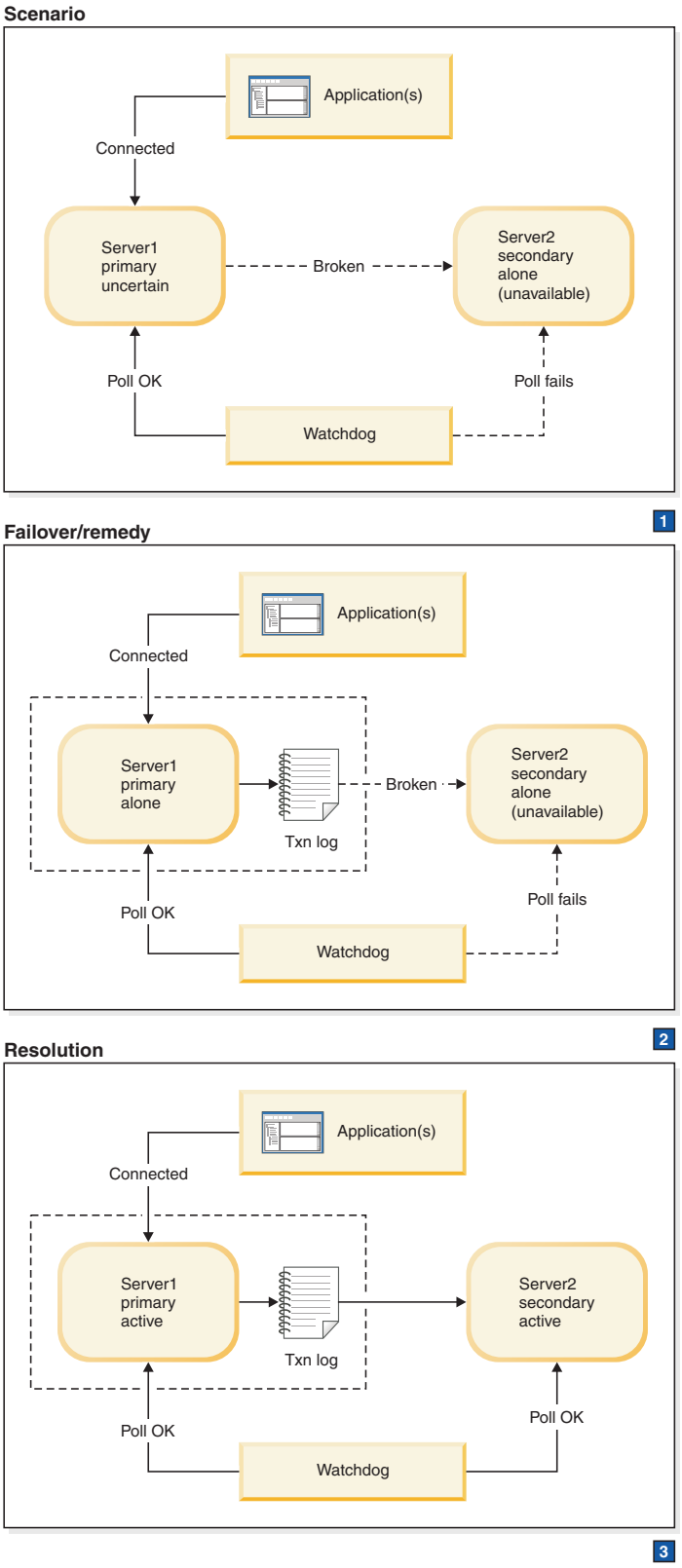


Figure 28. Broken link between Watchdog and Secondary and between Primary and Secondary scenario and remedy

1. Server2 sees its role as Secondary Alone, but the Watchdog cannot see Server2 and therefore believes Server2 is unavailable.

   Watchdog instructs Server 1:

HSB SET PRIMARY ALONE

2. After the connections to secondary are fixed, Watchdog instructs Server 1:

   HSB NETCOPY

   HSB CONNECT

3. If the transaction log fills up, you may have to switch the PRIMARY ALONE server to STANDALONE. In that case, you will also need to execute HSB NETCOPY before you reconnect the servers. If the transaction log does not fill up, then you must skip the NETCOPY command.

Symptoms

The Watchdog poll fails at the Secondary; the Primary server has no Secondary connected and switches to state PRIMARY UNCERTAIN or PRIMARY ALONE.

How to recover when communication links between the Watchdog and Secondary, and between the Primary and Secondary, are down

About this task

To recover from the scenario where the connection between the Watchdog and the Secondary server, and the connection between the Primary server and Secondary server are broken:

Procedure

1. Try to fix the connections.
2. After the connections are fixed, check the state of the Primary server using the command **ADMIN COMMAND 'hotstandby state'**.
3. If the state of the Primary is STANDALONE:
   a. Ensure that both servers are running.
   b. Set the state of the Primary server to PRIMARY ALONE using command:
      ```
      ADMIN COMMAND 'hotstandby set primary alone';
      ```
   c. Copy the database from the Primary to the secondary using command:
      ```
      ADMIN COMMAND 'hotstandby netcopy';
      ```

      Read 3.4.5, “Synchronizing primary and secondary servers,” on page 50 for details.
4. Reconnect the Primary to the Secondary using the command:
   ```
   ADMIN COMMAND 'hotstandby connect';
   ```

Further scenarios where the communication links between the Watchdog and Secondary, and between the Primary and Secondary are down

If an application receives error message 10047 or 14537 from the Primary:

- Try to connect to the Secondary to check if it switched to be Primary.
- If the Secondary is not switched as the new Primary, re-execute the transaction with the original Primary in PRIMARY ALONE state.

F.3 Watchdog section of the `solid.ini` configuration file

The `solid.ini` file for the Watchdog contains a `[Watchdog]` configuration section to specify Watchdog-specific parameters.

Important: The parameters in the [Watchdog] section of the solid.ini file are NOT all predefined by solidDB. Depending upon how you write your Watchdog and whether you want it to read parameter information from the solid.ini file, you can use any mix of the parameters defined here and parameters that you have defined. You can also ignore parameters. The parameters shown here are for the sample C-language Watchdog program that solidDB provides.

Table 42. Watchdog parameters

[Watchdog]	Description	Factory value

AutoSwitch	If the **AutoSwitch** parameter is set to yes, the Watchdog automatically does the following:
	1. If the Secondary server fails, then the Watchdog tells the Primary server to switch to PRIMARY ALONE state (rather than stay in PRIMARY UNCERTAIN) state.
	2. If the Primary server fails, then the Watchdog automatically sends the commands:
	`'hsb switch primary'`
	`'hsb set primary alone'`
	to switch the original Secondary to be the new Primary.
	For example:
	`[Watchdog]`
	`AutoSwitch = NO`
	This parameter is optional.
		Yes

Connect1	The **Connect1** parameter in the [Watchdog] section enables the Watchdog application to connect to the Primary or Secondary server. This is a required parameter that defines the protocol and network address for the **Connect1** server. For example: `connect1 = tcp primarymachine 1313`	None
Connect2	The **Connect2** parameter in the [Watchdog] section enables the Watchdog application to connect to the Primary or Secondary server. This is a required parameter that defines the protocol and network address for the **Connect2** server. For example: `connect2 = tcp secondarymachine 1313`	None
DualSecAutoSwitch	If **DualSecAutoSwitch** = Yes and both servers are secondary, then the Watchdog will automatically select one of the two secondaries to be a new primary and switch it to primary. If **DualSecAutoSwitch** = No then the system administrator must switch one server to be the primary. Note that **DualSecAutoSwitch** applies whether the Watchdog is in "normal" mode or "failure" mode.	Yes

Table 42. Watchdog parameters (continued)

[Watchdog]	Description	Factory value

NumRetry	The **NumRetry** parameter in the [Watchdog] section lets you specify the number of Watchdog attempts to connect to a Secondary or Primary server before the connection attempt is considered a response failure or error.
	
	For example:
	`[Watchdog]`
	`NumRetry = 3`
	
	The retries are in addition to the original try. If number of retries is set to 3, then the total number of attempts is 4. Note that the retries are immediate. The Watchdog does not wait for an interval of time (such as **PingTimeout**) in between retries when there is a failure.
	
	This parameter is optional.
		0

Password1 Password2	See the description of the **Username1** and **Username2** parameters below.	No factory value.
Pessimistic	Setting this parameter to Yes can speed up Watchdog reactions. When **Pessimistic** = No, the Watchdog checks its connections with the servers, but does not actually act (for example, change the state of a server to PRIMARY ALONE) until after one of the servers detects that there is a problem and changes its state (for example, to PRIMARY UNCERTAIN). When **Pessimistic** = Yes, the Watchdog acts as soon as the Watchdog itself loses contact with one of the servers; the Watchdog does not wait for the remaining server to change states. This can speed up the reaction time, but may also increase the odds of false alarms, for example due to network problems. When **Pessimistic** = Yes, the Watchdog reacts as follows: If the Watchdog has lost contact with the Primary, then the Watchdog switches the Secondary to be the Primary; if the Watchdog loses contact with the Secondary, then the Watchdog sets the Primary to PRIMARY ALONE. **CAUTION:** **Setting Pessimistic = Yes may cause extra switching or even dual primaries. This parameter should not be set to Yes unless the network is much less likely to fail than the server.** You can also turn on Pessimistic behavior by using the optional command-line switch "**-p**".	No

Table 42. Watchdog parameters (continued)

[Watchdog]	Description	Factory value

PingInterval	The **PingInterval** parameter in the [Watchdog] section lets you specify the interval in milliseconds between querying status connect information in normal Watchdog mode. To detect server failure, the Watchdog sends the hotstandby status connect command to both Primary and Secondary servers after every **PingInterval** milliseconds.
	
	For example:
	`[Watchdog]`
	`PingInterval = 5000`
	
	This parameter is optional.
	
	Note that the **PingInterval** parameter for the Watchdog is different from the **PingTimeout** parameter for the servers.
	
	CAUTION:
	Previous sample Watchdogs required that the PingInterval be specified in seconds, not milliseconds. If you are using an older solid.ini file, you should update it.
		1000
		
		(1 second)

Username1 Username2	The **Username** and **Password** parameters in the [Watchdog] section are optional. They specify the username and password that are authorized for using the connect1 server.
	
	For example:
	`[Watchdog]`
	`Username1 = Tom`
	`Password1 = dr17xy`
	`Username2 = Jerry`
	`Password2 = M89tvt`
	
	If (for security reasons) these parameters are not specified in the solid.ini configuration file, the Watchdog will prompt for them when the Watchdog is started.
		No factory value.

WatchdogLog	The **WatchdogLog** parameter in the [Watchdog] section lets you specify the file name of the Watchdog log. The Watchdog log is created in the current working directory. It is used to record Watchdog messages, alerting administrators of the need to issue Watchdog commands.
	
	For example:
	`[Watchdog]`
	`WatchdogLog = Watchdog.log`
	
	Note that quotation marks around the file name are not required (as long as it does not contain special characters such as the blank or certain punctuation marks).
	
	This parameter is optional.
		Watchdog.log

When using the parameter

```
[Logging]
DurabilityLevel
```

the **DurabilityLevel** parameter value affects only the Primary server. The logging mode of the Secondary server is dictated by the **2SafeAckPolicy** parameter in the [HotStandby] section.

Index

Special characters

Numerics

A

B

C

D

O

P

R

S

T

U

V

W

Notices

This product is protected by U.S. patents 6144941, 7136912, 6970876, 7139775, 6978396, 7266702, 7406489, 7502796, and 7587429.

This product is assigned the U.S. Export Control Classification Number ECCN=5D992b.

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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

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.

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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

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 is for planning purposes only. The information herein is subject to change before the products described become available.

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.

COPYRIGHT LICENSE:

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.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, ibm.com®, Solid, solidDB, InfoSphere®, DB2®, Informix®, and WebSphere® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

Microsoft and Windows are trademarks 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.

Other product and service names might be trademarks of IBM or other companies.

SC27-3843-05