SYBASE®
An SAP® Company

In-Memory Database Users Guide

Adaptive Server® Enterprise

15.7

DOCUMENT ID: DC01186-01-1570-01

LAST REVISED: September 2011

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

CHAPTER 1

In-Memory Databases

In-memory databases run entirely in a named cache (that is, in the Adaptive Server® memory space), without using disk storage for data or logs. Because an in-memory database does not require I/O, its performance can be much better than a traditional, disk-resident database. In-memory databases are not designed for recovery: their transaction logs are written to the cache and not to disk, and any data changes are lost if the database fails. In-memory databases perform transactional logging for runtime rollback, and for other operations such as firing triggers, deferred mode updates, replication, and so on.

Disk-resident databases write to disk, and ensure that the transactional properties of atomicity, consistency, integrity, and durability (known as the ACID properties) are maintained. Durability refers to the persistence of transactions after they have committed. A traditional Adaptive Server database, also known as disk-resident, operates at full durability by writing its transaction log to disk when a transaction commits. This, along with data pages being written periodically to disk, ensures that all committed transactions are durable.

In-memory databases do not write data or log to disk, and trade the guarantee of transaction durability for performance improvements. In the event of a database failure, in-memory databases cannot be recovered. If your applications require data recoverability following a server failure or a normal shutdown, consider using a traditional Adaptive Server database.

With support for relaxed durability, Sybase® extends the performance benefits of an in-memory database to disk-resident databases. A disk-resident database operates at full durability to guarantee transactional recovery from a server failure. Relaxed-durability databases trade the full durability of committed transactions for enhanced runtime performance for transactional workloads.

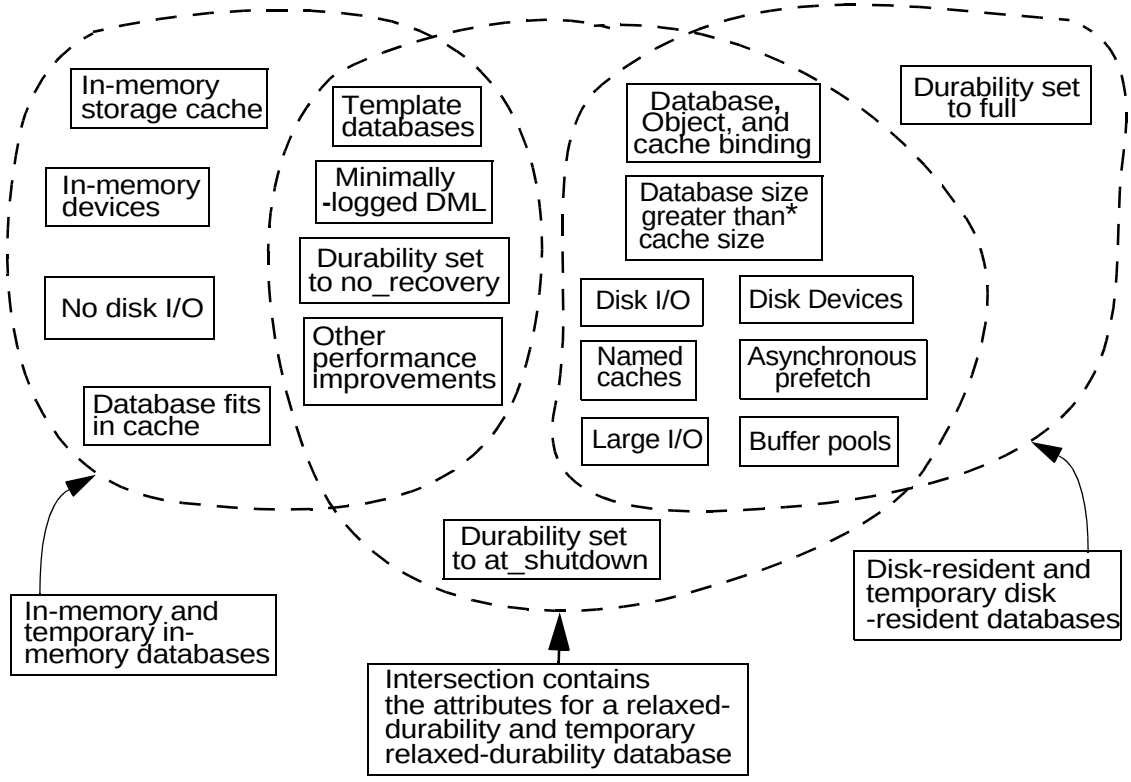
The performance benefits of in-memory and relaxed-durability databases include:

- An in-memory database does not wait for I/O.
- Improved buffer and user log cache management, so you need incure the overhead of user log cache flushes and buffer management when Adaptive Server performs concurrent updates to the same data.
- Runtime strategies that may avoid flushing in the user-log cache to the transaction log when the transaction commits or aborts. These strategies reduce the contention on in-memory log pages.
- Support for minimally-logged DML operations that use in-memory logging techniques improves the performance of large-volume DML operations.

Adaptive Server version 15.5 and later allows you to create these types of databases (illustrated in Figure 1-1, below):

- Disk-resident databases with durability set to full (this is the default, or traditional, Adaptive Server database)
- User-created disk-resident temporary databases
- In-memory user databases with durability set to no_recovery
- User-created in-memory temporary databases with durability set to no_recovery
- Disk-resident relaxed-durability databases with durability set to no_recovery or at_shutdown

Figure 1-1: Attributes of in-memory, relaxed-durability and disk-resident databases



*Greater than or equal to the cache size

Note See your Replication Server® documention for information about using in-memory and relaxed-durability databases and DML logging in a replicated environment.

Cache and buffer support

Caches that host in-memory databases must be large enough to contain the entire database, and every page in the database must reside in the cache, without any buffer replacement or I/O to disk. You cannot use caches that have been created:

- To host in-memory databases to bind other databases or other objects.
- For binding other databases or objects to host in-memory databases. Named caches used to host an in-memory database use a different structure, and are dedicated to in-memory databases.

Use sp_cacheconfig to create the cache for the in-memory database. Use disk init to divide the cache into in-memory devices, which are similar to disk devices, and support segments. You can bind one or more logical segments to in-memory devices, allowing you to bind objects to individual segments.

Consider the following before you bind any objects to a cache for in-memory or relaxed durability databases:

- Use named caches to bind entire relaxed-durability or full durability database. You may bind
  - Individual objects in an relaxed-durability database to a named cache, similar to binding individual objects in a regular database
  - Relaxed-durability database to a named cache (for example, the default data cache)
  - In-memory storage caches are similar to named caches, but are configured for efficient in-memory access. Individual objects in the relaxed-durability database to different caches
- Caching behavior for in-memory storage caches is similar to the caching behavior for regular caches.
- Use the same monitoring tools and tuning techniques you use to improve the performance of named caches to improve the performance of relaxed-durability databases bound to named caches.
- Because a single in-memory storage cache hosts the entire database, you do not bind a database or object to individual caches. Most existing cache manager monitoring and tuning apply to in-memory caches. See Chapter 2, “Monitoring Performance with sp_sysmon,” in *Performance and Tuning Series: Monitoring Adaptive Server with sp_sysmon*.

In-memory databases must be hosted by a single cache, but can reside on multiple in-memory devices created from that cache. Figure 1-2 shows imdb_cache cache, which contains a single in-memory storage device, imdb_dev:

Figure 1-2: Single cache hosting a single device

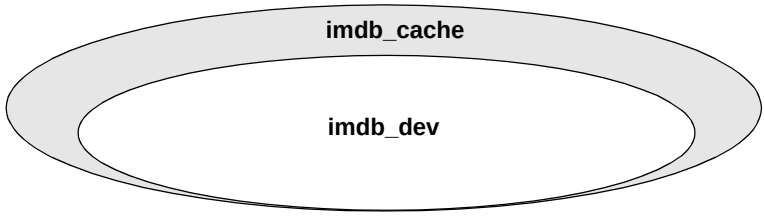
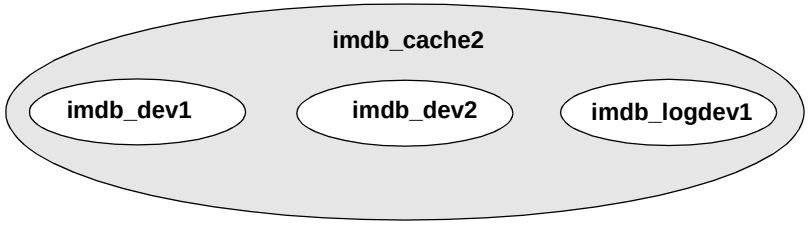


Figure 1-3 shows imdb_cache2 cache, which includes two in-memory data devices, imdb_dev1 and imdb_dev2, and a log device, imdb_logdev1:

Figure 1-3: Single cache hosting multiple devices



Use create inmemory database to create the in-memory database directly on the logical devices. If your installation does not use segments to limit space for objects, create the database on the in-memory device to be used for the entire in-memory storage cache. For finer granularity in space management for individual objects and threshold procedure support, create the database and log on separate in-memory devices.

Use ddlgen to generate database and object definitions for in-memory and relaxed-durability databases. See the *Utility Guide*.

Use alter database to change the layout of existing in-memory databases. See the *Reference Manual: Commands*.

Durability levels

Data that you change in a committed, durable transaction survives after you restart the server following a system failure or shutdown with nowait. The durability of a transaction in a traditional disk-resident database results from flushing the transaction log and the database pages to disk. In-memory databases provide no transactional durability after a server fails or impolite shutdown.

A relaxed-durability database offers two levels of durability from which you can select. The first level of durability is similar to an in-memory database: if the server fails, data is lost. The second level of durability is between that of a disk-resident database and that of an in-memory database: all transactions are completed and are persisted to disk only with a polite shutdown. This enables relaxed-durability databases to take advantage of many performance optimizations of in-memory databases.

Databases with a durability set to no_recovery or at_shutdown—whether they are in-memory or disk-resident—are referred to as low-durability databases. Data in low durability databases survives after a commit (provided you do not restart the server).

Use create database with durability=*durability_level* to set a database's durability level. Adaptive Server supports full, no_recovery, and at_shutdown durability levels. See the *Reference Manual: Commands*.

Table 1-1 describes the operations you can perform at each durability level:

Table 1-1: Durability levels for databases

Operation	no_recovery	at_shutdown	full
Create disk-resident databases	Yes	Yes	Yes
Create in-memory databases	Yes	No	No
Runtime rollback	Yes	Yes	Yes
Failure recovery	No	No	Yes
Database restored after a polite shutdown and restart.	No	Yes	Yes
Dump database to and load from archive.	Yes	Yes	Yes
Dump transaction log from device to archive. Load transaction log from archive to device.	No	No	Yes

Reduced durability and improved performance applies only to relaxed-durability databases using at_shutdown and no_recovery. You can bind relaxed-durability databases to named caches, for which the cache size can be smaller than the database size.

Temporary databases and in-memory temporary databases

Temporary databases implicitly use a durability of no_recovery. You may create temporary databases with an explicit no_recovery durability to enhance performance, and you may alter existing temporary databases to explicitly set their durability to no_recovery after upgrade.

Temporary databases that you use entirely in-memory explicitly use a durability of no_recovery.

Adaptive Server allows you to create and manage user-created tempdb groups in addition to managing the default tempdb group. User-created tempdb groups may include other user-created temporary databases, and support application and login binding.

You cannot remove the system tempdb from the default temporary database group. You cannot add system tempdb to any other user-created tempdb group.

Temporary database groups can mix disk-resident or in-memory temporary databases.

You can designate and administer user-created tempdb groups to contain only disk-resident or only in-memory temporary databases. Although Adaptive Server does not explicitly impose any restrictions, by controlling the membership to the tempdb groups, you can assign specific disk-only or in-memory-only tempdb groups to specific logins or applications.

Multidatabase transactions and database types

A database's durability affects transactions that span multiple databases.

Low-durability databases—user-created temporary databases, in-memory databases, or disk-resident relaxed-durability databases—can participate in a multi-database transaction, even if the coordinating database uses full durability. However, if the coordinating database uses low durability, transactions that span other full-durability databases are not allowed.

Table 1-2: Database participants in a multidatabase transaction

Coordinating database	Participant database	Multidatabase transaction
Full durability database	Temporary, in-memory, relaxed-durability, or temporary in-memory database	Allowed
Temporary, in-memory, relaxed-durability, or temporary in-memory database	Temporary, in-memory, relaxed-durability, or temporary in-memory database	Allowed
Temporary, in-memory, relaxed-durability, or temporary in-memory database	Full-durability database	Not allowed

Template databases

You can use a database other than model as the template for an in-memory database to load reference data (for example, tables and stored procedures) into the database created with the template. The template database must be an existing disk-resident user database with full durability. Using templates to create a database and minimally logged DML are supported only on low durability databases.

You cannot use templates to create relaxed-durability databases that use a durabilty of at_shutdown. You can use templates only for databases that have a durability of no_recovery.

Use the create database use *database_name* as template command to specify a database other than model as a template for any database that has been created with the no_recovery parameter.

Once you create a database that uses a database other than model as its template, Adaptive Server re-creates the dependent database, using data from the template database, when the server restarts.

When you restart Adaptive Server, template databases are recovered before databases that use them as templates. If Adaptive Server cannot recover a template database, databases that depend on that template database cannot be re-created.

Adaptive Server applies the attributes of the template database when you create the dependent database. Attributes you specify as part of the create database command override the template database's attributes. Database options and attributes for databases you create using the template persist when Adaptive Server restarts, and when Adaptive Server re-creates the database from its template. Any changes you make to attributes of the template database are not used when Adaptive Server re-creates the dependent database during subsequent restarts.

You cannot drop a database if any databases uses it as a template. You must first drop all databases that use this database as a template, or use alter database to detach the template database from all its dependent databases.

Note User-defined segments in template database may not function as expected after they are copied from the template database into an in-memory or relaxed-durability database. Segments direct space allocation onto specific database devices (or device fragments) by mapping the syssegments table to the sysuages table in the master database. Template databases may use a different mapping than the in-memory or relaxed-durability databases for which they are providing a template since the device layout for these databases may use a different number of devices and differently sized fragments. You must carefully plan and define the template database's user-defined segments before using these segments for in-memory or relaxed-durability databases.

Run sp_helpdb to report information about templates for:

- User databases – determine if, and for which databases, the user database has been used as a template. This section of sp_helpdb output shows that the pubs2 database was used as a template for the pubs3 and pubs5 in-memory databases:

  ```
  template_for
  ------------------------------------------------
  pubs3
  pubs5
  ```

- Databases created from a template – determine which database was used as a template during creation (if it was not model). This section of sp_helpdb output shows that you used the pubs2 database as the template for the pubs3 database:

  ```
  template
  ------------------------------------------------
  pubs2
  ```

Altering the database to use a new template

Adaptive Server does not change a database's existing data when you use alter database to change the template. Adaptive Server uses data from the new template to re-create the database when you restart Adaptive Server.

You can change the template only for in-memory databases and for databases that use a durability of no_recovery; you cannot change the templates for system or traditional disk-resident databases using full durability.

Minimally logged commands

In Adaptive Server version 15.5, you can perform minimal logging for data manipulation language (DML commands) on a per-database, per-table, and session-specific basis so that minimal row and page changes, as well as page allocations and deallocations, are logged.

You can control DML logging at the database, table, and session levels by configuring commands such as create database, alter database, create table, select into, set DML logging, and alter table.

See Chapter 3, "Minimally Logged DML."

Limits for in-memory and relaxed-durability databases

In-memory and relaxed-durability databases include these limits:

- The Cluster Edition does not currently support in-memory databases, relaxed-durability databases, template databases, or minimally-logged DML.
- Replication Server does not currently support replication of in-memory databases, or databases with durability set to no_recovery.
- You cannot use an in-menory database as an archive database.Sybase reommends that you do not use an in-memory database as a scratch database.
- In-memory databases do not support queries using compatibility mode. Use compatibility mode in Adaptive Server only on regular disk-resident database tables. See the *Migration Technology Guide* about enabling compatibility mode.

  If you have enabled compatibility mode and a query touches a table in an in-memory database, Adaptive Server reverts to the native version 15.0 query optimizer and execution engine using the “restricted compatibility mode.” Generally, this mode produces query plans similar to the plans in Adaptive Server version 12.5. If you notice a degradation in performance, Sybase recommends that you disable compatibility mode for this query.
- You cannot change the durability or logging level of in-memory or low-durability databases in the same command in which you are increasing the size of the database.
- alter database, when used to change either the durability or minimal_logging attribute of the database, automatically puts a database into single-user mode, and the command fails if it cannot acquire exclusive access to the database. To avoid this failure, manually put the database in single-user mode before running alter database.
- In-memory and relaxed-durability databases cannot participate in distributed transactions.
- In-memory and relaxed-durability databases cannot coordinate multidatabase transactions involving databases with full durability. When you execute system procedures that perform transactional updates when you run a stored procedure from an in-memory or relaxed-durability database, you see this error:

```
Msg 3952, Level 16, State 2:
Procedure 'sp_XX', Line 258:
```

```
Command not allowed. You cannot start a
multidatabase operation in database 'master'
after starting the master transaction in 'imdb1' as
it may render the database 'master' unrecoverable.
```

To run the system procedure:

a Execute it from a fully durable database. For example, to run sp_XX in master, enter:

```
use master
go
exec sp_XX
```

b Use this formate to reference the stored procedure from the current database (in-memory or relaxed-durability):
`database_name.owner.`***`sp_name`***.

For example, to run sp_XX in the imdb_1 in-memory database, enter:

```
use imdb_1
go
exec master.dbo.sp_XX
```

- From an in-memory database, you cannot use a proxy table or database to map to another in-memory database or disk-resident database object.
- From a disk-resident database, you cannot use a proxy table or database to map to an in-memory database or table.

Changed system procedures

Table 1-3 lists the stored procedures that have been changed to support in-memory storage caches, in-memory devices, and in-memory databases.

Table 1-3: System procedures changed for in-memory databases

System procedure	Comments
sp_addsegment	Updated to manage space in in-memory databases.
sp_addthreshold	Updated to manage space in in-memory databases.
sp_bindcache	You cannot bind objects or databases to in-memory storage caches, and you cannot bind an in-memory database or objects in an in-memory database to any cache.

System procedure	Comments
sp_cacheconfig	Creates, extends the size of, or drops, an in-memory storage cache.
sp_cachestrategy	The prefetch and MRU parameters do not apply to tables and indexes in in-memory databases.
sp_dbextend	Automatic database expansion is currently not supported for in-memory databases.
sp_deviceattr	The directio and dsync device attributes do not apply to in-memory devices.
sp_diskdefault	You cannot use sp_diskdefault to specify in-memory devices as a default device.
sp_downgrade	Supports downgrading an Adaptive Server containing in-memory or relaxed-durability databases, or databases using templates or minimal logging to an earlier version.
sp_dropdevice	Drops an in-memory device created from an in-memory storage cache.
sp_dropsegment	Updated to manage space in in-memory databases.
sp_dropthreshold	Updated to manage space in in-memory databases.
sp_extendsegment	Updated to manage space in in-memory databases.
sp_help	Reports properties, such as minimal logging attribute, for a table.
sp_helpcache	Displays properties of the in-memory storage cache, the in-memory database created on it, and details of free space on this cache.
sp_helpdb	Reports database properties such as durability, DML logging level, in-memory or not, use, if any, of a template database or as a template database.
sp_helpdevice	Reports the in-memory device properties created from an in-memory storage cache.
sp_modifythreshold	Updated to manage space in in-memory databases.

System procedure	Comments
sp_plan_dbccdb	Sets up dbccdb for checkstorage execution in an in-memory database.
sp_poolconfig	Large I/O buffer pools are not supported in an in-memory database.
sp_post_xpload	Supports cross-platform operations for in-memory databases.
sp_tempdb	Supports login or application bindings for an in-memory temporary database.
sp_unbindcache, sp_unbindcache_all	You cannot unbind objects to the in-memory database itself from the host in-memory storage cache.

CHAPTER 2

Managing In-Memory and Relaxed-Durability Databases

Specifying named caches for in-memory databases

Sybase recommends that you use huge pages for in-memory storage cache. See the *Configuration Guide* for your platform.

Caches that hold in-memory databases must be large enough to contain the entire database. A cache that contains an in-memory database is called in-memory storage, and disables:

- I/O during runtime operations
- Buffer washing
- Buffer replacement and washing

Once created, divide the in-memory storage cache into one or more pieces, each of which holds a fragment of the database or log. See “Cache and buffer support” on page 4.

Use sp_cacheconfig with the inmemory_storage parameter to create the in-memory storage cache. See the *Reference Manual: Procedures*.

Note Before you create the in-memory storage cache, verify that the value for max memory is sufficient for the specified cache size. If max memory is insufficient, Adaptive Server issues an error message.

For example, to create an in-memory storage cache named imdb_cache, enter:

```
sp_cacheconfig imdb_cache, '2G', inmemory_storage
```

Note For regular named cache, if the available memory size is less than the requested memory size for the cache, Adaptive Server creates the cache with the reduced memory size. That means the cache is created sucessfully, but with a smaller size. However, if there is insufficient space to create the cache for an in-memory database, the command fails.

Verifying changes to the configuration file

Verify that the configuration file (*$SYBASE/server_name.cfg*) correctly specifies in-memory storage cache information. Each in-memory storage cache includes a heading in the configuration file that is labelled "Named Cache: *cache_name*." The `Named Cache` entries include:

- `cache size` – size of the cache must be large enough to hold the entire in-memory database.
- `cache status` – set to "in-memory storage cache."
- `cache replacement policy` – set to "DEFAULT" or "none."
- `local cache partition number` – number of the local cache partitions or "DEFAULT."

An entry for a cache named imdb_cache looks similar to:

```
[Named Cache:imdb_cache]
cache size = 2G
cache status = in-memory storage cache
cache replacement policy = none
local cache partition number = 8
```

Changing static configuration parameters for in-memory databases

Adaptive Server makes changes to static configuration parameters when you restart it. Because in-memory databases are re-created when you restart the server (and any changes to them are lost), perform either of the following to make sure the data in the in-memory database is not lost when you restart Adaptive Server:

- Make all static configuration changes before the in-memory databases are created, or,
- Dump the in-memory database to an archive, make the static configuration change, restart the server, and load the in-memory database from the archive.

Creating in-memory devices

Use disk init to divide an in-memory storage cache into smaller pieces called in-memory devices, which are used to create in-memory databases. Sybase suggests that you use the same naming convention for in-memory devices that you use for disk-resident devices. Bind user- or system-defined segments with in-memory device logical names to control the space usage for objects bound to these segments.

Note You cannot use regular named caches to create an in-memory device. That is, you must use the type=inmemory parameter for disk init to create in-memory devices.

The syntax to create an in-memory device is:

```
disk init name = device_name
   physname = {"physical_name" | "cache_name"}
. . .
[, type = "inmemory"]
```

where *device_name* is the logical name of the in-memory device, *cache_name* is the name of the in-memory storage cache created with sp_cacheconfig, and inmemory indicates the device is used for an in-memory database.

For example:

```
disk init name = imdb_cache_dev,
physname = "imdb_cache" ,
```

```
size = "50M",
type = "inmemory"
```

Creating in-memory databases

Use create inmemory database to create an in-memory database, using model or another user database as its template.

You can also create temporary databases as in-memory databases that reside entirely in in-memory storage. However, you cannot specify a template database for an in-memory temporary database.

When you use a user-database template to create an in-memory database, all users, permissions, objects, and procedures are copied from the template database to the in-memory database.

When you create an in-memory temporary database:

- The guest user is added to the temporary database.
- create table privileges are granted to public.

You cannot create system databases (for example, sybsecurity) as in-memory databases, because these databases must be updated in the event of an Adaptive Server failure.

When you create in-memory databases, you can specify that insert, update, delete and some bulk-copy-in operations are minimally or fully logged on a per-database, per-object, or a session-specific basis. See Chapter 3, “Minimally Logged DML.”

This example creates an in-memory database on a 2GB in-memory device. with override allows you to create the data and log segments on the same in-memory device (the only durability level supported for in-memory databases is no_recovery: attempts to use another durability level result in an error):

```
create inmemory database imdb1
on imdb_data_dev1 = '1.0g'
log on imdb_data_dev1 = '0.5g'
with override, durability = no_recovery
```

Creating disk-resident databases with relaxed durability

Set the durability level for relaxed-durability databases to no_recovery or at_shutdown. See “Durability levels” on page 6.

You can create a relaxed-durability database on existing disks. A relaxed-durability database uses disk storage, so you must use disk init to create the device on which it resides.

This example creates the pubs6 database with a durability level of at_shutdown:

```
create database pubs6
on pubs6_dev
with override, durability=at_shutdown
```

Administering in-memory databases

Once you create in-memory databases, administer them by resizing and deleting the in-memory storage caches, increasing their size, and performing dumps and loads.

Resizing in-memory storage caches

Use sp_cacheconfig to increase the size of an in-memory storage cache at runtime. For example, to increase the size of imdb_cache to 3GB, enter:

```
sp_cacheconfig imdb_cache, '3G', inmemory_storage
```

You can reduce the size of an in-memory storage cache using the same procedure. You can reduce the size of the in-memory storage cache only by the amount of space on the cache that is currently unused by an in-memory database—in other words, the in-memory storage cache cannot be smaller than the in-memory database.

A reduced cache size is created when you restart the server.

Deleting in-memory storage caches

You must drop all devices and in-memory databases before you delete the in-memory storage cache. Delete the in-memory storage cache by setting its size to zero:

```
sp_cacheconfig imdb_cache, '0g'
```

Increasing the size of in-memory databases

Use alter database to increase the size of an in-memory database. The devices on which you increase the size must be part of the same cache that is currently hosting the devices on which the database resides (that is, you cannot create an additional in-memory storage cache and increase the size of an existing in-memory database on this storage cache). You cannot run disk resize to increase the size of in-memory database devices (as you would for a standard database device).

To increase the size of an in-memory database:

1 Use sp_cacheconfig to enlarge the in-memory storage cache.

2 Use disk init to create a second in-memory device on the enlarged in-memory storage cache.

3 Run alter database to extend the in-memory database onto the new database device you created in step 2.

Use the same steps that you use to increase the size of a standard disk-based database to increase the size of a relaxed-durability database.

Dumping and loading in-memory databases

Use dump database and load database to dump to or from an archive device for in-memory and relaxed-durability databases. Dumping and loading in-memory or relaxed-durability databases requires no special parameters for the dump or load commands.

load database into an in-memory database loads the data directly into the in-memory storage cache.

You can:

- Dump and load databases across durability levels. For example, you can dump a database with a durability level of full to an in-memory database, which always has a durability level of no_recovery
- Perform dumps and loads across platforms.

You cannot:

- Perform dump transaction from in-memory or relaxed-durability databases because load transaction cannot perform required tasks using a transaction log that may contain log records that are not ordered.
- Load a dump from a database with durability set to no_recovery or at_shutdown into a version of Adaptive Server that does not support in-memory or relaxed-durability databases. However, you can load a database dump from an earlier version of Adaptive Server in to an in-memory or relaxed-durability database.

Note You must use the version of Backup Server that ships with the version of Adaptive Server you use to dump and load in-memory or relaxed-durability databases.

Configuring *number of backup connections*

Dumping and loading an in-memory database requires the Backup Server to connect to Adaptive Server. The load command uses the same number of connections as there are stripes. The dump command uses same number of connections as there are stripes, plus an additional connection. Use number of backup connections to configure the maximum number of user connections the Backup Server can use.

See Chapter 5, "Setting Configuration Parameters," in the *System Administration Guide Volume 1*.

Backup Server requires the following to connect to Adaptive Server:

- The Backup Server interfaces file must have an entry for Adaptive Sever.
- The user name and password used by Backup Server to connect to Adaptive Server is the same as the user executing the dump and load commands. If the connection is established using secure external authentication (such as Kerberos), Backup Server cannot retrieve the password token from Adaptive Server. Use sp_remotelogin to define a trusted remote login for SYB_BACKUP, or Backup Server receives an authentication failure.

Dropping in-memory databases

Use drop database to remove an in-memory database. This example drops the pubs6 database:

```
drop database pubs6
```

Dropping an in-memory database removes the database from the system tables, and relinquishes the in-memory storage cache, although the buffers and data in these devices remain. Dropping an in-memory database does not affect the in-memory storage cache on which it was created, which can be used for other in-memory databases.

Dropping in-memory devices

Use sp_dropdevice to remove in-memory devices. sp_dropdevice removes an in-memory device only if it is currently unused by any database. You must first drop the database, then drop the in-memory device. Dropping an in-memory device deletes its entry from sysdevices and returns the memory to the cache on which it was created, where you can use it for any other purpose, including creating new in-memory devices. This example drops the device named pubs6_device:

```
sp_dropdevice 'pubs6_device'
```

CHAPTER 3

Minimally Logged DML

To optimize the log records that are flushed to the transaction log on disk, Adaptive Server can perform minimal to no logging when executing some data manipulation language (DML) commands—insert, update, delete, and slow bcp—on all types of low-durability databases, such as in-memory databases and low-durability databases that use the at_shutdown and no_recovery options. You can perform minimal logging for DMLs on a per-database, per-table, and session-specific basis.

Types of DML logging settings

The hierarchy of control for DML logging settings are:

- Database-level logging – by default, logging for DML is enabled at the database level for all tables. DML logging settings affect only user and temporary tables.
- Table-level logging – overrides the level of logging set at the database level, depending on how the table was created or altered.
- Session-level logging – overrides the logging level set at the table and database levels.

Database-level logging

Database-level support for enabling and disabling logging is provided primarily for temporary databases, where entire applications that do not rely on logging can run more efficiently, without changing the code for applications or procedures that create temporary tables

You cannot change the logging mode of system databases, including the model database. You can change the DML logging mode of the system tempdb—and any user temporary databases—to minimal logging. Before doing so, Sybase recommends that learn how changing to minimal logging affects rollback semantics on all applications that use temporary tables. See "Transactional semantics" on page 28.

Minimally logged DML is allowed only in a database that has a durability level set to no_recovery or at_shutdown. You must set the database's select into option to on for minimal logging to take effect.

You can alter the default logging mode for a database only from the master database. In addition, the database you are altering must:

- Be in single-user mode for user databases
- Have the dbo-use only set to true (on) for temporary databases, so that only the database owner can use the databases

If the database is not already in the required mode, the server tries to put the database in that mode. If the attempt is unsuccessful, the server raises an error, prompting the user to explicitly put the database in its correct mode.

Commands

The syntax to change the DML logging mode at the database level is:

```
create [temporary] database database_name
    [on {default | database_device [= size]
                [, {database_device [= size]...]
    [log on {database_device [= size]
          [, {database_device [= size]]...]
    [with {override | default_location = "pathname"
    | [[,]durability = { no_recovery | at_shutdown | full} ]
    | [[,]dml_logging = {full | minimal} ]
    }...
]
    [for {load | proxy_update}]
```

To change the database-level setting of DML logging in an existing database, use:

```
alter database dbname
set dml_logging = {full | minimal}
```

See the *Reference Manual: Commands*.

Table-level logging

Setting the DML logging option at the table level overrides the database-level setting, depending on how the table was created or altered. The default mode is to use the same setting as the database.

If minimal logging is not enabled at the database level:

- Minimal DML logging is done for tables that have dml_logging explicitly set to minimal via create table or alter table.
- DML is fully logged for all other tables.

If minimal logging is enabled at the database level:

- Full DML logging is done for tables that have dml_logging explicitly set to full.
- Minimal DML logging is done for all other tables.

The default logging setting of a table allows it to inherit—when the DML is executed—the then-current database-level setting for logging. The database administrator may want to periodically turn logging off, then back on again, at the database level. Then, only those tables with specific needs for full or minimal DML logging need to be controlled through explicit table-level settings.

You can execute minimally logged DML commands on a table only if the database has the select into database option turned on; otherwise, all DML commands are fully logged.

For any DML statement on a table where the corresponding trigger is enabled (for example, if an insert trigger is enabled when you execute an insert statement) Adaptive Server performs full logging. This avoids situations where a trigger implements business rules and security mechanisms that need the log records while executing the trigger. To perform minimal logging, specific triggers must be disabled by the table owner before executing the DML statement.

If a DML statement is executed on an view that can be updated, and it eventually resolves to a DML statement on a base table that is eligible for minimally logged DML operations, DML statements executed on the view result in minimally logged DML on the base table. To control the logging mode on base tables that are updated through views, use alter table or set dml_logging to set the logging mode on the underlying table.

You can use alter table to change only the logging mode of user tables, and not of views or other objects.

Commands

To create a table with full or minimal DML logging, use:

```
create table tablename (
    <rest of the column list specifications>
)
lock lock_scheme
with { max_rows_per_page = num_rows
    , exp_row_size = num_bytes
    , reservepagegap = num_pages
    , identity_gap = value
    , dml_logging = {full | minimal}
    }
on segment_name
```

To create a table using select into so that the target table has DML logging enabled or disabled, use

```
select <column list>
into table_name
    [ <external table specifications> ]
    on segment_name
    [ partition_clause ]
    lock lock_scheme
    with { max_rows_per_page = num_rows
        , exp_row_size = num_bytes
        , reservepagegap = num_pages
        , identity_gap = value
            , dml_logging = {full | minimal}
        }
[ from_clause ]
[ where_clause ]
...
```

Note In version 15.5, you cannot unconditionally turn on logging on a table when the table owner created the table explicitly turning off logging. This prevents large DMLs from generating huge amounts of logging on a table that was created to use minimal logging.

Session-level logging

Session-specific setting of the logging option overrides the table-level and database-level setting of the logging option.

To enable or disable logging for DML in the current session, even when database-level, or table-specific DML logging is full, use:

```
set dml_logging = { full | minimal | default}
```

See the *Reference Manual: Commands*.

Setting the DML logging to minimal affects only the logging mode on objects owned by the current session user. If the session user has the sa_role, the logging mode of all user objects is minimal.

Once you have set session-specific DML logging to minimal, set dml_logging default returns the logging mode currently in effect for the affected tables to the table's default logging mode, based on the table- and database-level settings.

You can use the set dml_logging command to perform minimal logging for a table, but you cannot use it to perform fully logged DML if the database owner or table owner has already set up the table to run with minimal logging.

The session-specific setting for the logging mode eventually determines whether a particular object is logged, given the various rules described earlier and in the following sections, regarding choice of the logging mode for a specific table.

- Upon a successful set command invocation, all tables in the current session become candidates for consideration on choice of the logging mode, subject to permissions and privileges.
- You can use the set command to change the logging mode only for user tables, and not for other objects like system tables, views and so on.
- Minimal DML logging requires that the select into database option is turned on, which requires a database owner or sa_role privilege.

Additional minimal logging rules

In addition to the basic rules related to databases, tables, and sessions, these rules also affect minimal logging:

- You can use minimally logged DMLs only in in-memory or relaxed-durability databases. You cannot use them in databases that have full durability.
- The logging mode for a table in a multistatement transaction remains unchanged throughout the entire transaction.
- All DML commands are fully logged on tables that participate in declarative or logical referential integrity constraints.

- You can export the set dml_login option from a login trigger to a client session. However, unlike most set options, you cannot export set dml_logging from stored procedures or issue execute immediate to their callers, even when you enable set export_options.
- All DML commands after a savepoint is set executes with full logging even though the table would have otherwise qualified for minimal logging.
- Full logging is performed if any active triggers exist on the table. For DML to run in minimal-logging mode, disable any triggers before executing the DML statement.
- An optimizer selecting deferred-mode update processing overrides the minimal DML logging setting, and the DML is executed with full logging.
- To support log-based replication, DML on replicated tables always performs full logging.

Transactional semantics

When operating in minimally logged mode, the atomicity of transactions is not guaranteed after a runtime rollback.

Because logging is incomplete at runtime, you cannot completely roll back the changes made by a failed command. All changes are applied to the database in commit mode. For example, you cannot roll back a transaction that deleted a number of rows; the changes to the deleted rows are already committed. If a transaction deletes rows from a page that is then deallocated, the page remains deallocated.

However, not logging changes does not interfere with locking the rows or pages affected. Even if a DML command runs in minimally logged mode with no rollback available, the locks on the affected rows and pages are acquired and held until the end of the transaction, and are released only upon transaction completion, be it rollback or commit.

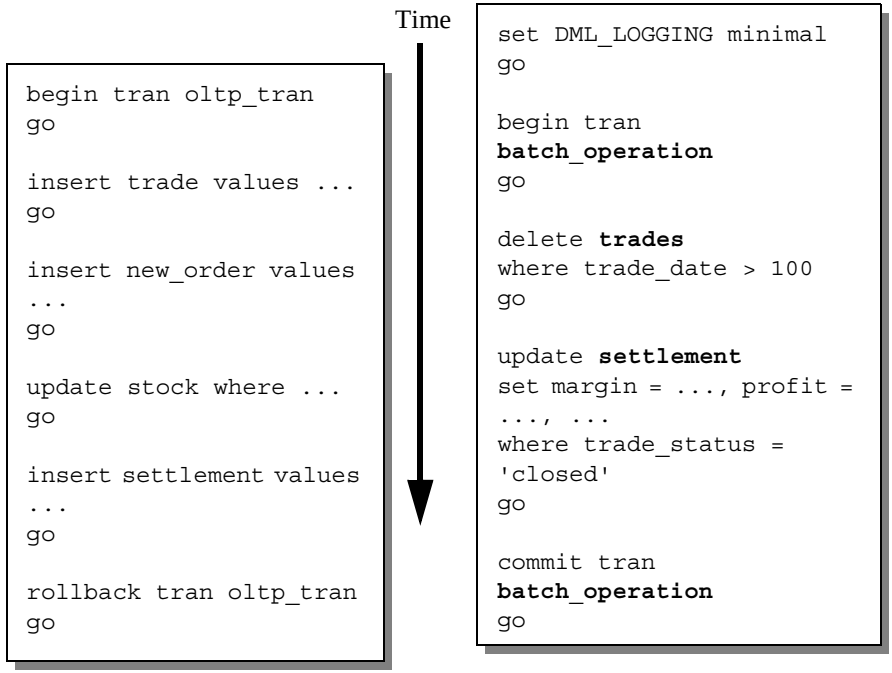
Issuing a rollback command when the transaction was executed with minimal logging on one or more tables raises a warning that Adaptive Server is committing the transaction at the point where it encountered the rollback.

Logging concurrent transactions

The set dml_logging command affects the logging mode only in the current session, and only for DML statements on tables owned by the session's user. This allows DML commands to execute concurrently from multiple transactions, where full logging is employed in one session executing concurrently with minimal logging from another session. Any rollback in the session executing with full logging is undone, whereas the changes are not undone in a rollback from the other session.

A common example of such a usage is the execution of a transactional system, running small transactions concurrently with large batch updates or deletes in minimally logged mode. Any errors in either session do not affect the transactional consistency of the other session. In the example below, the OLTP transaction runs with full logging, enabling complete recoverability, whereas the batch operation runs from another session with minimal logging.

Figure 3-1: Concurrent execution of transactions with different logging modes



Minimal logging with *ddl in tran* set to true

If the ddl in tran database option is set to true, you cannot execute a DML statement in minimally logged mode on a table that has already had any DDL operations executed against it in the same transaction. The DML commands in the transaction execute with full logging.

In the following transaction, the sp_dboption database option ddl in tran is set to true, and table t1 is configured with minimal DML logging. However, at run-time Adaptive Server executes the DML commands following the drop index command with full logging because it executed the drop index command with full logging:

```
sp_dboption pubs1, 'ddl in tran', on
go

begin tran
go

update t1 set ...
go

drop index t1.ind1
go

insert t1 values ...
go

insert t1 values ...
go

delete t1 where ...
go

update t1 where ...
go

rollback tran
go
```

The create index and drop index commands affect how minimal logging is selected when they are executed in the same transaction (that is, either of these commands would have caused Adaptive Server to perform full instead of minimal logging in the script above).

Effect of referential integrity constraints

When tables are involved in referential integrity constraints, the rules for selecting minimal logging are:

- All DML statements on a table with referential integrity constraints are always fully logged, overriding any logging settings arrived at for the table based on the database-level or session-specific settings.
- You cannot use alter table to change the logging mode of a table to minimal if the table includes referential integrity constraints.
- You cannot create any referential integrity constraints between tables that have minimal logging defined for them. For example, creating a foreign referential integrity constraint from a table to its primary table when minimal logging has been explicitly set on the primary table raises an error.

Multistatement transactions in minimally logged mode

Follow these rules to use minimally logged mode DML in an explicit begin transaction, followed by multiple batches of DML statements.

- You may use DML statements that operate on different tables that are either in fully logged or minimally logged mode in a multi-statement transaction. You can use the set dml_logging command to change the logging mode partway through a transaction, but its effect on subsequent DML statements varies, depending on what other DML operations have already been executed in the same transaction. The following SQL transaction is allowed, assuming that the logging mode is full when you start the transaction:

```
begin tran
go
delete t1 where ...
go
set dml_logging minimal
go
insert t1 values ...
go
update t2 where ...
go
commit tran
```

```
go
```

`delete t1` is performed with full logging, but `update t2` is performed in minimally logged mode. In the case of a rollback, the update is not undone, but the delete is rolled back.

- Once you execute a command with full or minimal logging on a table in a multi-statement transaction, subsequent commands on the same table must use the same logging (full or minimal), regardless of the session's dml_logging setting. In example above, the insert t1 is executed with full logging because the earlier delete on t1 which was executed with full logging.
- Conversely, resetting the session's logging mode to the default mode may not resume logging for a DML statement if the table was previously operated on in a minimally logged mode in the same transaction.
- Mixing the logging mode for different tables within the same transaction has different results for the tables involved if the transaction is rolled back. Changes to tables with full logging are rolled back, whereas changes to tables with minimal logging remain committed.
- The rules for logging mode choice and use of set dml_logging command inside stored procedures are identical to those for a multi-statement batch. If:
  - The procedure is run outside an explicit transaction, then each statement is executed as an individual transaction.
  - The procedure is run inside a transaction, then the same rules described above apply.
- There is no restriction on changing the logging mode inside a transaction and then executing select from a table that was previously operated on in the same transaction in a different logging mode. The delete t1 is performed in logged mode, while the update t2 is performed in minimally logged mode. Referencing the same table t1, for read, which was once operated on in fully logged mode when the logging mode is now minimal is not an error.

```
begin tran
go

delete t1 where ...
go

set dml_logging minimal
go
```

```
update t2
where t2.c2 = (select c1 from t1 where ...)
go

commit tran
go
```

Stored procedures and minimally logged DML

The DML logging setting in a session is inherited from called system procedures and the behavior setting for any set dml_logging command executed in a procedure is inherited by subprocedures. After exiting from the procedure's scope, regardless of whether set export_options is on, the dml_logging setting in the parent session or parent procedure is restored.

Examples

These examples show how these rules are applied to affect the logging mode of tables inside a procedure.

Example 1 In this example, the user executing the procedure is also its owner and the owner of all tables affected by this procedure:

```
create procedure p1 as
begin
    delete t1 where...

    set dml_logging minimal

    update t2 where...
end
go

set dml_logging default
go

exec p1
go
/*
** Exiting from the procedure restores the
** session's setting to what it was before
** calling the procedure, in this case, the
** logging mode will be back to DEFAULT
** (i.e. FULL).
*/
```

```
-- This will operate in logged mode now.
delete t2
go
```

1 When execution starts in the procedure p1, logging mode is full.

2 `delete t1` is performed in fully logged mode, and then the procedure's session-level logging mode is changed to minimal.

3 `update t2` is performed in minimally logged mode.

4 Upon exit from p1, when control returns to the outer SQL batch, logging mode setting is back to full. The next delete t2 is performed in fully-logged mode.

Example 2 This example executes procedure p1 with the session-level logging mode set to minimal. delete t1 operates in minimally logged mode, and so does update t2. Once p1 is done, the next delete t2 also operates in minimally logged mode, as the logging mode has been restored to what it was before p1 was called.

```
set dml_logging minimal
go

exec p1
go

-- This will operate in minimally logged mode.
delete t2
go
```

As the logging mode for DML statements inside a procedure is affected by the session-level setting of the calling session or procedure, Sybase recommends that you explicitly select the desired logging mode at the start of the procedure body. You may optionally set it back to default when the procedure finishes.

```
create procedure p1 as
begin
    set dml_logging minimal

    delete t1 where ...
    update t2 where ...

    -- Optionally, reset upon exit
    set dml_logging default
end
go
```

However, if the DML statements inside a procedure are executed mostly in a single-logging mode (for example, full) but must occasionally run in a different (minimally logged) mode, Sybase recommends that you control the logging DML statements with the session-level setting from the calling procedure or isql session instead of including the logging mode inside the body of the procedure.

Example 3 The DML logging setting affects only those tables owned by the session's user. This affects procedures that must perform minimally logged DML on certain tables, but which are executed using the exec proc privilege by a user who does not own those tables.

In this example, Joe executes the procedure mary.delete_proc, which performs a delete on a table owned by Mary. Joe uses the set command to request minimal logging, but doing so affects the logging mode for tables owned by Mary in the procedure:

```
isql –Sservername –Umary –Pmaryspwd

create procedure mary.delete_proc as
begin
    delete mary.large_table where ...
end
go

grant exec on mary.delete_proc to joe
go

isql –Sservername –Ujoe –Pjoespwd

-- User 'joe' executes the following SQL:
--
set dml_logging MINIMAL
go

exec mary.delete_proc
go
```

Example 4 Adaptive Server does not permit the procedure owner, Mary, to allow minimal logging for certain statements when the procedure is executed by a user who does not own the table. The set dml_logging command inside a procedure applies only to those tables owned by the session's owner.

In this example, minimal logging does not apply to `delete mary.large_table`, but it dos apply to the `update joe.very_large_table` when user Joe executes the procedure with default logging settings.

```
isql –Sservername –Umary –Pmaryspwd

create procedure mary.delete_proc2 as
begin
    set dml_logging MINIMAL

    delete mary.large_table where ...

    update joe.very_large_table where ...
end
go

grant exec on mary.delete_proc2 to joe
go

isql –Sservername –Ujoe –Pjoespwd

exec mary.delete_proc2
go
```

If a procedure performs DML statements on tables owned by multiple owners, then, depending on which user executes the procedure, the set of tables against which minimally logged DML is executed changes. Only the table owner or user with sa_role can execute the procedure that performs the minimally logged DML on specific tables.

Recompiling a running procedure, or the cached plan of a previously executed procedure, does not affect the logging mode chosen at runtime for individual DML commands that appear in the procedure body. Any set commands that might change the logging mode for a table are taken into consideration at the start of execution of the DML statement.

Including *set dml_logging* in a trigger

DML statements are fully logged if there is an active trigger for that DML operation on the table. If a trigger is created and logging is disabled on the table, a warning is raised, indicating that the DML statements will operate in fully logged mode. However, the trigger is successfully created.

The logging mode of DML statements in triggers varies, depending on the user who fired the trigger. Tables owned by the same user who fired the trigger can be operated on in minimally logged mode. DML statements executed on tables that are not owned by the user who fired the trigger are executed in fully logged mode, unless these tables are explicitly set up for minimal logging.

Consider delete_trig_m1, a delete trigger on the object m1 owned by user Mary. This trigger performs DML statements on other tables, such as Joe.j2 and Paul.p3, in minimally logged mode. delete privilege has been granted on Mary.m1 to user Sally.

```
Create trigger Mary.delete_trig_m1 FOR DELETE
On Mary.m1
as
Begin
    Delete Mary.min_logged_table_t3
    WHERE ...

    SET DML_LOGGING MINIMAL

    DELETE Paul.p3 WHERE ...
End
```

When user Sally executes the delete statement on Mary.m1, the trigger, Mary.delete_trig_m1 is fired and privilege checks are performed on the appropriate tables on behalf of Sally. Because Mary owns the trigger, Adaptive Server peforms no permission checks for the `delete Mary.min_logged_table_t3` statement, where the DML statement is run in minimally logged mode. (The table has been defined for minimal logging through some other means.) Because Sally does not have the privilege to turn off dml_logging for Paul.p3, the set command does not take effect silently. The next statement, `delete Paul.p3 where...`, runs in fully logged mode. When user Mary executes the trigger, the DML logging behavior is the same (fully logged).

However, if user Paul executes the outer delete on Mary.m1, causing the trigger to fire, then both the statements in the body of the trigger execute in minimally logged mode.

Once the trigger fires, the outer DML statement must have been fully logged; attempts to perform minimally logged DML on the same outer table in the body of the trigger are ignored, and these DML statements are fully logged.

Logging mode cannot be disabled on views, so instead of triggers, which are currenly only supported on views, are unaffected by the logging mode of the base tables referenced by the view, or by the session-level setting of the logging mode when the DML statements on the view with the instead of trigger executes. However, the rules for chosing logged or minimally logged modes on multiple tables, using of the set command inside the trigger body, and transactional semantics all apply for the DML statements inside the instead of trigger.

Using deferred updates

If a query optimizer picks a deferred-mode update for a table that qualifies for minimally logged operations, the minimally logged settings are overridden for that statement and the statement works in deferred mode, but with full logging. Applications that generate DML statements in deferred mode with large transaction logging do not benefit from minimal logging. Once you execute a deferred-mode operation for multistatement transactions on a table, all subsequent DML statements on this table are fully logged.

Obtaining diagnostic information

Many rules interact to determine the logging mode of a table. Application developers may need to know whether Adaptive Server is currently generating minimally logged DML commands for a particular table, and the type of schema, constraints, session settings, and so on.

object_attr reports the table's current logging mode, depending on the session-table- and database-level settings. See the *Reference Manual: Building Blocks*.

Use set print_minlogged_mode_override to determine whether Adaptive Server has overwritten your logging-mode choices (see the *Reference Manual: Commands*). print_minlogged_mode_override generates trace information to the session output, reporting on the statement for which the minimally logged mode of a table has been overwritten by other rules, such as presence of referential integrity constraints, deferred mode choice, name of the table affected, a description of the affecting rules, and so on. Enable this switch server-wide to capture diagnostic output from the entire application. Redirect this output to the error log using the print_output_to_log switch.

CHAPTER 4

Performance and Tuning for In-Memory Databases

This chapter discusses performance and tuning characteristics of in-memory and relaxed-durability databases, including caches, sp_sysmon results, monitor counters, and Backup Server.

Configuring in-memory storage cache

Adaptive Server supports a wide variety of database and cache configurations for all database types using named caches and database bindings. Consider the following when sizing or binding a cache:

- Bind large, disk-resident databases to smaller named caches. This is a standard Adaptive Server configuration, with only a small portion of the database's contents in cache. Depending on the workload, much of the database page space may get recycled due to normal cache replacement policies.
- Make the cache size similar to the database size, which may reduce the recycling of cached pages. However, depending on the workload, writes to disk may be high.

- Bind entire temporary databases to a named cache that is large enough to host the entire temporary database. Default optimizations the server applies to temporary databases (and using strategies such as delayed commit) allow you to significantly improve the performance of fully cached temporary databases.

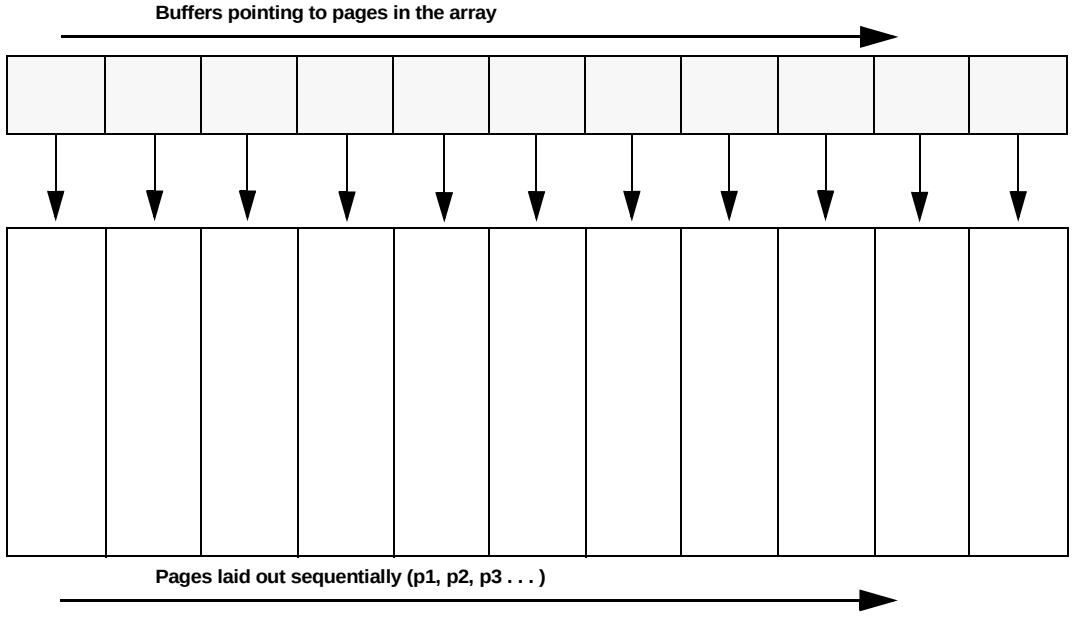
Cache layout

Relaxed-durability databases, and objects in them, that are bound to any named cache use the same layout as regular caches.

The access pattern of the object bound to the cache dictates the layout for a regular cache. Adaptive Server selects the least recently used (LRU) buffer and page pair for replacement with strict LRU cache replacement policy. Adaptive Server loads pages for objects bound to the named cache that reside on disk (identified by the database, table, and index ID) into the cache when they are requested. These objects remain in the cache until Adaptive Server selects them for replacement. Adaptive Server determines the location in the cache where this page is read, according to the availability of free buffers or according to which buffers are replaceable, depending on the LRU or most-recently used (MRU) strategy. The buffer Adaptive Server uses to hold a page in memory might vary, and the location of the buffer in the buffer cache may also vary in this cache configuration

Databases that use in-memory storage caches are bound to the cache, and all the objects and their indexes in this database use the same cache. The entire database is hosted in the cache. All the pages in the database, both allocated and unallocated, are hashed, so page searches in the cache should always find the required page. Pages are laid out sequentially (described in Figure 4-1), and because all the pages reside in the cache, the position of the buffer and page pair does not change.

Figure 4-1: Pages arranged sequentially in an in-memory database



In-memory storage cache supports only the default pool (which uses the server page size for any logical read or write on that page). In-memory storage caches do not need large buffer pools, which are used to perform large I/O from disk. Asynchronous prefetch, as a strategy to improve runtime I/O performance and to reduce cache misses while searching for a page, is not supported for in-memory storage caches.

Because they do not store any data in disk, in-memory storage caches:

- Do not use any replacement policy (the replacement policy is defined as none for each cache). The pages in the cache never change positions.
- Do not support I/Os, so the wash region and large pools are not supported.
- Need not perform buffer replacement. Occasionally, for private buffers, the default data cache is used.
- Need not dynamically hash buffers. All pages are loaded and hashed in the hash table when you create the database. In memory databases may need to hash buffers at runtime if you increase the size of the database with alter database.
- Need not perform asynchronous prefetch, because all pages reside in the cache memory.

Cache partitions are supported for in-memory storage caches, and reduce the contention on a single spinlock by distributing subsets of pages in an in-memory database to a partition, where each subset is controlled by a separate spinlock.

sp_sysmon output for in-memory databases

Run sp_sysmon for in-memory databases to monitor their performance.

Cache Hits should always have a value of 100% for an in-memory database cache, and Cache Misses should always have a value of 0:

```
Cache: imdb
                              per sec      per xact       count   % of total
-----------------------   ------------  ------------  ----------   ----------
Spinlock Contention                n/a           n/a         n/a      50.9 %

Utilization                        n/a           n/a         n/a      99.9 %

Cache Searches
Cache Hits                     16954.4        5735.8     1152897     100.0 %
Found in Wash                      0.0           0.0           0       0.0 %
Cache Misses                       0.0           0.0           0       0.0 %
-----------------------   ------------  ------------  ----------   ----------
Total Cache Searches            16954.4         5735.8     1152897
```

In-memory caches should not have a wash region, as reported below:

```
Buffer Wash Behavior
   Statistics Not Available - No Buffers Entered Wash Section Yet

Cache Strategy
Cached (LRU) Buffers          21225.3         7180.7     1443321      100.0 %
Discarded (MRU) Buffers           0.0             0.0           0        0.0 %
```

See *Performance and Monitoring Series: Monitoring Adaptive Server with sp_sysmon*.

The following sp_sysmon sections do not apply to in-memory storage caches because the entire database is bound to the cache and all database pages reside in the cache:

- Utilization
- Cache Hits

- Found in Wash
- Cache Misses
- Large I/O Usage
- Pool Turnover

These sections do not apply to in-memory databases because they do not perform buffer grabs:

- LRU Buffer Grab
- Grabbed Dirty
- Total Cache Turnover

These sections do not apply to in-memory databases because they do not define a wash region. Disk reads and writes do not take place in a steady state:

- Buffer Wash Behavior
- Cache Strategy

These sections do not apply to in-memory databases because they do not use large pools:

- Large I/Os Performed
- Large I/Os Denied
- Total Large I/O Requests
- Large I/O Detail

Monitoring the default data cache performance

Because all pages are hashed in an in-memory storage cache, tasks requiring a small number of private buffers for storing intermediate pages, for example during sorting, use the default data cache. Use the buf_imdb_privatebuffer_grab monitor counter to determine if Adaptive Server is temporarily using the default data cache for sort buffers:

```
buf_imdb_privatebuffer_grab  buffer_0          244              510525
```

This script collects monitor counters from all monitoring groups. It first clears all the monitor counters, samples the monitor counters for one minute, and reports values for all monitor counters updates for that monitoring interval.

To determine the current value for buf_imdb_privatebuffer_grab, run:

```
dbcc monitor ('clear','all','on')
go
dbcc monitor ('sample','all','on')
go
waitfor delay "00:01:00"
go
dbcc monitor ('sample','all','off')
go
dbcc monitor ('select','all','on')
go
select * from master.dbo.sysmonitors
where field_name = 'buf_imdb_privatebuffer_grab'
go
```

The value for buf_imdb_privatebuffer_grab indicates the default data cache's buffer usage from queries that require temporary buffers for intermediate sorting that are run against tables in an in-memory database. Evaluate the value for buf_imdb_privatebuffer_grab according to the size of the default data cache:

- If the number of buffers grabbed is very low compared to the size of the default data cache, queries run against tables in in-memory databases are not heavily using temporary buffers.
- If the number of buffers grabbed is a significant portion of the number of buffers in the default data cache, it indicates a heavy load on the default data cache for buffers used for queries running against in-memory databases. This generally occurs only when the default data cache is very small, for example, if it is using the default size of 8MB.

  Using a small default data cache may affect the performance of other applications that rely on the default data cache. Increase the size of the default data cache to accommodate both requests for temporary buffer usage that come from in-memory databases, and to accommodate other concurrent applications using the same cache.

See Chapter 2, "Monitoring Performance with sp_sysmon," in *Performance and Tuning Series: Monitoring Adaptive Server with sp_sysmon*.

Organizing physical data for in-memory devices

Because in-memory databases do not use I/O, you need not consider device I/O characteristics (for example, the speed of the I/O device) when organizing physical data placement. You need not consider issues like locating frequently accessed index pages on fast devices, or pages for other infrequently utilized objects (for example, text and image pages) on slower devices. However, because in-memory databases remove other bottlenecks caused by disk I/O, other bottlenecks such as spinlock and latch contention may be high.

Consider the following when you organize physical data for in-memory devices:

- To reduce latch contention, use separate data and log devices (that is, do not configure in-memory databases for mixed log and data).
- To control log space consumption, use separate log devices.
- To restrict object space usage, organize devices in segments.
- To reduce contention for page allocation, use segments bound to different devices.

Performance optimization for low-durability databases

Low-durability databases have their durability set to no_recovery and at_shutdown. Because low-durability databases are not recovered, they provide these performance benefits:

- Low-durability databases do not flush the user log cache (ULC) due to unpinning, therefore increasing transaction throughput in high-transaction systems.

  ULC unpinning occurs when two transactions attempt to update the same datarows-locked data page, causing the second transaction to flush the log records from the first transaction's ULC to syslogs, before the second transaction uses the data page. Increasing ULC flushes to syslogs increases log contention, adversely affecting transaction throughput in high-transaction systems.

The Transaction Management section in sp_sysmon output shows that the values for ULC Flushes to Xact Log, by Unpin, and by Single Log Record (all actions associated with unpinning) are noticeably lower for low-durability databases than in full-durability databases. This increases the transaction throughput in high-transaction systems. This sp_sysmon output from a low-durability database shows no ULCs flushed from unpinning:

```
ULC Flushes to Xact Log    per sec       per xact    count  % of total
-----------------------  ----------   ------------  -------  ---------
Any Logging Mode DMLs
by End Transaction             40.3            0.0     2416   100.0 %
by Change of Database           0.0            0.0        1     0.0 %
by Unpin                        0.0            0.0        0     0.0 %
by Other                         0.0            0.0        0       0.0 %
```

- Low-durability databases do not perform transaction log flushing – during a commit, because transactions are not written to disk.

  Transaction log flushing is typically unnecessary for relaxed-durability databases; they do not require reliably recorded committed transactions because they are not recovered during a restart. Adaptive Server typically flushes the transaction log when a transaction completes and, for relaxed-durability databases, during the buffer wash.

  The Device Activity Detail section in sp_sysmon output shows the number of writes for log devices of relaxed-durability databases. This shows the sp_sysmon output for a disk-resident, version 15.0.3 database:

```
Device:
/disk_resident/device/1503esd2/drdb_device.dat
drdb_device               per sec      per xact      count  % of total
-------------------   -----------  ------------  ---------  ----------
Reads
  APF                         0.0           0.0          0       0.0 %
  Non-APF                    14.3           0.0       2380       0.2 %
Writes                     9247.3           1.5    1544304      99.8 %
-------------------   -----------  ------------  ---------   ----------
Total I/Os                 9261.6            1.5    1546684     100.0 %
```

  This shows the same output for a relaxed durability database from a version 15.5 Adaptive Server. Because this is a relaxed-durability database, the writes are about 23 percent of the writes for the disk-resident database:

```
Device:
/relaxed_durability/IMDB/devices/ariesSMP/rddb_device.dat
rddb_device               per sec      per xact      count   % of total
```

```
------------------   -----------  ------------  ---------   ----------
Reads
  APF                        0.0           0.0           0       0.0 %
  Non-APF                   14.7           0.0        1031       0.7 %
Writes                    2106.4           0.1      147446      99.3 %
------------------   -----------  ------------  ---------   ----------
Total I/Os                2121.1           0.1      148477      99.9 %
```

- Low-durability databases do not log transactions contained in the ULC. Adaptive Server discards log records for transactions in low-durability databases when it commits transactions, if all log records for each transaction are fully contained within the ULC, and the transaction does not require the log records for any post-commit processing.

  Because you need not transfer the log records, the amount of contention on the log decreases, which increases throughput in high-transaction systems. Generally, logging transactions in the ULC favors smaller transactions because large transactions cannot contain all their log records in the ULC at commit time; large transactions that log more records than fit in the ULC must flush the ULC to syslogs when it fills.

  ULC flushing may also occur when small transactions are contained within the ULC at commit time because Adaptive Server requires these log records to perform post-commit work. For example, ULC flushing occurs in any transaction that deallocates any space within the database when the transaction requires post-commit work.

  See “ULC Flushes to Xact Log” in Chapter 2, “Monitoring Performance with sp_sysmon,” in the *Performance and Tuning Series: Monitoring Adaptive Server with sp_sysmon*.

- Low-durability databases do not flush partially logged changes to disk. Adaptive Server must occasionally flush data pages for full-durability databases to disk because the corresponding log records do not fully describe the changes that took place. Some examples are:
  - Any index- or data page split for a table with a clustered index
  - A sort
  - A writetext command
  - Fast bcp
  - alter table to change a table’s locking scheme
  - alter table to change a table’s partitioning scheme or a table’s schema
  - A full table reorg rebuild

- alter table...unparition

  In-memory databases need not flush data pages to disk because they do not use disks. Relaxed-durability databases do not flush data pages to disk because there is no need for database recovery. Not having to flush data pages to disk can significantly improve performance when the number of disk I/Os would be high, for example, where changed pages are flushed to disk at the end of a sort operation.

Additional improvements to low-durability databases that are not directly related to their durability, occur internally, and require no administration:

- Improvements for updating the database timestamp (a frequent operation in a high-transaction database)
- Improvements to deletes using an index scan for data-only-locked tables
- Improvements for bulk inserts into a data-only-locked table with non-unique indexes

These improvements also apply to temporary databases that have durability explicitly set to no_recovery (through create database or alter database); they do not apply to temporary databases with durability implicitly set to no_recovery.

Tuning checkpoint intervals

In addition to determining the length of the recovery time, the recovery interval in minutes configuration parameter determines how frequently Adaptive Server checkpoints a database. In-memory databases are never checkpointed, but Adaptive Server does checkpoint relaxed durability databases, flushing all modified buffers from disk according to recovery interval in minutes.

Use recovery interval in minutes to reduce the pressure on buffer washing, and to maintain replaceable buffers.

The lower the value for recovery interval in minutes, the more frequently Adaptive Server performs a checkpoint and washes all changed buffers. However, you must balance the benefits of performing the buffer wash with the number of disk I/Os that occur while Adaptive Server is performing the checkpoint.

When you set recovery interval in minutes to higher values, you must balance the benefit of a smaller number of disk I/Os that result from less frequent checkpoints with the possibility that a request for a buffer may find it “dirty,” delaying its use while the buffer wash takes place.

recovery interval in minutes applies to all databases server-wide, and you should change this parameter only after evaluating the impact this change has on full-durability databases. If you have at least one full-durability database that generally needs significant recovery after a server crash, continue to use the value required for a timely recovery of this database (generally, the default value of 5 minutes). If a change has little impact on full-durability databases (that is, they require little recovery after a server failure), start with a recovery interval higher than the default value of 5 minutes (for example, 30). After the change, view the Buffers Grabbed Dirty value or the per-cache information in the Data Cache Management sections of sp_sysmon. If the Buffer Grabbed Dirty value is high, decrease the value for recovery interval in minutes.

To decrease the number of I/Os to disk checkpoint performs for relaxed-durability databases without affecting the behavior of full-durability databases (regardless of the value for recovery interval), set the no chkpt on recovery database option to true for the relaxed-durability database. Use the method described above to evaluate the value for Buffers Grabbed Dirty and verify that disabling checkpoints does not negatively impact the availability of reusable buffers.

Minimally logged DML

During minimally logged DML, insert, update, delete, and slow bcp-in commands are performed with minimal or no logging. If a statement fails (for example, from an internal error or a user-issued rollback), the portion of the work already completed stays committed, and is not rolled back. However, when configured for minimally logged DML, Adaptive Server must maintain logical consistency for changed tables (for example, rolling back a data row insert must result in a rollback of the related index rows). To ensure this consistency, minimally logged DML commands are divided into elementary unit operations named subcommands, which are all the data changes required to perform a single row change, including those for indexes and any text, image, or off-row columns. To maintain the logical consistency at the subcommand level, Adaptive Server logs minimally logged DML commands in the user log cache (ULC). Once a subcommand completes, Adaptive Server discards the log records for the subcommand from the ULC, there is no need to flush the ULC to syslogs.

If a ULC is not large enough to contain all of a subcommands' log records, Adaptive Server may be unable to discard the log records. This generally happens if the data row affected by the DML is very large, or if a table contains many indexes. If Adaptive Server cannot discard the ULC, it flushes the ULC log records to syslogs, which:

- Increases log contention in a busy system
- Impedes transaction throughput
- Increases the amount of log space required, which offsets any benefits gained by using minimally logged DMLs

Make sure that ULCs are large enough to contain the log records for most subcommands. Sybase recommends that your ULC be twice as large as the default server page size:

```
declare @ulc_size int
select @ulc_size = @@maxpagesize * 2
exec sp_configure "user log cache size", @ulc_size
```

For in-memory temporary databases or relaxed-durability temporary databases for which the durability has been explicitly set, improve the efficiency of minimal logging and avoid ULC flushes to syslogs by creating the sessions tempdb-specific ULC twice the size as the server logical page size:

```
declare @ulc_size int
select @ulc_size = @@maxpagesize * 2
exec sp_configure "session tempdb log cache size", @ulc_size
```

Generally, these scripts configure a ULC sized to contain all changes from one subcommand entirely in-memory, and yield significant improvement in concurrent DML performance.

Changing the ULC size requires you to restart Adaptive Server.

To determine if minimal logging is efficient, view the Transaction Management section of sp_sysmon output.

This example shows efficient minimally logged DML, because Adaptive Server discards most log records generated from subcommands:

```
ML-DMLs ULC Efficiency    per sec      per xact       count  % of total
------------------------  -------  ------------  ----------  ----------
Discarded Sub-Commands    33383.9       11087.8     3071323       100.0
Logged Sub-Commands           0.4           0.1          37         0.0
```

The Transaction Detail section presents a summary of DML commands performed in fully-logged versus minimally-logged mode for the specified sample. In this output, Adaptive Server performed nearly all the inserts in minimally-logged mode on a data-only locked table:

```
Transaction Detail             per sec      per xact     count  % of total
-----------------------   ------------  ------------  -------   ----------
Inserts
  Fully Logged
   APL Heap Table                57.8         173.5       694        0.7 %
   APL Clustered Table            0.0           0.0         0        0.0 %
   Data Only Lock Table           0.7           2.0         8        0.0 %
   Fast Bulk Insert               0.0           0.0         0        0.0 %
  Minimally Logged
   APL Heap Table                 0.0           0.0         0        0.0 %
   APL Clustered Table            0.0           0.0         0        0.0 %
   Data Only Lock Table        7775.8       23327.5     93310       99.3 %
```

The Transaction management section provides details about how Adaptive Server logs subcommands instead of discarding them. The output below shows the events causing ULC flushes to the transaction log, with a break-up of ULC flushes caused by fully-logged and minimally-logged DML. For the minimally-logged DML section, nearly an equal amount of flushes were due to a Full ULC and to the end of a sub-command:

```
ULC Flushes to Xact Log        per sec      per xact     count  % of total
-----------------------   ------------  ------------  -------   ----------
Any Logging Mode DMLs
by End Transaction                0.3           1.0         4       11.1 %
by Change of Database             0.0           0.0         0        0.0 %
by Unpin                          0.0           0.0         0        0.0 %
by Other                          0.0           0.0         0        0.0 %

Fully Logged DMLs
  by Full ULC                     0.2           0.5         2        5.6 %
  by Single Log Record            0.0           0.0         0        0.0 %

Minimally Logged DMLs
  by Full ULC                     1.3           4.0        16       44.4 %
  by Single Log Record            0.0           0.0         0        0.0 %
  by Start of Sub-Command         0.0           0.0         0        0.0 %
  by End of Sub-Command           1.2           3.5        14       38.9 %
-----------------------     ------------  ------------  -------   ----------
 Total ULC Flushes                3.0           9.0        36
```

If the value of the `count` column in the ULC Flushes to Xact Log by Full ULC section for Minimally Logged DMLs is high compared with the number of rows being affected, increase the value for the user log cache or the session tempdb log cache size configuration parameters.

The output below indicates the efficiency of ULC operations for minimally-logged commands: the system incurs a small amount of logging overhead from minimal logging because nearly all the logging activity is entirely contained within the ULC, and very little flushing to syslogs.

```
ML-DMLs ULC Efficiency          per sec      per xact     count  % of total
------------------------   ------------  ------------  -------   ----------
Discarded Sub-Commands           7774.7       23324.0    93296        100.0
Logged Sub-Commands                 1.2           3.5       14          0.0
-------------------------  ------------  ------------  -------
Total ML-DML Sub-Commands        7775.8       23327.5     93310
```

Dumping and loading in-memory databases

Dumping and loading relaxed-durability databases is identical to dumping and loading full-durability databases. During a dump of a relaxed-durability database, Backup Server reads directly from the disk-based database devices, During a load, Backup Server writes directly to these devices by copying over pages from the dump archive. You can improve dumping and loading performance by using striped devices for relaxed-durability databases.

Because in-memory databases do not use disk devices, Backup Server reads pages directly from Adaptive Server shared memory during a dump, and writes it to the archive medium. During a load, Backup Server reads pages from the archive medium (for example, tape) and writes the load directly to the pages of the in-memory storage cache. Because in-memory databases do not use disk I/O to read or write pages on the server, the dump and load performance for in-memory databases is generally superior to that of disk-based databases of the same specifications.

During the dump and load commands, Backup Server opens one CT-library connection per stripe to Adaptive Server, and creates a communication channel to read database pages directly from Adaptive Server shared memory.

Backup Server synchronizes directly with concurrent tasks active in Adaptive Server while reading pages directly from its shared memory. To support load database recovery, Adaptive Server disables the strategies described in "Performance optimization for low-durability databases" on page 45 for the duration of the dump operation. This may reduce the performance of transactional activity during the dump database command.

Tuning for spinlock contention and network connections

In-memory databases do not have as many latency and contention issues as disk-resident databases because they do not use disk I/O. However, in heavy workload situations, in-memory databases may suffer from other spinlock contention and in-memory access issues:

- Cache spinlock contention – may become a bottleneck for in-memory caches under a heavy workload. Consider increasing the number of cache partitions to 64 or more. The additional memory resources required for large number of cache partitions is insignificant and provide improved performance by reducing the cache manager spinlock contention.
- Object manager spinlock contention – if your application frequently accesses a small number of objects, you may observe spinlock contention for metadata structures, which are reported by sp_sysmon.

  Bind descriptors for frequently accessed objects using dbcc tune 'des_bind' so they are never scavenged. Binding the descriptors for even a few commonly used objects may greatly reduce the overall metadata spinlock contention, and improve performance.

Improving contention for lock manager hashtable spinlock ratios

The throughput for in-memory databases may produce contention for the lock manager hashtable spinlock ratios. The table lock hashtable, and the page and row lock hashtable spinlocks may contribute considerably to the contention.

The Lock Management section of sp_sysmon shows the percentage of contention for the spinlocks that govern hash buckets for these hashtables:

```
Lock Management
```

```
---------------

Lock Summary                  per sec     per xact     count     % of total
------------------------  ------------  ------------  --------  ------------
Total Lock Request          285063.3         43.0  17103795           n/a
Avg Lock Contention           1857.3          0.3    111435         0.7 %
Cluster Locks Retained           0.0          0.0         0         0.0 %
Deadlock Percentage              0.1          0.0         8         0.0 %

Lock Detail                   per sec     per xact     count     % of total
------------------------  ------------  ------------  --------  ------------
Table Lock Hashtable
  Lookups                  130160.4          19.6   7809622           n/a
  Avg Chain Length              n/a           n/a   0.00000           n/a
  Spinlock Contention           n/a           n/a       n/a          4.6 %
[...]
Page & Row Lock HashTable
  Lookups                  268968.1          40.6  16138085           n/a
  Avg Chain Length              n/a           n/a   1.03330           n/a
  Spinlock Contention           n/a           n/a       n/a          4.8 %
```

Generally, if this contention is more than approximately 4 percent, consider reducing the ratio of these spinlocks to hash buckets. The default value for the configuration option controlling the number of hash buckets controlled by one spinlock is:

- lock table spinlock ratio with a default value of 20 affects the spinlock contention on the Table Lock Hashtable output.
- lock spinlock ratio with a default of 85 affects the spinlock contention on the Page & Row Lock Hashtable output.

When the spinlock contention is significant, reducing the values for lock table spinlock ratio and lock spinlock ratio may improve run-time performance. The additional memory overhead of more spinlocks controlling fewer hash buckets is not significant. Initially, reduce the configuration parameters by one half their values. In cases of extreme spinlock contention (in excess of 10 percent), reducing the appropriate configuration option to a very small value (say, .3 – 5), may help remove performance bottlenecks from spinlock overheads.

Determining the number of network connections

By default, Adaptive Server uses a single network listener, which can run on any engine. When Adaptive Server receives a connection request, the engine that accepts the connection becomes the network engine for that connection, and when the corresponding task performs network I/O, it must migrate to this engine.

If many client connections take place simultaneously, Adaptive Server may not be able to schedule the listener between the connection requests (unless it yields due to running out of timeslice), and accepts all connections on the same engine. Since any of the corresponding tasks must run on this engine to perform network I/O, this engine becomes a bottleneck. Use the sp_sysmon Network I/O Management section to determine how Adaptive Server distributes the network I/O. This example shows Adaptive Server not distributing the network I/O proportionally: Engine 2 uses more than 88 percent of the network I/O, while other engines use as little as 0 percent:

```
Network I/O Management
----------------------

Total Network I/O Requests       7301.5           1.4      438092
n/a
Network I/Os Delayed              0.0           0.0           0
0.0 %

Total TDS Packets Received      per sec     per xact     count      % of total
--------------------------  ------------  ------------  --------    ------------
Engine 0                          308.8           0.1    18528          7.7 %
Engine 1                          163.6           0.0     9818          4.1 %
Engine 2                         3558.8           0.7   213527         88.3 %
Engine 3                            0.0           0.0        2          0.0 %
Engine 4                            0.0           0.0        0          0.0 %
Engine 5                            0.0           0.0        0          0.0 %
--------------------------  ------------       --------  -------
Total TDS Packets Rec'd          4031.3           0.8   241875

Avg Bytes Rec'd per Packet          n/            n/a      136
n/a
-------------------------------------------------------------------------------



Total TDS Packets Sent          per sec     per xact     count      % of total
--------------------------  ------------  ------------  --------    ------------
Engine 0                          308.8           0.1     18529          7.7 %
```

```
Engine 1                          163.6           0.0      9818          4.1 %
Engine 2                         3558.9           0.7    213531         88.3 %
Engine 3                            0.0           0.0         2          0.0 %
Engine 4                            0.0           0.0         0          0.0 %
Engine 5                            0.0           0.0         0          0.0 %
------------------------  -----------  ------------  --------
Total TDS Packets Sent            4031.3             0.8    241880
```

To resolve unbalanced network I/O usage, use multiple network listeners and bind them to different engines (typically, one listener per engine). To determine how many clients to bind to each network listener, divide the client connections so that each listener accepts approximately the same number of connections. For example, if there are 6 network listeners and 60 clients, connect each group of 10 clients to one listener.

The sp_sysmon output after balancing the network listeners above looks similar to:

```
Network I/O Management
----------------------

Total Network I/O Requests       8666.5         1.3     519991             n/a
n/a
Network I/Os Delayed                0.0         0.0          0          0.0 %

Total TDS Packets Received      per sec     per xact     count      % of total
------------------------  ------------  ------------  --------    ------------
Engine 0                         893.4           0.1     53602         17.8 %
Engine 1                         924.5           0.1     55468         18.5 %
Engine 2                         701.9           0.1     42113         14.0 %
Engine 3                         906.0           0.1     54358         18.1 %
Engine 4                         896.1           0.1     53763         17.9 %
Engine 5                         683.8           0.1     41028         13.7 %
------------------------  ------------       --------  -------
Total TDS Packets Rec'd          5005.5            0.8   300332

Avg Bytes Rec'd per Packet          n/             n/a      136
n/a
-------------------------------------------------------------------------------

Total TDS Packets Sent          per sec     per xact     count      % of total
------------------------  ------------  ------------  --------    ------------
Engine 0                         893.3           0.1     53595         17.8 %
Engine 1                         924.5           0.1     55467         18.5 %
Engine 2                         701.9           0.1     42113         14.0 %
Engine 3                         905.9           0.1     54355         18.1 %
Engine 4                         896.1           0.1     53763         17.9 %
```

```
Engine 5                          683.8          0.1     41026          13.7 %
------------------------   -----------  -----------  --------
Total TDS Packets Sent             4031.3          0.8    241880
```

Unbalanced network listeners are not specific to in-memory databases, but can also occur in disk-resident databases. However, because they do not use disk I/O, in-memory-resident databases typically have greater throughput than disk-resident databases. The increased throughput and non-existent disk I/O latency results in in-memory databases performing more work than disk-resident databases, including more network I/O, which could increase the severity of the bottleneck due to unbalanced network listener loads.

Index

N

O

P

R

S

T

U