CHAPTER 4

Tuning the Database with Initialization Parameters (DBA)

The init.ora file (and spfile) determines many Oracle operating system environment attributes, such as memory allocated for data, memory allocated for statements, resources allocated for I/O, and other crucial performance-related parameters. Each version of Oracle continues to add to the total number of initialization parameters. In Oracle 10*g* Release 2 there are now 1381 (257 documented and 1124 hidden) different initialization parameters (these numbers vary slightly on different versions of Oracle and platforms). As you might expect, an entire book could be written on how to set and tune each parameter; this book focuses on the key parameters that affect database performance. The key to an optimized Oracle database is often the architecture of the system and the parameters that set the environment for the database. Setting four key initialization parameters (SGA_MAX_SIZE, PGA_AGGREGATE_TARGET, DB_CACHE_SIZE, and SHARED_POOL_SIZE) can be the difference between sub-second queries and queries that take several minutes. There is also a new SGA_TARGET parameter that can replace some of the key parameter that can be set as well that is covered in this chapter. This chapter will focus on the crucial initialization parameters but also list the top 25 initialization parameters near the end of the chapter. The chapter concludes with a look at typical server configurations for various database sizes.

This chapter contains the following tips and techniques designed to achieve the greatest performance gain with the least effort by focusing on the parameters that yield the biggest impact:

- Crucial initialization parameters in Oracle
- Modifying the initialization parameter file without a restart
- Viewing the initialization parameters via Enterprise Manager
- Tuning DB_CACHE_SIZE and monitoring hit ratios
- Tuning the SHARED_POOL_SIZE
- Checking library cache and dictionary cache
- Querying the X$KSMSP table to get another picture of SHARED_POOL_SIZE
- Using multiple buffer pools
- Tuning the PGA_AGGREGATE_TARGET
- User, session, and system memory use
- Cost- vs. rule-based optimization
- The top 25 performance-related initialization parameters to consider
- Undocumented initialization parameters (more in Appendix A)
- Typical server setups with different size databases

Identifying Crucial Initialization Parameters

While tuning specific queries alone can lead to performance gains, the system will still be slow if the parameters for the initialization file are not set correctly because the initialization file plays such an integral role in the overall performance of an Oracle database. While you can spend

time setting all the initialization parameters, there are just four main parameters that need to be set correctly to realize significant performance gains:

- SGA_MAX_SIZE
- PGA_AGGREGATE_TARGET
- DB_CACHE_SIZE
- SHARED_POOL_SIZE

TIP
The key initialization parameters in Oracle are SGA_MAX_SIZE, PGA_AGGREGATE_TARGET, DB_CACHE_SIZE, and SHARED_POOL_SIZE.

There is also a new parameter, SGA_TARGET, which can be set so that Oracle manages the shared memory on your system (Automatic Shared Memory Management); Metalink Note 295626.1 describes this in detail. While this is a new parameter, the Oracle Application Development team recommends this for 10*g* (I included these recommendations at the end of this chapter). I would like to see this mature a bit more before I hand the "keys to the car" to Oracle, but I like the approach to simplicity, especially for beginners. The following query can be used to find the current settings of the key initialization parameters on your database (if SGA_TARGET is set to a non-zero value, then some of these parameters will be set to zero):


```
Col name for a25
Col value for a50

select   name, value
from     v$parameter
where    name in ('sga_max_size', 'pga_aggregate_target',
                  'db_cache_size', 'shared_pool_size');

NAME                 VALUE
-------------------- --------------------
shared_pool_size     50331648
sga_max_size         135338868
db_cache_size        25165824
pga_aggregate_target 25165824
```

Changing the Initialization Parameters Without a Restart

With each version of Oracle, more and more parameters can be altered without needing to restart the database. This has greatly reduced the need for scheduled downtime to implement system tuning changes. The next example shows changing the SHARED_POOL_SIZE to 128M while the database is running:

```
SQL> ALTER SYSTEM SET SHARED_POOL_SIZE = 128M;
```

In addition to being able to dynamically change parameters, Oracle 10*g* provides for the use of a SPFILE to persistently store dynamic changes to the instance parameters. Prior to Oracle 9*i,* any dynamic changes were lost when the database was restarted unless the parameters were added to the initialization parameter file manually. As of Oracle 9*i* and continuing into Oracle 10*g* Release 2, dynamic changes can be stored in a server parameter file (spfile). The default order of precedence when an instance is started is to read parameter files in the following order:

1. spfile<SID>.ora
2. spfile.ora
3. init<SID>.ora

Parameters can be dynamically modified at a system-wide or session-specific scope. In addition, parameters can be changed in memory only or persist across restarts via an SPFILE.

TIP
If you can't figure out why your system isn't using the value in your init.ora file, you probably have an spfile overriding it. And, don't forget, you can also use a hint to override parameters at the query level in 10gR2.

Finally, in a Real Application Cluster environment, parameters can be changed for a single node or for all nodes in a cluster.

There are two key fields in the V$PARAMETER view:

- **ISSES_MODIFIABLE** Indicates if a user with the ALTER SESSION privilege can modify this initialization parameter for their session.
- **ISSYS_MODIFIABLE** Indicates if someone with ALTER SYSTEM privilege can modify this particular parameter.

The following query illustrates a list of initialization parameters that can be set without shutting down and restarting the database. This query displays the initialization parameters that can be modified with an ALTER SYSTEM or ALTER SESSION command (partial result displayed):

```
select    name, value, isdefault, isses_modifiable, issys_modifiable
from      v$parameter
where     issys_modifiable <> 'FALSE'
or        isses_modifiable <> 'FALSE'
order by  name;
```

The result of the query is all of the initialization parameters that may be modified:

```
NAME                      VALUE                          ISDEFAULT ISSES ISSYS_MOD
------------------------- ------------------------------ --------- ----- ---------
aq_tm_processes           0                              TRUE      FALSE IMMEDIATE
archive_lag_target        0                              TRUE      FALSE IMMEDIATE
background_dump_dest      C:\oracle\admin\orcl9ir2\bdump FALSE     FALSE IMMEDIATE
backup_tape_io_slaves     FALSE                          TRUE      FALSE DEFERRED
```

Be careful granting the ALTER SESSION privilege to users, as knowledgeable developers can set individual parameters that positively affect their session at the expense of others on the system.

TIP

Changing initialization parameters dynamically is a powerful feature for both developers and DBAs. Consequently, a user with the ALTER SESSION privilege is capable of irresponsibly allocating 100M+ for the SORT_AREA_SIZE for a given session, if it is not restricted.

Viewing the Initialization Parameters with Enterprise Manager

You can also use Enterprise Manager to view the initialization parameter settings in the Configuration screen under the Instance option. The section of Enterprise Manager displayed in Figure 4-1 shows the initialization parameters. It shows the current settings for the parameters and also shows if the parameters can be modified (dynamic=Y) without shutting down the database. Oracle Enterprise Manager is covered in detail in Chapter 5.

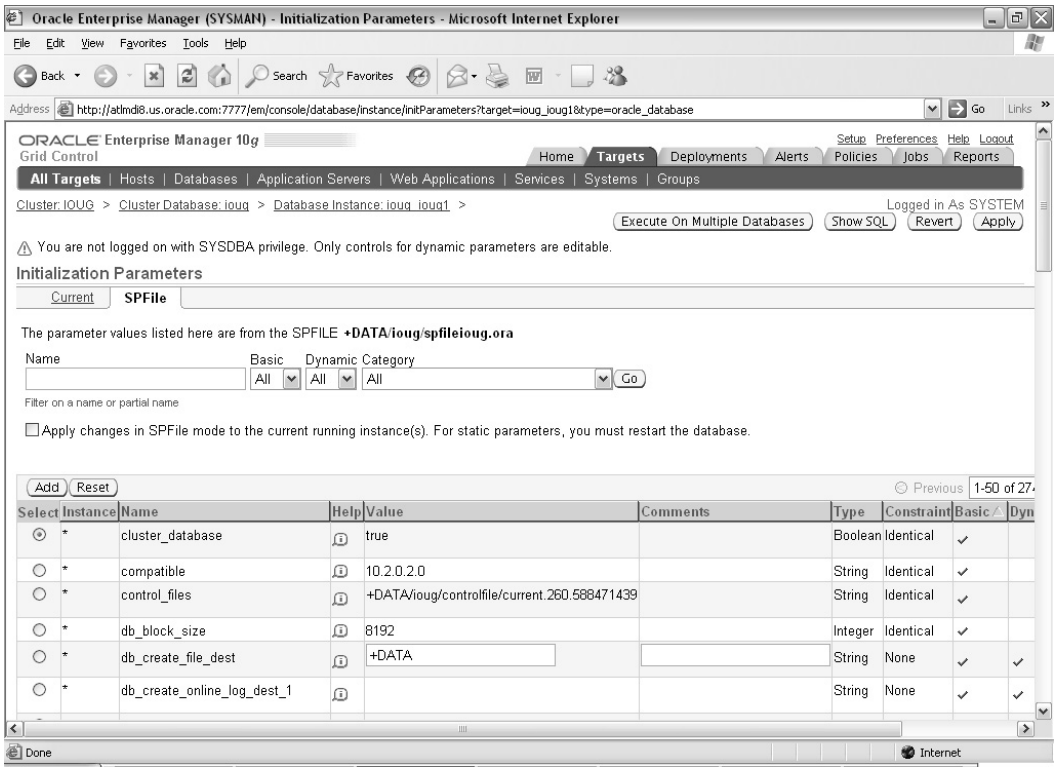

FIGURE 4-1. *Enterprise Manager—initialization parameters in the SPFILE*

Increasing Performance by Tuning the DB_CACHE_SIZE

Long-time users of Oracle and readers of prior editions of this book will notice that some familiar parameters have not been mentioned. This is because parameters such as DB_BLOCK_BUFFERS have been deprecated (a parameter _DB_BLOCK_BUFFERS is set behind the scenes for backward compatibility). While many of the familiar parameters from prior version of Oracle are still valid, using them disables many Oracle 10*g* Release 2 features, including automatic cache memory management. This chapter focuses on the Oracle 10*g* Release 2 parameters for tuning your system.

DB_CACHE_SIZE is the first parameter to look at in the initialization parameter file because it's the most crucial parameter in Oracle. If the DB_CACHE_SIZE is set too low, Oracle won't have enough memory to operate efficiently and the system may run poorly, no matter what else you do to it. If DB_CACHE_SIZE is too high, your system may begin to swap and may come to a halt. DB_CACHE_SIZE makes up the area of the SGA that is used for storing and processing data in memory. As users request information, data is put into memory. If the DB_CACHE_SIZE parameter is set too low, then the least recently used data will be flushed from memory. If the flushed data is recalled with a query, it must be reread from disk (causing I/O and CPU resources to be used).

Retrieving data from memory can be over 10,000 times faster than disk (depending on the speed of memory and disk devices). Even if you take into consideration disk caching (memory on disk) and Oracle inefficiencies, retrieving data from memory is still about 100 times faster than reading data from disk. Therefore, the higher the percentage of frequency that records are found in memory (without being retrieved from disk), the faster the overall system performance (usually at least 100 times faster for well-tuned queries). Having enough memory allocated to store data in memory depends on the value used for DB_CACHE_SIZE (or for SGA_TARGET if used).

TIP

Retrieving data from physical memory is generally substantially faster than retrieving it from disk, so make sure that the SGA is large enough. One Oracle study showed Oracle memory access as averaging about 100 times faster than disk access. However, this takes into account disk caching advances, which you may or may not have on your system. The same study also showed an individual case where Oracle memory access was well over 10,000 times faster than disk (which was hard for me to believe), but it shows how important it is to measure this on your own unique system.

DB_CACHE_SIZE is the key parameter to use when tuning the data cache hit ratio. The data cache hit ratio is the percentage of the data block accesses that occur without requiring a physical read from disk. While there are several situations that can artificially inflate or deflate the data cache hit ratio, this ratio is a key indicator of system efficiency. The following query can be used to view the data cache hit ratio:

```
column phys           format 999,999,999    heading 'Physical Reads'
column gets           format 999,999,999    heading ' DB Block Gets'
column con_gets       format 999,999,999    heading 'Consistent Gets'
column hitratio format 999.99 heading ' Hit Ratio '
 select   sum(decode(name,'physical reads',value,0)) phys,
          sum(decode(name,'db block gets',value,0)) gets,
```

```
          sum(decode(name,'consistent gets', value,0)) con_gets,
         (1 - (sum(decode(name,'physical reads',value,0)) /
         (sum(decode(name,'db block gets',value,0)) +
          sum(decode(name,'consistent gets',value,0))))) * 100 hitratio
from      v$sysstat;

Physical Reads  DB Block Gets Consistent Gets  Hit Ratio
-------------- -------------- --------------- -----------
     1,671          39,561          71,142       98.49
```

While there are exceptions for every application, a data cache hit ratio of 95 percent or greater should be achievable for a well-tuned transactional application with the appropriate amount of memory. Because there is such a performance difference between some disk devices and memory access, improving the data cache hit ratio from 90 to 95 percent can nearly double system performance when reading disk devices that are extremely slow. Improving the cache hit ratio from 90 to 98 percent could yield nearly a 500 percent improvement where disks are extremely slow and under the right (or should I say wrong) architectural setup.

Poor joins and poor indexing can also yield very high hit ratios due to reading many index blocks, so make sure that your hit ratio isn't high for a reason other than a well-tuned system. An unusually high hit ratio may indicate the introduction of code that is poorly indexed or includes join issues.

TIP

Hit ratios are useful to experienced DBAs but can be misleading for inexperienced DBAs. The best use of hit ratios is still to compare over time to help alert you to a substantial change to a system on a given day. While there are some that have deprecated hit ratios, they are usually tool vendors who don't see the value of tracking hit ratios over time, since their tools are point-in-time or reactive-based tuning solutions. Hit ratios should never be your only tool, but they should definitely be one of many proactive tools in your arsenal.

Oracle continues to downplay the importance of hit ratios by reducing the discussions on hit ratio tuning. Oracle is beginning to focus on analyzing system performance in terms of work done (CPU or service time) versus time spent waiting for work (wait time). Areas where hit ratios are still the primary tuning method are library cache and dictionary cache. See Chapter 14 on STATSPACK for more information on balancing the entire tuning arsenal including hit ratios.

Using V$DB_CACHE_ADVICE in tuning DB_CACHE_SIZE

V$DB_CACHE_ADVICE is a view introduced in Oracle 9*i* to assist in tuning DB_CACHE_SIZE. The view can be queried directly, and the data in the view is used by the Oracle kernel (or database engine) to make automatic cache management decisions. Here is an Oracle 10*g* Release 2 query (note that Oracle 9*i* Release 1 does not have the column size_factor) to view the effect of changing DB_CACHE_SIZE on the data cache hit ratio:

```
select name, size_for_estimate, size_factor, estd_physical_read_factor
from   v$db_cache_advice;
```

```
NAME                 SIZE_FOR_ESTIMATE SIZE_FACTOR ESTD_PHYSICAL_READ_FACTOR
-------------------- ----------------- ----------- -------------------------
DEFAULT                              4       .1667                    1.8136
DEFAULT                              8       .3333                    1.0169
DEFAULT                             12          .5                    1.0085
DEFAULT                             16       .6667                         1
DEFAULT                             20       .8333                         1
DEFAULT                             24           1                         1
```

Reading these results, we see the following:

- The current cache size is 24MB (size_factor = 1).
- We can decrease the cache size to be 16MB and maintain the current cache hit ratio, since the physical_read_factor remains at 1 up to a decrease to 16MB.

While this view provides an estimate of the effect of changing the cache size on the cache hit ratio, any changes should be tested to validate that the results are as forecasted. Oracle Enterprise Manager provides a graphical view of the data in V$DB_CACHE_ADVICE.

Keeping the Hit Ratio for the Data Cache Above 95 Percent

The hit ratio for the data cache should generally be above 95 percent for transactional systems. But, the best use for a hit ratio is to study your system over time to see major changes that should warrant further investigation. Usually, if your hit ratio is below 95 percent, you may need to increase the value of DB_CACHE_SIZE. In some instances, you can increase performance substantially by increasing the hit ratio from 95 to 98 percent—especially if the last 5 percent of the hits going to disk are the main lag on the system.

Monitoring the V$SQLAREA View to Find Bad Queries

Although hit ratios below 95 percent are usually a sign that your DB_CACHE_SIZE is set too low or that you have poor indexing, distortion of the hit ratio numbers is possible and needs to be taken into account while tuning. Hit ratio distortion and non-DB_CACHE_SIZE issues include the following:

- Recursive calls
- Missing or suppressed indexes
- Data sitting in memory
- Rollback segments
- Multiple logical reads
- Physical reads causing the system to use CPU

To avoid being misled, locate bad queries by monitoring the V$SQLAREA view. Once you isolate the queries that are causing performance hits, tune the queries or modify how the information is stored to solve the problem. Using the Performance page of Enterprise Manager

Grid Control, a DBA can generate the TopSQL for their system. The TopSQL section of Enterprise Manager Grid Control (Figure 4-2) displays a list of the worst SQL statements in the current cache based on Activity and also the Top Sessions by Activity. The DBA can then click on the problem SQL to begin the process of analyzing and tuning the problem SQL statement. Chapter 5 discusses the benefits of Oracle's Enterprise Manager in detail and how to tune the SQL statements using Enterprise Manager Grid Control.

TIP
In Oracle 10g Release 2, use the Enterprise Manager Grid Control to find problem queries.

Hit Ratios Are Not Always Accurate

If you are utilizing hit ratios to measure performance, note that within Performance Manager, during peak times the number of disk reads is larger than the number of in-memory reads and thus the negative hit ratio is being computed in terms of the deltas between physical reads and logical reads.

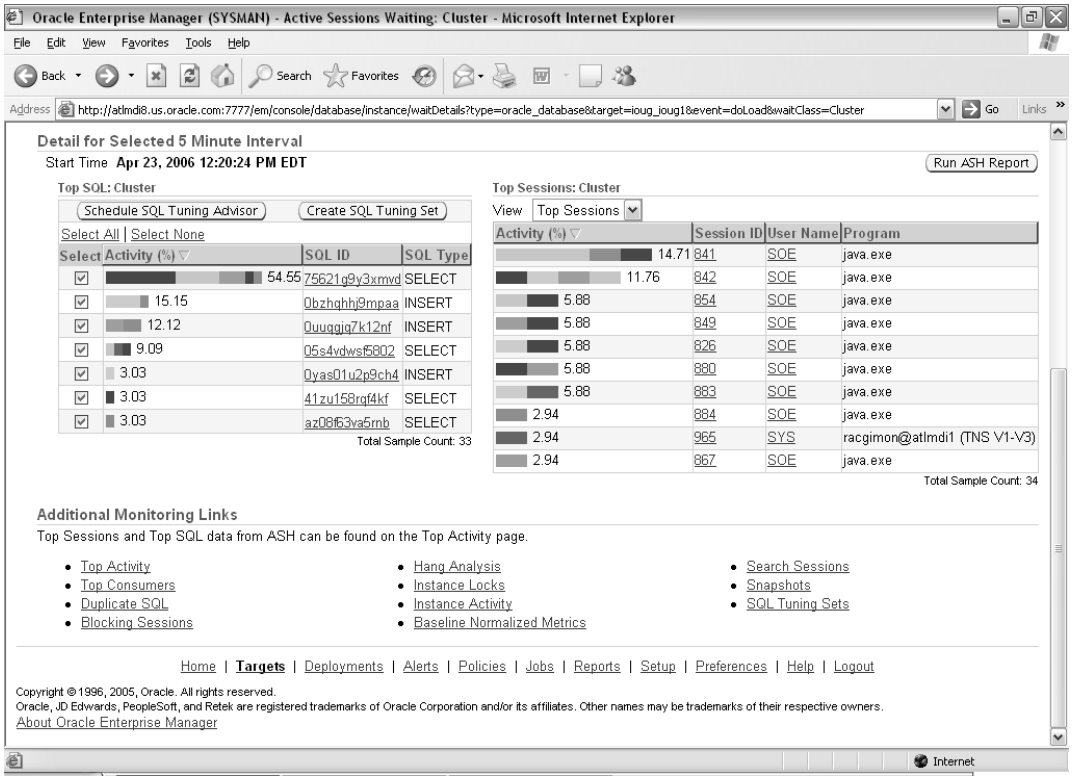

FIGURE 4-2. *Use Oracle's Enterprise Manager Grid Control to find problem queries.*

Bad Hit Ratios Can Occur When an Index Is Suppressed

Consider the following query where the CUSTOMER table is indexed on the unique custno column. It is *not* optimal to have this index suppressed by using the NVL, because it results in a poor hit ratio.

```
select    custno, name
from      customer
where     nvl(custno,0)  = 5789;

Tries (Logical Reads) = 105      Physical = 100
% hit ratio    =     (1   -     Physical/Tries) x 100
% hit ratio    =     (1 - 100/105) x 100%
% hit ratio    =     4.8%  (A very low/bad hit ratio)
```

If you are looking at this in Enterprise Manager, there is an index missing on a query that is being executed at the current time. Focus on the query that is causing this problem and fix the query. The query can be found by accessing the V$SQLAREA view as shown in Chapter 8.

TIP
A low hit ratio for a query is an indication of a missing or suppressed index.

Getting Good Hit Ratios with Well-Indexed Queries

Consider the following query, where the customer table is indexed on the unique custno column. In this situation, it is optimal to utilize the custno index because it results in an excellent hit ratio.

```
select    custno, name
from      customer
where     custno = 5789;

Tries (Logical Reads) = 105 Physical = 1
% hit ratio    =     (1 - Physical/Tries) x 100
% hit ratio    =     (1 - 1/105) x 100%
% hit ratio    =     99% (A very high/usually good hit ratio)
```

If you are looking at this in the Enterprise Manager, there is usually an index on the query that is being executed.

Bad Queries Executing a Second Time Can Result in Good Hit Ratios

When a full table scan is completed for the second time and the data is still in memory, you may see a good hit ratio even though the system is trying to run a bad query.

```
Tries (Logical Reads) = 105     Physical = 1
% hit ratio    =     (1 - Physical/Tries) x 100
% hit ratio    =     (1 - 1/105) x 100%
% hit ratio    =     99% (A very high/usually good hit ratio)
```

If you are looking at this in the Enterprise Manager, it appears that there is an index on the query being executed when in fact the data is in memory from the last time it was executed. The result is that you are "hogging up" a lot of memory even though it appears that an indexed search is being done.

Color profile: Generic CMYK printer profile
Composite Default screen

TIP
Bad (slow) queries show in V$SQLAREA view with poor hit ratios the first time they are executed. Make sure your tune them at that time. The second time that they execute, they may not show a poor hit ratio.

Other Hit Ratio Distortions

There are several other hit distortions to consider:

- **Oracle Forms distortion** Systems that use Oracle Forms (screens) frequently might use the same information over and over. This reuse by some of the users of the system will drive up the hit ratio. Other users on the system may not be experiencing hit ratios that are as good as the Forms users, yet the overall system hit ratio may look very good. The DBA must take into consideration that the Forms users can be boosting the hit ratio to an artificially high level.
- **Rollback segment distortion** Because the header block of the rollback segment is usually cached, the activity to the rollback segment gives a falsely high hit ratio impact when truly there is no significant impact on the hit ratio.
- **Index distortion** An index range scan results in multiple logical reads on a very small number of blocks. Hit ratios as high as 86 percent can be recorded when none of the blocks are cached prior to the query executing. Make sure you monitor the hit ratio of individual poorly tuned queries in addition to monitoring the big picture (overall hit ratio).
- **I/O distortion** Physical reads that appear to be causing heavy disk I/O may be actually causing you to be CPU bound. In tests, the same amount of CPU was used for 89 logical reads as it was to process 11 physical reads. The result is that the physical reads are CPU costly because of BUFFER MANAGEMENT. Fix the queries causing the disk I/O problems and you will usually free up a large amount of CPU as well. Performance degradation can be exponentially downward spiraling, but the good news is that when you begin to fix your system, it is often an exponentially upward-spiraling event. It's probably the main reason why some people live to tune; tuning can be exhilarating.

Setting DB_BLOCK_SIZE to Reflect the Size of Your Data Reads

The DB_BLOCK_SIZE is the size of the default data block size when the database is created. With Oracle 10*g* Release 2, each tablespace can have a different block size, thus making block size selection a *less* critical selection before the database is created. That said, a separate cache memory allocation must be made for each different database block size. But, it is still very important to choose wisely. While you can have different block size tablespaces, this is not truly a performance feature, as the non-default buffer caches are not optimized for performance. So, you still want to put the bulk of your data in the default buffer cache. The database must be rebuilt if you want to increase the DB_BLOCK_SIZE.

The data block cache for the default block size is set using the DB_CACHE_SIZE initialization parameter. Cache is allocated for other database block sizes by using the DB_*n*K_CACHE_SIZE, where *n* is the block size in KB. The larger the DB_BLOCK_SIZE, the more that can fit inside a

P:\010Comp\OracleT&T\305-9\ch04.vp
Friday, May 18, 2007 11:24:21 AM

single block and the more efficient large amounts of data can be retrieved. A small DB_BLOCK_SIZE actually lets you retrieve single records faster and saves space in memory. In addition, a smaller block size can improve transactional concurrency and reduce log file generation rates. As a rule of thumb, a data warehouse should use the maximum block size available for your platform (either 16KB or 32KB), while a transaction processing system should use an 8KB block size. Rarely is a block size smaller than 8KB beneficial. If you have an extremely high transaction rate system or very limited system memory, you might consider a block size smaller than 8KB.

Full table scans are limited to the maximum I/O of the box (usually 64K, but as high as 1M on many systems). You can up the amount of data read into memory in a single I/O by increasing DB_BLOCK_SIZE to 8K or 16K. You can also increase the DB_FILE_MULTIBLOCK_READ_COUNT to the value of (max I/O size)/DB_BLOCK_SIZE.

Environments that run a lot of single queries to retrieve data could use a smaller block size, but "hot spots" in those systems will still benefit from using a larger block size. Sites that need to read large amounts of data in a single I/O read should increase the DB_FILE_ MULTIBLOCK_READ_ COUNT. Setting the DB_FILE_MULTIBLOCK_READ_COUNT higher is especially important for data warehouses that retrieve lots of records. If the use of DB_FILE_MULTIBLOCK_READ_COUNT starts to cause many full table scans (since the optimizer now decides it can perform full table scans much faster and decides to do more of them) then set OPTIMIZER_INDEX_COST_ADJ between 1 and 10 (I usually use 10) to force index use more frequently.

TIP

The database must be rebuilt if you increase the DB_BLOCK_SIZE. Increasing the DB_FILE_MULTIBLOCK_READ_COUNT will allow more block reads in a single I/O, giving a benefit similar to a larger block size.

The general rule of thumb is to start with an SGA_MAX_SIZE parameter at 25 percent of the size allocated to your main memory. A large number of users (300+) or a small amount of available memory may force you to make this 15–20 percent of physical memory. A small number of users (less than 100) or a large amount of physical memory may allow you to make this 30–50 percent of physical memory. If you set the SGA_MAX_SIZE less than 128M, then the _ksm_granule_size will be 4M. If the SGA_MAX_SIZE is greater or equal to 128M, then the _ksm_granule_size will be 16M. This granule size will determine the multiples for other initialization parameters. A granule size of 4M means that certain initialization parameters will be rounded up to the nearest 4M. Therefore, if I set SGA_MAX_SIZE to 64M and DB_CACHE_SIZE to 9M, then the DB_CACHE_SIZE will be rounded to 12M (since the granule size is 4M). If I set SGA_MAX_SIZE to 200M and DB_CACHE_SIZE to 9M, then the DB_CACHE_SIZE will be rounded to 16M (since the granule size is 16M).

TIP

The SGA_MAX_SIZE determines the granule size for other parameters. An SGA_MAX_SIZE<128M means a 4M granule, whereas an SGA_MAX_SIZE>=128M means a 16M granule size.

Tuning the SHARED_POOL_SIZE for Optimal Performance

Sizing the SHARED_POOL_SIZE correctly will make sharing SQL statements that are identical possible. Getting the statement parsed is priority #1. If the query never makes it into memory, it can never request the data to be accessed; that's where the SHARED_POOL_SIZE comes in. SHARED_POOL_SIZE specifies the memory allocated in the SGA for data dictionary caching and shared SQL statements.

The data dictionary cache is very important because that's where the data dictionary components are buffered. Oracle references the data dictionary several times when a SQL statement is processed. Therefore, the more information (database and application schema and structure) that's stored in memory, the less information that'll have to be retrieved from disk. While the dictionary cache is part of the shared pool, Oracle also caches SQL statements and their corresponding execution plans in the library cache portion of the shared pool (see next section for how the shared SQL area works).

The data dictionary cache portion of the shared pool operates in a manner similar to the DB_CACHE_SIZE when caching information. For the best performance, it would be great if the entire Oracle data dictionary could be cached in memory. Unfortunately, this usually is not feasible, so Oracle uses a least recently used algorithm for deciding what gets to stay in the cache.

Using Stored Procedures for Optimal Use of the Shared SQL Area

Each time a SQL statement is executed, the statement is searched for in the shared SQL area and, if found, used for execution. This saves parsing time and improves overall performance. Therefore, to ensure optimal use of the shared SQL area, use stored procedures as much as possible, since the SQL parsed is exactly the same every time and therefore shared. However, keep in mind the only time the SQL statement being executed can use a statement already in the shared SQL area is if the statements are identical (meaning they have the same content exactly—the same case, the same number of spaces, etc.). If the statements are not identical, the new statement will be parsed, executed, and placed in the shared SQL area (exceptions to this are possible when the initialization parameter CURSOR_SHARING has been set to SIMILAR or FORCE).

In the following example, the statements are identical in execution, but the word *from* causes Oracle to treat the two statements as if they were different, thus *not* reusing the original cursor that was located in the shared SQL area:

```
SQL>     select name, customer from customer_information;
SQL>     select name, customer FROM customer_information;
```

TIP

SQL must be written exactly the same to be reused. Case differences and any other differences will cause a reparse of the statement.

In the following example, we are using different values for ENAME, which is causing multiple statements to be parsed.

```
declare
      temp VARCHAR2(10);
```


```
begin
      select ename into temp
      from   rich
      where  ename = 'SMITH';
      select ename into temp
      from   rich
      where  ename = 'JONES';
end;
```

A query of V$SQLAREA shows that two statements were parsed even though they were very close to the same thing. Note, however, that PL/SQL converted each SQL statement to uppercase *and* trimmed spaces *and* carriage returns (which is a benefit of using PL/SQL).

```
select sql_text
from   v$sqlarea
where  sql_text like 'SELECT ENAME%';

SQL_TEXT
------------------------------------------------
SELECT ENAME   FROM RICH  WHERE ENAME = 'JONES'
SELECT ENAME   FROM RICH  WHERE ENAME = 'SMITH'
```

In the following example, we see a problem with third-party applications that do not use bind variables (they do this to keep the code "vanilla" or capable of working on many different databases without modification). The problem with this code is that the developer has created many statements that fill the shared pool and these statements can't be shared (since they're slightly different). We can build a smaller shared pool so that there is less room for cached cursors and thus fewer cursors to search through to find a match (this is the band-aid inexperienced DBAs use). If the following is your output from v$sqlarea, you may benefit from lowering the SHARED_POOL _SIZE, but using CURSOR_SHARING is a better choice.

```
SQL_TEXT
------------------------------------------------
select empno from rich778 where empno =451572
select empno from rich778 where empno =451573
select empno from rich778 where empno =451574
select empno from rich778 where empno =451575
select empno from rich778 where empno =451576
etc. . .
```

Set CURSOR_SHARING=FORCE and the query to V$SQLAREA will change to the one listed next, because Oracle builds a statement internally that can be shared by all of the preceding statements. Now the shared pool is not inundated with all of these statements; only one simple statement that can be shared by all of them:

```
SQL_TEXT
------------------------------------------------
select empno from rich778 where empno =:SYS_B_0
```

Setting the SHARED_POOL_SIZE High Enough to Fully Use the DB_CACHE_SIZE

If the SHARED_POOL_SIZE is set too low, then you will not get the full advantage of your DB_CACHE_SIZE (since statements that can't be parsed can't be executed). The queries that can be performed against the Oracle V$ views to determine the data dictionary cache hit ratio and the shared SQL statement usage are listed in the sections that follow. These will help you determine if increasing the SHARED_POOL_SIZE will improve performance.

The SHARED_POOL_SIZE parameter is specified in bytes. The default value for the SHARED_POOL_SIZE parameter varies per system but is generally lower than necessary for large production applications.

Keeping the Data Dictionary Cache Hit Ratio at or above 95 Percent

The data dictionary cache is a key area to tune because the dictionary is accessed so frequently, especially by the internals of Oracle. At startup, the data dictionary cache contains no data. But as more data is read into cache, the likelihood of cache misses decreases. For this reason, monitoring the data dictionary cache should be done only after the system has been up for a while and stabilized. If the dictionary cache hit ratio is below 95 percent, then you'll probably need to increase the size of the SHARED_POOL_SIZE parameter in the initialization parameter file. Implementing Locally Managed Tablespaces (LMT) can also help your dictionary cache (see Metalink Note 166474.1, "Can We Tune the Row Cache!") However, keep in mind that the shared pool also includes the library cache (SQL statements) and Oracle decides how much the distribution will be for the library cache versus the row cache.

Use the following query against the Oracle V$ view to determine the data dictionary cache hit ratio:

```
select    ((1 - (Sum(GetMisses) / (Sum(Gets) + Sum(GetMisses)))) * 100) "Hit Rate"
from      V$RowCache
where     Gets + GetMisses <> 0;

 Hit Rate
---------
91.747126
```

TIP

Measure hit ratios for the row cache (data dictionary cache) of the shared pool with the V$ROWCACHE view. A hit ratio of over 95 percent should be achieved. However, when the database is initially started, hit ratios will be around 85 percent.

Using Individual Row Cache Parameters to Diagnose Shared Pool Use To diagnose a problem with the shared pool or the overuse of the shared pool, use a modified query to the V$ROWCACHE view. This will show how each individual parameter makes up the data dictionary cache, also referred to as the row cache.

```
column parameter      format a20          heading 'Data Dictionary Area'
column gets           format 999,999,999  heading 'Total|Requests'
column getmisses      format 999,999,999  heading 'Misses'
```


```
column modifications  format 999,999      heading 'Mods'
column flushes        format 999,999      heading 'Flushes'
column getmiss_ratio  format 9.99         heading 'Miss|Ratio'
set pagesize 50
ttitle 'Shared Pool Row Cache Usage'

select  parameter, gets, getmisses, modifications, flushes,
       (getmisses / decode(gets,0,1,gets)) getmiss_ratio,
       (case when (getmisses / decode(gets,0,1,gets)) > .1 then '*' else ' ' end) " "
from    v$rowcache
where   Gets + GetMisses <> 0;

Tue Aug 27                                                          page    1
                         Shared Pool Row Cache Usage
                           Total                                      Miss
Data Dictionary Area     Requests       Misses     Mods  Flushes Ratio
-------------------- ------------ ------------ -------- -------- ----- -
dc_segments                   637          184        0        0   .29 *
dc_tablespaces                 18            3        0        0   .17 *
dc_users                      126           25        0        0   .20 *
dc_rollback_segments          235           21       31       30   .09
dc_objects                    728          167       55        0   .23 *
dc_global_oids                 16            6        0        0   .38 *
dc_object_ids                 672          164       55        0   .24 *
dc_sequences                    1            1        1        1  1.00 *
dc_usernames                  193           10        0        0   .05
dc_histogram_defs              24           24        0        0  1.00 *
dc_profiles                     1            1        0        0  1.00 *
dc_user_grants                 24           15        0        0   .63 *
```

This query places an asterisk (*) for any query that has misses greater than 10 percent. It does this by using the CASE expression to limit the miss ratio to the tenth digit, and then analyzes that digit for any value greater than 0 (which would indicate a hit ratio of 10 percent or higher). A 0.1 miss or higher returns an *. Explanations of each of the columns are listed in the next section.

Keeping the Library Cache Reload Ratio at 0 and the Hit Ratio Above 95 Percent

For optimal performance, you'll want to keep the library cache reload ratio [sum(reloads) / sum(pins)] at zero and the library cache hit ratio above 95 percent. If the reload ratio is not zero, then there are statements that are being "aged out" that are later needed and brought back into memory. If the reload ratio is zero (0), that means items in the library cache were never aged or invalidated. If the reload ratio is above 1 percent, the SHARED_POOL_SIZE parameter should probably be increased. Likewise, if the library cache hit ratio comes in below 95 percent, then the SHARED_POOL_SIZE parameter may need to be increased. Also, if you are using ASMM, the SGA_TARGET includes both auto-tuned and manual parameters. When you decide to raise a parameter specifically (such as SHARED_POOL_SIZE), it will influence the auto-tuned part. (Other parameters will be affected; see Metalink Note 295626.1, "How to Use Automatic Shared Memory Management (ASMM) in Oracle 10*g*.")

There are a couple of ways to monitor the library cache. The first method is to execute the STATSPACK report (STATSPACK is covered in detail in Chapter 14). The second is to use the

Color profile: Generic CMYK printer profile
Composite Default screen

V$LIBRARYCACHE view. The following query uses the V$LIBRARYCACHE view to examine the reload ratio in the library cache:

```
select    Sum(Pins) "Hits",
          Sum(Reloads) "Misses",
          ((Sum(Reloads) / Sum(Pins)) * 100)"Reload %"
from      V$LibraryCache;

Hits         Misses   Reload %
1969             50   0.253936
```

The next query uses the V$LIBRARYCACHE view to examine the library cache's hit ratio in detail:

```
select    Sum(Pins) "Hits",
          Sum(Reloads) "Misses",
          Sum(Pins) / (Sum(Pins) + Sum(Reloads)) "Hit Ratio"
from      V$LibraryCache;

HITS       MISSES     HIT RATIO
1989            5     .99749248
```

This hit ratio is excellent (over 99 percent) and does not require any increase in the SHARED_POOL_SIZE parameter.

Using Individual Library Cache Parameters to Diagnose Shared Pool Use Using a modified query on the same table, we can see how each individual parameter makes up the library cache. This may help diagnose a problem or show overuse of the shared pool.

```
set numwidth 3
set space 2
set newpage 0
set pagesize 58
set linesize 80
set tab off
set echo off
ttitle 'Shared Pool Library Cache Usage'
column namespace   format a20           heading 'Entity'
column pins        format 999,999,999   heading 'Executions'
column pinhits     format 999,999,999   heading 'Hits'
column pinhitratio format 9.99          heading 'Hit|Ratio'
column reloads     format 999,999       heading 'Reloads'
column reloadratio format .9999         heading 'Reload|Ratio'
spool cache_lib.lis
select   namespace, pins, pinhits, pinhitratio, reloads, reloads
         /decode(pins,0,1,pins) reloadratio
from     v$librarycache;

Sun Mar 19                                                  page    1
                     Shared Pool Library Cache Usage
                                          Hit                  Reload
```


P:\010Comp\OracleT&T\305-9\ch04.vp
Friday, May 18, 2007 11:25:59 AM


```
Entity       Executions       Hits        Ratio     Reloads   Ratio

SQL AREA      1,276,366      1,275,672    1.00           2   .0000
TABLE/PROC      539,431        539,187    1.00           5   .0000
BODY                  0              0    1.00           0   .0000
TRIGGER               0              0    1.00           0   .0000
INDEX                21              0     .00           0   .0000
CLUSTER              15              5     .33           0   .0000
OBJECT                0              0    1.00           0   .0000
PIPE                  0              0    1.00           0   .0000
JAVA SRCE             0              0    1.00           0   .0000
JAVA RES              0              0    1.00           0   .0000
JAVA DATA             0              0    1.00           0   .0000
11 rows selected.
```

Use the following list to help interpret the contents of the V$LIBRARYCACHE view:

- **namespace** The object type stored in the library cache. The values SQL AREA, TABLE/PROCEDURE, BODY, and TRIGGER show the key types.
- **gets** Shows the number of times an item in library cache was requested.
- **gethits** Shows the number of times a requested item was already in the library cache.
- **gethitratio** Shows the ratio of gethits to gets.
- **pins** Shows the number of times an item in the library cache was executed.
- **pinhits** Shows the number of times an item was executed where that item was already in the library cache.
- **pinhitratio** Shows the ratio of pinhits to pins.
- **reloads** Shows the number of times an item had to be reloaded into the library cache because it aged out or was invalidated.

Keeping the Pin Hit Ratio for Library Cache Items Close to 100 Percent

The pin hit ratio for all library cache items "sum(pinhits) / sum(pins)" should be close to one (or a 100 percent hit ratio). A pin hit ratio of 100 percent means that every time the system needed to execute something, it was already allocated and valid in the library cache. While there will always be some misses the first time a request is made, misses can be reduced by writing identical SQL statements.

TIP

Measure hit ratios for the library cache of the shared pool with the V$LIBRARYCACHE view. A hit ratio of over 95 percent should be achieved. However, when the database is initially started, hit ratios will be around 85 percent.

Keeping the Miss Ratio Less Than 15 Percent

The miss ratio for data dictionary cache "sum(getmisses) / sum(gets)" should be less than 10 to 15 percent. A miss ratio of zero (0) means that every time the system went into the data dictionary cache, it found what it was looking for and did not have to retrieve the information from disk.

If the miss ratio "sum(getmisses) / sum(gets)" is greater than 10–15 percent, the initialization SHARED_POOL_SIZE parameter should be increased.

Using Available Memory to Determine if the SHARED_POOL_SIZE is Set Correctly

The main question that people usually want answered is: "Is there any memory left in the shared pool?" To find out how fast memory in the shared pool is being depleted (made noncontiguous or in use) and also what percent is unused (and still contiguous), run the following query after starting the database and running production queries for a short period of time (for example, after the first hour of the day):

```
col value for 999,999,999,999 heading "Shared Pool Size"
col bytes for 999,999,999,999 heading "Free Bytes"
select to_number(v$parameter.value) value, v$sgastat.bytes,
      (v$sgastat.bytes/v$parameter.value)*100 "Percent Free"
from   v$sgastat, v$parameter
where  v$sgastat.name = 'free memory'
and    v$parameter.name = 'shared_pool_size'
and    v$sgastat.pool = 'shared pool';

Shared Pool Size       Free Bytes Percent Free
---------------- ---------------- ------------
      50,331,648       46,797,132   92.9775476
```

If there is plenty of contiguous free memory (greater than 2MB) after running most of the queries in your production system (you'll have to determine how long this takes), then there is no need to increase the SHARED_POOL_SIZE parameter. I have never seen this parameter go all of the way to zero (Oracle saves a portion for emergency operations via the SHARED_POOL_RESERVED_SIZE parameter).

TIP

The V$SGASTAT view shows how fast the memory in the shared pool is being depleted. Remember that it is only a rough estimate. It shows you any memory that has never been used combined with any piece of memory that has been reused. Free memory will go up and down as the day goes on, depending on how the pieces are fragmented.

Using the X$KSMSP Table to Get a Detailed Look at the Shared Pool

The X$KSMSP table can be queried to get total breakdown for the shared pool. This table will show the amount of memory that is free, memory that is freeable, and memory that is retained for large statements that won't fit into the current shared pool. Consider the following query for a more accurate picture of the shared pool. Refer to Chapter 13 for an in-depth look at this query and how it is adjusted as Oracle is started and as the system begins to access shared pool memory.

```
select    sum(ksmchsiz) Bytes, ksmchcls Status
from      x$ksmsp
group by  ksmchcls;

      BYTES    STATUS
-----------    ------
 50,000,000    R-free
```


```
         40    R-freea
888,326,956    free
    837,924    freeabl
 61,702,380    perm
    359,008    recr
```

Oracle does not state anywhere what the values for status in the X$KSMSP table indicate. In the following table, I offer the following possible descriptions based on the behavior of these values as researched in Chapter 13.

Status	Possible Meaning
Free	This is the amount of contiguous free memory available.
Freeabl	Freeable but not flushable shared memory; currently in use.
Perm	I have read that this is permanently allocated and non-freeable memory, but in testing this, I find that it behaves as free memory not yet moved to the free area for use.
Recr	Allocated memory that is flushable when the shared pool is low on memory.
R-free	This is SHARED_POOL_RESERVED_SIZE (default 5 percent of SP).
R-freea	This is probably reserved memory that is freeable but not flushable.
R-recr	Recreatable chucks of memory in the reserved pool.
R-perm	Permanent chucks of memory in the reserved pool.

TIP

The general rule of thumb is to make the SHARED_POOL_SIZE parameter 50–150 percent of the size of your DB_CACHE_SIZE. In a system that makes use of a large amount of stored procedures or Oracle supplied packages but has limited physical memory, this parameter could make up as much as 150 percent the size of DB_CACHE_SIZE. In a system that uses no stored procedures but has a large amount of physical memory to allocate to DB_CACHE_SIZE, this parameter may be 10–20 percent of the size of DB_CACHE_SIZE. I have worked on larger systems where the DB_CACHE_SIZE was set as high as 100G. Note that in a shared server configuration (previously known as MTS) items from the PGA are allocated from the shared pool rather than the session process space.

Points to Remember about Cache Size

Here are some quick further notes about setting your cache and share pool sizes:

- If the dictionary cache hit ratio is low (below 95 percent), then consider increasing SHARED_POOL_SIZE.
- If the library cache reload ratio is high (>1 percent), then consider increasing SHARED_POOL_SIZE.

- Size the data cache and shared pool appropriately for your systems in terms of workload requirements.

Waits Related to Initialization Parameters

Setting initialization parameters incorrectly will often result in various types of performance issues that will show up as general "waits" or "latch waits" in a STATSPACK report. In Chapter 14, we cover every type of wait and latch issue related to this. The following tables identify some waits and latch waits and their potential fixes.

Wait Problem	Potential Fix
Free buffer	Increase the DB_CACHE_SIZE; shorten the checkpoint; tune the code
Buffer busy	Segment Header — Add freelists or freelist groups or use ASSM
Buffer busy	Data Block — Separate hot data; use reverse key indexes; small block sizes
Buffer busy	Data Block — Increase initrans and/or maxtrans
Buffer busy	Undo Header —Use automatic undo management
Buffer busy	Undo Block — Commit more; use automatic undo management
Latch free	Investigate the detail (listing in next table of this chapter for fixes)
Log buffer space	Increase the log buffer; use faster disks for the redo logs
Scattered read	Indicates many full table scans — tune the code; cache small tables
Sequential read	Indicates many index reads — tune the code (especially joins)
Write complete waits	Adds database writers; checkpoint more often; buffer cache too small

Latch Problem	Potential Fix
Library cache	Use bind variables; adjust the shared_pool_size
Shared pool	Use bind variables; adjust the shared_pool_size
Row cache objects	Increase the shared pool. This is not a common problem.
Cache buffers chain	If you get this latch wait, it means you need to reduce logical I/O rates by tuning and minimizing the I/O requirements of the SQL involved. High I/O rates could be a sign of a hot block (meaning a block highly accessed). Cache *buffer lru chain* latch contention can be resolved by increasing the size of the buffer cache and thereby reducing the rate at which new blocks are introduced into the buffer cache. You should adjust DB_BLOCK_BUFFERS, and possible DB_BLOCK_SIZE. Multiple buffer pools can help reduce contention on this latch. You can create additional *cache buffer lru chain* latches by adjusting the configuration parameter DB_BLOCK_LRU_LATCHES. You may be able to reduce the load on the *cache buffer chain* latches by increasing the configuration parameter. _DB_BLOCK_HASH_BUCKETS may need to be increased or set to a prime number (in pre-9*i* versions).

Some latch problems have often been bug related in the past, so make sure that you check Metalink for issues related to latches. Any of the latches that have a hit ratio below 99 percent should be investigated.

Using Oracle Multiple Buffer Pools

There are pools for the allocation of memory. They relate to the DB_CACHE_SIZE and SHARED_POOL_SIZE. Each of these parameters, which were all-inclusive of the memory they allocate, now has additional options for memory allocation within each memory pool. I will cover each of the two separately.

Pools Related to DB_CACHE_SIZE and Allocating Memory for Data

In this section, we will focus on the Oracle pools that are used to store the actual data in memory. The initialization parameters DB_CACHE_SIZE, DB_KEEP_CACHE_SIZE, and DB_RECYCLE_CACHE_SIZE will be the determining factors for memory used to store data. DB_CACHE_SIZE refers to the total size in bytes of the main buffer cache (or memory for data) in the SGA. Two additional buffer pools are DB_KEEP_CACHE_SIZE and DB_RECYCLE_CACHE_SIZE. These additional two pools serve the same purpose as the main buffer cache (DB_CACHE_SIZE), with the exception that the algorithm to maintain the pool is different for all three available pools. Note that the BUFFER_POOL_KEEP, DB_BLOCK_BUFFERS, and BUFFER_POOL_RECYCLE parameters have been deprecated and should no longer be used. Unlike BUFFER_POOL_KEEP and BUFFER_POOL_RECYCLE, DB_KEEP_CACHE_SIZE and DB_RECYCLE_CACHE_SIZE are not subtracted from DB_CACHE_SIZE; they are allocated in addition to DB_CACHE_SIZE.

The *main buffer cache* (defined by *DB_CACHE_SIZE*) maintains the LRU (least recently used) list and flushes the oldest buffers in the list. While all three pools utilize the LRU replacement policy, the goal for the main buffer cache is to fit most data being used in memory.

The *keep pool* (defined by *DB_KEEP_CACHE_SIZE*) is hopefully never flushed; it is intended for buffers that you want to be "pinned" indefinitely (buffers that are very important and need to stay in memory). Use the keep pool for small tables (that will fit in their entirety in this pool) that are frequently accessed and need to be in memory at all times.

The *recycle pool* (defined by *DB_RECYCLE_CACHE_SIZE*) is a pool from which you expect the data to be regularly flushed, since there is too much data being accessed to stay in memory. Use the recycle pool for large, less important data that is usually accessed only once in a long while (usually ad hoc user tables for inexperienced users are put here).

The following examples provide a quick look on how information is allocated to the various buffer pools. Remember, if no pool is specified, then the buffers in the main pool are used.

1. Create a table that will be stored in the keep pool upon being accessed:

```
Create table state_list (state_abbrev varchar2(2), state_desc varchar2(25))
Storage (buffer_pool keep);
```

2. Alter the table to the recycle pool:

```
Alter table state_list storage (buffer_pool recycle);
```

3. Alter the table back to the keep pool:

```
Alter table state_list storage (buffer_pool keep);
```

4. Find the disk and memory reads in the keep pool:

```
select   physical_reads "Disk Reads",
         db_block_gets + consistent_gets "Memory Reads"
from     v$buffer_pool_statistics
where    name = 'KEEP';
```

Modifying the LRU Algorithm

In this section, we're going to go over the deep edge for experts only. Skip this section if you've used Oracle for only a decade or less. There are five undocumented initialization parameters (defaults are in parentheses) that can be used to alter the LRU algorithm for greater efficiency when you really have studied and understand your system buffer usage well:

- **_db_percent_hot_default (50)** The percent of buffers in the hot region
- **_db_aging_touch_time (3)** Seconds that must pass to increment touch count again
- **_db_aging_hot_criteria (2)** Threshold to move a buffer to the MRU end of LRU chain
- **_db_aging_stay_count (0)** Touch count reset to this when moved to MRU end
- **_db_aging_cool_count (1)** Touch count reset to this when moved to LRU end

We can see that by decreasing the value of the first of these parameters, we allow buffers to remain longer; setting it higher will cause a flush sooner. Setting parameter 2 lower will give higher value to buffers that are executed a lot in a short period of time. Parameters 3, 4, and 5 all relate to how quickly to move things from the hot end to the cold end and how long they stay on each end.

Pools Related to SHARED_POOL_SIZE and Allocating Memory for Statements

In this section, we will focus on the pools that are used to store the actual statements in memory. Unlike the pools related to the data, the LARGE_POOL_SIZE is allocated outside the memory allocated for SHARED_POOL_SIZE, but it is still part of the SGA.

The LARGE_POOL_SIZE is a pool of memory used for the same operations as the shared pool. Oracle defines this as the size set aside for large allocations of the shared pool. You'll have to do your own testing to ensure where the allocations are coming from in your system and version of Oracle. The minimum setting is 300K, but the setting must also be as big as the _LARGE_POOL_MIN_ALLOC, which is the minimum size of shared pool memory requested that will force an allocation in the LARGE_POOL_SIZE memory. Unlike the shared pool, the large pool does not have an LRU list. Oracle does not attempt to age memory out of the large pool.

You can view your pool settings by querying the V$PARAMETER view:

```
select    name, value, isdefault, isses_modifiable, issys_modifiable
from      v$parameter
where     name like '%pool%'
and       isdeprecated <> 'TRUE'
order by 1;
```

```
NAME                           VALUE ISDEFAULT ISSES ISSYS_MOD
------------------------- ---------- --------- ----- ---------
java_pool_size              54525952    FALSE     FALSE IMMEDIATE
large_pool_size              2516582    FALSE     FALSE IMMEDIATE
olap_pool_size               8388608    FALSE     FALSE DEFERRED
shared_pool_reserved_size    4613734    FALSE     FALSE FALSE
shared_pool_size           134217728    TRUE      FALSE IMMEDIATE
streams_pool_size                       TRUE      FALSE IMMEDIATE
6 rows selected.
```

TIP
The additional buffer pools (memory for data) available in Oracle are initially set to zero.

Tuning the PGA_AGGREGATE_TARGET for Optimal Use of Memory

The PGA_AGGREGATE_TARGET specifies the total amount of session PGA memory that Oracle will attempt to allocate across all sessions. PGA_AGGREGATE_TARGET was introduced in Oracle 9*i* and should be used in place of the *_SIZE parameters such as SORT_AREA_SIZE. Also, in Oracle 9*i,* the PGA_AGGREGATE_TARGET parameter does not automatically configure ALL *_SIZE parameters. For example, both the LARGE_POOL_SIZE and JAVA_POOL_SIZE parameters are not affected by PGA_AGGREGATE_TARGET. The advantage of using PGA_AGGREGATE_TARGET is the ability to cap the total user session memory to minimize OS paging.

When PGA_AGGREGATE_TARGET is set, WORKAREA_SIZE_POLICY must be set to AUTO. Like the V$DB_CACHE_ADVICE view, the V$PGA_TARGET_ADVICE (Oracle 9.2 and later versions) and V$PGA_TARGET_ADVICE_HISTOGRAM views exist to assist in tuning the PGA_AGGREGATE_TARGET. Oracle Enterprise Manager provides graphical representations of these views.

The PGA_AGGREGATE_TARGET should be set to attempt to keep the ESTD_PGA_CACHE_HIT_PERCENTAGE greater than 95 percent. By setting this appropriately, more data will be sorted in memory that may have been sorted on disk. The next query returns the minimum value for the PGA_AGGREGATE_TARGET that is projected to yield a 95 percent or greater cache hit ratio:

```
select min(pga_target_for_estimate)
from   v$pga_target_advice
where  estd_pga_cache_hit_percentage > 95;

MIN(PGA_TARGET_FOR_ESTIMATE)
----------------------------
                    12582912
```

Modifying the Size of Your SGA to Avoid Paging and Swapping

Before you increase the size of your SGA, you must understand the effects on the physical memory of your system. If you increase parameters that use more memory than what is available on your

Color profile: Generic CMYK printer profile
Composite Default screen

system, then serious degradation in performance may occur. When your system processes jobs, if it doesn't have enough memory, it will start paging or swapping to complete the active task.

When *paging* occurs, information that is *not currently* being used is moved from memory to disk. This allows memory to be used by a process that *currently* needs it. If paging happens a lot, the system will experience decreases in performance, causing processes to take longer to run.

When *swapping* occurs, an *active* process is moved from memory to disk temporarily so that another *active* process that also desires memory can run. Swapping is based on system cycle time. If swapping happens a lot, your system is dead. Depending on the amount of memory available, an SGA that is too large can cause swapping.

Understanding the Cost-Based Optimizer

The cost-based optimizer was built to make your tuning life easier by choosing better paths for your poorly written queries. Rule-based optimization was built on a set of rules on how Oracle processes statements. Oracle 10*g* Release 2 now only supports the use of the cost-based optimizer; the rule-based optimizer is no longer supported. Oracle 10*g* Release 2 has automatic statistics gathering turned on to aid the effectiveness of the cost-based optimizer. In Oracle, many features are only available when using cost-based optimization. The cost-based optimizer now has two modes of operation, normal mode, and tuning mode. Normal mode should be used in production and test environments; tuning mode can be used in development environments to aid developers and DBAs in testing specific SQL code.

How Optimization Looks at the Data

Rule-based optimization is *Oracle-centric,* while cost-based optimization is *data-centric.* The optimizer mode under which the database operates is set via the initialization parameter OPTIMIZER_MODE. The possible optimizer modes are as follows:

- **CHOOSE** Uses cost-based optimization for all analyzed tables. This is a good mode for well-built and well-tuned systems (for advanced users). This option is not documented for 10*g*R2 but is still usable.
- **RULE** Always uses rule-based optimization. If you are still using this, you need to start using cost-based optimization, as rule-based optimization is no longer supported under Oracle 10*g* Release 2.
- **FIRST_ROWS** Gets the first row faster (generally forces index use). This is good for untuned systems that process lots of single transactions (for beginners).
- **FIRST_ROWS (1|10|100|1000)** Gets the first *n* rows faster. This is good for applications that routinely display partial results to users such as paging data to a user in a web application.
- **ALL_ROWS** Gets all rows faster (generally forces index suppression). This is good for untuned, high-volume batch systems (usually not used).

The default optimizer mode for Oracle 10*g* Release 2 is ALL_ROWS. Also, cost-based optimization will be used even if the tables are not analyzed.

P:\010Comp\OracleT&T\305-9\ch04.vp
Friday, May 18, 2007 11:28:16 AM

NOTE
The optimizer in Oracle 10g Release 2 uses cost-based optimization regardless of whether the tables have been analyzed or not.

TIP
There is no OPTIMIZER MODE called COST (a misconception). If you are using Oracle Database 9i Release 2 or an earlier version and are not sure what optimizer mode to use, then use CHOOSE or FIRST_ROWS and analyze all tables. By doing this, you will be using cost-based optimization. As the data in a table changes, tables need to be re-analyzed at regular intervals. Oracle 10g Release 2 automatically does this right out of the box.

Creating Enough Dispatchers

When using a shared server, some of the things you need to watch for are high busy rates for the existing dispatcher processes and increases in wait times for response queues of existing dispatcher processes. If the wait time increases, as the application runs under normal use, you may wish to add more dispatcher processes, especially if the processes are busy more than 50 percent of the time.

Use the following statement to determine the busy rate:

```
select    Network,
          ((Sum(Busy) / (Sum(Busy) + Sum(Idle))) * 100) "% Busy Rate"
from      V$Dispatcher
group by Network;

NETWORK        % Busy Rate
TCP1                     0
TCP2                     0
```

Use the following statement to check for responses to user processes that are waiting in a queue to be sent to the user:

```
select    Network Protocol,
          Decode (Sum(Totalq), 0, 'No Responses',
          Sum(Wait) / Sum(TotalQ) || ' hundredths of a second')
          "Average Wait Time Per Response"
from      V$Queue Q, V$Dispatcher D
where     Q.Type = 'DISPATCHER'
and       Q.Paddr = D.Paddr
group by Network;

PROTOCOL         Average Wait Time Per Response
TCP1             0 hundredths of a second
TCP2             1 hundredths of a second
```

Use the following statement to check the requests from user processes that are waiting in a queue to be sent to the user:

```
select    Decode (Sum(Totalq), 0, 'Number of Requests',
          Sum(Wait) / Sum(TotalQ) || 'hundredths of a second')
          "Average Wait Time Per Request"
from      V$Queue
where     Type = 'COMMON';

Average Wait Time Per Request
12 hundredths of a second
```

Open Cursors

If you don't have enough open cursors, then you will receive errors to that effect. The key is to stay ahead of your system by increasing the OPEN_CURSORS initialization parameter before you run out of open cursors.

25 Important Initialization Parameters to Consider

1. **DB_CACHE_SIZE** Initial memory allocated to data cache or memory used for data itself.
2. **SGA_TARGET** If you use Oracle's Automatic Shared Memory Management, this parameter is used to automatically determine the size of your data cache, shared pool, large pool, and Java pool (see Chapter 1 for more information). Setting this to 0 disables it.
3. **PGA_AGGREGATE_TARGET** Soft memory cap for total of all users' PGAs.
4. **SHARED_POOL_SIZE** Memory allocated for data dictionary and SQL and PL/SQL.
5. **SGA_MAX_SIZE** Maximum memory that the SGA can dynamically grow to.
6. **OPTIMIZER_MODE** CHOOSE, RULE, FIRST_ROWS, FIRST_ROWS_*n* or ALL_ROWS. Although RULE is definitely desupported and obsolete and people are often scolded for even talking about it, I was able to set the mode to RULE in 10*g*.
7. **CURSOR_SHARING** Converts literal SQL to SQL with bind variables, reducing parse overhead.
8. **OPTIMIZER_INDEX_COST_ADJ** Coarse adjustment between the cost of an index scan and the cost of a full table scan. Set between 1 and 10 to force index use more frequently. Setting this parameter to a value between 1 and 10 would pretty much guarantee index use, even when not appropriate, so be careful, since it is highly dependent on the index design and implementation being correct. Please note that if you using Applications 11i: Setting OPTIMIZER_INDEX_COST_ADJ to any value other than the default (100) is not supported (see Note 169935.1). Also, see bug 4483286.
9. **QUERY_REWRITE_ENABLED** Used to enable Materialized View and Function-Based-Index capabilities and other features in some versions.
10. **DB_FILE_MULTIBLOCK_READ_COUNT** For full table scans to perform I/O more efficiently, this reads this many blocks in a single I/O.
11. **LOG_BUFFER** Buffer for uncommitted transactions in memory (not dynamic; set in pfile).
12. **DB_KEEP_CACHE_SIZE** Memory allocated to keep pool or an additional data cache that you can set up outside the buffer cache for very important data that you don't want pushed out of the cache.

13. **DB_RECYCLE_CACHE_SIZE** Memory allocated to a recycle pool or an additional data cache that you can set up outside the buffer cache and in addition to the keep cache described in the preceding item. Usually, DBAs set this up for ad hoc user query data that has queries that are poorly written.
14. **DBWR_IO_SLAVES** (also DB_WRITER_PROCESSES if you have async I/O) Number of writers from SGA to disk for simulated async I/O. If you have async I/O, then you use DB_WRITER_PROCESSES to set up multiple writers to more quickly write out dirty blocks during a database write (DBWR).
15. **LARGE_POOL_SIZE** Total blocks in the large pool allocation for large PL/SQL and a few other Oracle options less frequently used.
16. **STATISTICS_LEVEL** Used to enable advisory information and optionally keep additional OS statistics to refine optimizer decisions. TYPICAL is the default.
17. **JAVA_POOL_SIZE** Memory allocated to the JVM for Java stored procedures.
18. **JAVA_MAX_SESSIONSPACE_SIZE** Upper limit on memory that is used to keep track of user session state of JAVA classes.
19. **MAX_SHARED_SERVERS** Upper limit on shared servers when using shared servers.
20. **WORKAREA_SIZE_POLICY** Used to enable automatic PGA size management.
21. **FAST_START_MTTR_TARGET** Bounds time to complete a crash recovery. This is the time (in seconds) that the database will take to perform crash recovery of a single instance. If you set this parameter, LOG_CHECKPOINT_INTERVAL should *not* be set to 0. If you don't set this parameter, you can still see your estimated MTTR (mean time to recovery) by querying V$INSTANCE_RECOVERY for ESTIMATED_MTTR.
22. **LOG_CHECKPOINT_INTERVAL** Checkpoint frequency (in OS blocks—most OS blocks are 512 bytes) where Oracle performs a database write of all dirty (modified) blocks to the datafiles in the database. Oracle will also perform a checkpoint if more the one quarter of the data buffers are dirty in the db cache and also on any log switch. The LGWR (log writer) also updates the SCN in the control files and datafiles with the SCN of the checkpoint.
23. **OPEN_CURSORS** Specifies the size of the private area used to hold (open) user statements. If you get "ORA-01000: maximum open cursors exceeded," you may need to increase this parameter, but make sure you are *closing* cursors that you no longer need. Prior to 9.2.0.5, these open cursors were also cached and at times caused issues (ORA-4031) if OPEN_CURSORS was set too high. In 9.2.05, SESSION_CACHED_ CURSORS now controls the setting of the PL/SQL cursor cache. Do *not* set the parameter SESSION_CACHED_CURSORS as high as you set OPEN_CURSORS or you may experience ORA-4031 or ORA-7445 errors.
24. **DB_BLOCK_SIZE** Default block size for the database. A smaller block size will reduce contention by adjacent rows, but a larger block size will lower the number of I/Os needed to pull back more records. A larger block size will also help in range scans where the blocks desired are sequentially stored.

25. **OPTIMIZER_DYNAMIC_SAMPLING** Controls the number of blocks read by the dynamic sampling query. Very useful with systems that are using Global Temporary Tables.

TIP

Setting certain initialization parameters correctly could be the difference between a report taking two seconds and two hours. Test changes on a test system thoroughly before implementing those changes in a production environment.

Initialization Parameters over the Years

Oracle has moved from a point where there were over four times as many documented parameters as undocumented in Oracle 6 to where the undocumented parameters exceeded the documented in Oracle 8*i* to where there are four times as many undocumented as documented parameters in Oracle 10*g*. Clearly, we have moved to a place where there are more dials to set in 10*g* for the experts (undocumented), but the number of dials to set for the standard database setup (documented parameters) is not increasing any more and is becoming standardized. The following table charts the changing numbers of documented and undocumented parameters:

Version	Documented	Undocumented	Total
6	111	19	130
7	117	68	185
8.0	193	119	312
8.1	203	301	504
9.0	251	436	687
9.2	257	540	797
10.2	257 (+0%)	1124 (+108%)	1381 (+73%)

Finding Undocumented Initialization Parameters

Querying the table X$KSPPI shows you documented as well as undocumented initialization parameters. The query may only be done as user SYS, so be careful. See Chapter 13 for a complete look at the X$ tables. My top 13 undocumented initialization parameters are listed in Appendix A. Appendix C gives a complete listing as of the writing of this book of the X$ tables.

```
select ksppinm, ksppstvl, ksppstdf
from   x$ksppi a, x$ksppcv b
where  a.indx = b.indx
order  by ksppinm;
```

The following is a brief description of the columns in the x$ksppi & x$ksppcv tables:

- **KSPPINM** Parameter name
- **KSPPSTVL** Current value for the parameter
- **KSPPSTDF** Default value for the parameter

A partial output listing of the initialization parameters is shown here:

```
KSPPINM                        KSPPSTVL             KSPPSTDF
------------------------------ -------------------- ----------
...
_write_clones                  3                    TRUE
_yield_check_interval          100000               TRUE
active_instance_count                               TRUE
aq_tm_processes                1                    FALSE
archive_lag_target             0                    TRUE
...
```

TIP
Using undocumented initialization parameters can cause corruption. Never use these if you are not an expert and you are not directed by Oracle Support! Ensure that you work with Oracle Support before setting these parameters.

Understanding the Typical Server

The key to understanding Oracle is to understand its dynamic nature. Oracle continues to have many attributes of previous versions while also leading the way by implementing the future of distributed database and object-oriented programming. Experience from earlier versions of Oracle always benefits the DBA in future versions of Oracle. Here are some of the future changes to consider as you build your system:

- Oracle can be completely distributed and maintained at a single point. (Many databases and locations with one DBA managing the system looks like the corporate future.)
- Database maintenance is becoming completely visual (all point-and-click maintenance as in the Enterprise Manager). The V$ views are still your lowest-performance cost access method, but Enterprise Manager is easier to use for more complex inquiries that may require multiple V$ views to get the same result.
- Network throughput continues to be an issue that looks to be solved by technology (next three or so years).
- CPUs will continue to get faster, eliminating the CPU as a system resource issue. (I/O and correct design will continue to be the issues.)

Color profile: Generic CMYK printer profile
Composite Default screen

- Object-oriented development will be crucial to rapid system development.
- Current database design theory is being rewritten to focus more on denormalization.
- Graphics are causing the sizes of databases to become increasingly large. Also, the fact that disk space is getting cheaper and cheaper has made businesses more willing to keep data around longer.

Modeling a Typical Server

This section contains rough estimates designed as setup guidelines. However, it is important to emphasize that these are only guidelines and that the reality is that every system is different and must be tuned to meet the system's demands. (CPU speed will depend on the type of processor, e.g., RISC vs. Intel.) The following table does not include guidelines for Oracle Applications. Oracle Applications tends to have unique issues that are addressed by Oracle in the application documentation and on Metalink.

Database Size	Up to 25GB	100–200GB	500–1,000GB
Number of users	100	200	500
Number of CPUs	4	8	16+
System memory	8GB	16GB	32GB+
SGA_MAX_SIZE	2GB	4GB	8GB
PGA_AGGREGATE_TARGET	512MB	1GB	2GB
Total disk capacity	100GB	500–1000GB	1–50TB
Percentage of query	75 percent	75 percent	75 percent
Percentage of DML	25 percent	25 percent	25 percent
Number of redo logs multiplexed?	4–8 Yes	6–10 Yes	6–12 Yes
Number of control files	4	4	4
Percent batch	20 percent	20 percent	20 percent
Percent online	80 percent	80 percent	80 percent
Archiving used?	Yes	Yes	Yes
Buffer hit ratio	95 percent +	95 percent +	95 percent +
Dictionary hit ratio	95 percent +	95 percent +	95 percent +
Library hit ratio	95 percent +	95 percent +	95 percent +
Other system software (other than Oracle)	Minimum	Minimum	Minimum

P:\010Comp\OracleT&T\305-9\ch04.vp
Friday, May 18, 2007 11:29:58 AM

Database Size	Up to 25GB	100–200GB	500–1,000GB
Use raw devices?	No	No	No
Use parallel query?	Depends on queries	Depends on queries	Probably in many queries

The following variables can be reason to deviate from the typical server configuration:

- Heavy batch processing may need much larger rollback or undo, redo, and temp tablespace sizes.
- Heavy DML processing may need much larger rollback or undo, redo, and temp tablespace sizes.
- Heavy user access to large tables requires more CPU and memory, and larger temp tablespace sizes.
- Poorly tuned systems require more CPU and memory, and larger temp tablespace sizes.
- A greater number of disks and controllers always increase performance by reducing I/O contention.
- An increase in the disk capacity can speed backup and recovery time by going to disk and not tape.

Sizing the Oracle Applications Database

Oracle recommends via Metalink Note 216205.1 (written by Oracle Applications Development) the SGA settings shown in Table 4-1. This is also a nice guideline for sizing systems. Note that the SGA_TARGET takes the place of the other memory parameters and allows Oracle to allocate memory where needed. I don't think this should necessarily be used in all cases (in the Metalink note it is recommended for 10*g*); be careful to test this well if you use it as it is a new parameter. If you do set the initialization parameter SGA_TARGET to allow Oracle to use Automatic Shared Memory Management (ASMM), you can query the view V$SGA_DYNAMIC_COMPONENTS to see where (i.e., buffer cache, shared pool, etc.) the memory is being allocated. The SGA_TARGET parameter cannot be set larger than the SGA_MAX_SIZE or you will receive the ORA-00823 error.

The CSP and NOCSP options of the shared pool–related parameters refer to the use of cursor_space_for_time, which is documented in the common database initialization parameters section. The use of cursor space for time results in much larger shared pool requirements.

The Development / Test instance refers to a small instance used for only development or testing in which no more than 10 users exist. The range of user counts provided in the table refers to active Applications users, not total or named users. For example, if you plan to support a maximum of 500 active Oracle Applications users, then you should use the sizing per the range 101–500 users. The parameter values provided in this document reflect a development / test instance configuration, and you should adjust the relevant parameters according to the Applications user counts (refer to the table).

Color profile: Generic CMYK printer profile
Composite Default screen

Parameter Name	Development / Test Instance	11–100 Users	101–500 Users	501–1,000 Users	1001–2000 Users[1]
Processes	200	200	800	1200	2500
Sessions	400	400	1600	2400	5000
db_block_buffers	20000	50000	150000	250000	400000
db_cache_size[2]	156M	400M	1G	2G	3G
sga_target[3]	1G	1G	2G	3G	14G
undo_retention[4]	1800	3600	7200	10800	14400
Shared_pool_size (csp)	N/A	N/A	N/A	1800M	3000M
Shared_pool_reserved_size (csp)	N/A	N/A	N/A	180M	300M
Shared_pool_size (no csp)	400M	600M	800M	1000M	2000M
Shared_pool_reserved_size (no csp)	40M	60M	80M	100M	100M
pga_aggregate_target[5]	1G	2G	4G	10G	20G
Total Memory Required[6]	~ 2GB	~ 3GB	~ 6GB	~ 13GB	~ 25GB

[1]For instances supporting a minimum of 1000 Oracle Applications users, you should use the Oracle 64-bit Server for your platform in order to support large SGAs.

[2]The parameter db_cache_size should be used for 9*i*-based environments in place of db_block_buffers.

[3]The parameter sga_target should be used for 10*g*-based environments (I suggest that if you use this parameter, you test this well before handing over full memory management to Oracle).

[4]The values for undo_retention are recommendations only, and this parameter should be adjusted according to the elapsed times of the concurrent jobs and corresponding commit windows. It is not required to set undo_retention for 10*g*-based systems, as undo retention is automatically set as part of automatic undo tuning.

[5]pga_aggregate_target should only be used with a 9*i*- or 10*g*-based database instances. This parameter should not be set in 8*i*-based instances.

[6]The total memory required refers to the amount of memory required for the data server instance and associated memory, including the SGA and the PGA. You should ensure that your system has sufficient available memory in order to support the values provided in the table. The values provided should be adjusted in accordance with available memory so as to prevent swapping and paging.

TABLE 4-1. *SGA Settings*

Tips Review

- The key initialization parameters in Oracle are SGA_MAX_SIZE, PGA_AGGREGATE_TARGET, DB_CACHE_SIZE, and SHARED_POOL_SIZE. If you use ASMM, then SGA_TARGET is the key initialization parameter.
- If you can't figure out why your system isn't using the value in your init.ora file, you probably have an spfile overriding it. And don't forget, you can also use a hint to override parameters at the query level in 10*g*R2.

P:\010Comp\OracleT&T\305-9\ch04.vp
Friday, May 18, 2007 11:30:36 AM

- Changing initialization parameters dynamically is a powerful feature for both developers and DBAs. Consequently, a user with the ALTER SESSION privilege is capable of irresponsibly allocating 100M+ for the SORT_AREA_SIZE for a given session, if it is not restricted.
- In Oracle 10*g* Release 2, use the Enterprise Manager Grid Control to find problem queries.
- Physical memory is generally much faster than retrieving data from disk, so make sure that the SGA is large enough to accommodate memory reads when it is effective to do so.
- Poor joins and poor indexing also yield very high hit ratios, so make sure that your hit ratio isn't high for a reason other than a well-tuned system. An unusually high hit ratio may indicate the introduction of code that is poorly indexed or includes join issues.
- Hit ratios are useful to experienced DBAs but can be misleading to inexperienced DBAs. The best use of hit ratios is still to compare over time to help alert you to a substantial change to a system on a given day. While there are those who don't like using hit ratios, they are usually tool vendors who don't see the value of tracking hit ratios over time, since their tools are point-in-time or reactive-based tuning solutions. Hit ratios should never be your only tool, but they should definitely be one of many proactive tools in your arsenal.
- In Oracle 10*g* Release 2, use the TopSQL monitor of Oracle's SQL Analyze to find problem queries.
- A low hit ratio for a query is an indication of a missing or suppressed index**.**
- Bad (slow) queries show in V$SQLAREA view with poor hit ratios the first time they are executed. Make sure you tune them at that time. The second time that they execute, they may not show a poor hit ratio.
- The database must be rebuilt if you increase the DB_BLOCK_SIZE. Increasing the DB_FILE_MULTIBLOCK_READ_COUNT will allow more block reads in a single I/O, giving a benefit similar to a larger block size.
- SQL must be written *exactly* the same to be reused. Case differences and any other differences will cause a reparse of the statement.
- Measure hit ratios for the data dictionary row cache of the shared pool with the V$ROWCACHE view. A hit ratio of over 95 percent should be achieved. However, when the database is initially started, hit ratios will be around 85 percent.
- Measure hit ratios for the library cache of the shared pool with the V$LIBRARYCACHE view. A hit ratio of over 95 percent should be achieved. However, when the database is initially started, hit ratios will be around 85 percent.
- The V$SGASTAT view shows how fast the memory in the shared pool is being depleted. Remember that it is only a rough estimate. It shows you any memory that has never been used combined with any piece of memory that has been reused. Free memory will go up and down as the day goes on according to how the pieces are fragmented.
- The general rule of thumb is to make the SHARED_POOL_SIZE parameter 50–150 percent of the size of your DB_CACHE_SIZE.

Color profile: Generic CMYK printer profile
Composite Default screen

- The additional buffer pools (memory for data) available in Oracle are initially set to zero.
- The optimizer in Oracle 10*g* Release 2 uses cost-based optimization regardless of whether the tables have been analyzed or not.
- Setting certain initialization parameters correctly could be the difference between a report taking two seconds and two hours. Test changes on a test system thoroughly before implementing those changes in a production environment.
- Using undocumented initialization parameters can cause corruption. Never use these if you are not an expert and you are not directed by Oracle Support! Ensure that you work with Oracle Support before setting these parameters.

References

Craig Shallahamer, *All about Oracle's Touch-Count Data Block Buffer Algorithm* (OraPub, excellent)
Rich Niemiec, *DBA Tuning; Now YOU are the Expert* (TUSC)
Performance Tuning Guide, Oracle Corporation

Thanks to Randy Swanson, who did the update for this chapter in the 9*i* version of the book. (Where were you this time around?)

P:\010Comp\OracleT&T\305-9\ch04.vp
Friday, May 18, 2007 11:31:03 AM