MONITORING AND TUNING ORACLE - CHAPTER 22 PART 1

This material is extracted from the first half of Chapter 22 of *Configuring and Tuning Databases on the Solaris Platform*, by Allan N. Packer, (c) 2002, Sun Microsystems Press. The second half of the chapter will appear in the August edition of Sun BluePrints Online. Chapter 21, *Drill-Down Monitoring of Database Servers,* was presented in the June 2002 edition of Sun BluePrints Online.

In this chapter we consider tuning recommendations for Oracle in both OLTP and DSS environments after first examining methods of monitoring and configuring Oracle. We also explore Oracle9i enhancements that support dynamic reconfiguration, and we investigate issues related to crash recovery.

Managing Oracle Behavior

Oracle can be monitored and managed with Oracle Enterprise Manager (OEM), a powerful GUI-based tool that allows detailed monitoring of all aspects of database behavior and that supports database management. Oracle also provides access to the database information stored in its memory-resident performance tables (often referred to simply as system tables). This information can be retrieved either with SQL or supplied scripts. In this section we discuss these monitoring methods, explore ways of displaying and changing Oracle tunable parameters, and consider explain plans (query execution plans).

Running Administrative Commands

Starting and shutting down Oracle require special privileges, as do `alter system` statements. The method of connecting to Oracle to run administrative commands has changed more than once over the last few releases; the different methods for the major versions are shown below.

Before Oracle7.3, the sqldba command was used as shown in the following example based on Oracle7.1.3.

```
alameda% sqldba mode=line

SQL*DBA: Release 7.1.3.2.0 - Production on Mon Aug 6 12:10:17
2001
Copyright (c) Oracle Corporation 1979, 1994. All rights reserved.

Oracle7 Server Release 7.1.3.2.0 - Production Release
With the parallel query option
PL/SQL Release 2.1.3.2.0 - Production

SQLDBA> connect internal
Connected.
```

From Oracle7.3, the most commonly used command was svrmgrl, as shown in the following example based on Oracle 8.0.5.

```
1.oracle8 svrmgrl

Oracle Server Manager Release 3.0.5.0.0 - Production

(c) Copyright 1997, Oracle Corporation.  All Rights Reserved.

Oracle8 Enterprise Edition Release 8.0.5.0.0 - Production
PL/SQL Release 8.0.5.0.0 - Production

SVRMGR> connect internal
Connected.
```

From Oracle9i, svrmgrl is no longer supported. The approved connection method is based on the sqlplus command. This method also works with earlier versions of Oracle, such as Oracle8 and Oracle8i. The connection can be achieved in two steps, as shown in the example below, which is based on Oracle8.1.5.

```
oracle8.1.5% sqlplus /nolog

SQL*Plus: Release 8.1.5.0.0 - Production on Mon Aug 6
12:16:19 2001

(c) Copyright 1999 Oracle Corporation.  All rights reserved.

SQL> connect / as sysdba
Connected.
```

The same effect can be achieved with a single command, as shown in the following example based on Oracle9.0.1.

```
pae280% sqlplus "/ as sysdba"

SQL*Plus: Release 9.0.1.0.0 - Production on Mon Aug 6 12:19:15
2001

(c) Copyright 2001 Oracle Corporation.  All rights reserved.

Connected to an idle instance.
```

Throughout the rest of this chapter, I use "sysdba" as an abbreviation of the command sequences used to connect to Oracle to run administrative commands. You should substitute the appropriate command for your Oracle release (sqldba, svrmgrl, or sqlplus).

Viewing Current Oracle Tunable Parameters

You can display parameter settings for the current Oracle instance by running the show parameters command as sysdba. You can also display the settings for a single parameter or a group of parameters. For example, to display all settings for parameters containing the string block, run the following command as sysdba:

```
SQL> show parameter block

NAME                                 TYPE     VALUE
------------------------------------ -------  ------------
db_block_buffers                     integer  8192
db_block_checking                    boolean  FALSE
db_block_checksum                    boolean  FALSE
db_block_lru_latches                 integer  1
db_block_max_dirty_target            integer  8192
db_block_size                        integer  2048
db_file_multiblock_read_count        integer  8
hash_multiblock_io_count             integer  0
sort_multiblock_read_count           integer  2
```

Changing Tunable Parameters for Oracle

Most Oracle tunables reside in a file called init${ORACLE_SID}.ora (usually referred to as init.ora), where $ORACLE_SID is the environment variable used to set the instance ID of the current Oracle instance. The init.ora file is typically located in the $ORACLE_HOME/dbs directory.

This file allows the database administrator to set values for the tunable parameters that determine the behavior of the Oracle instance. System default values are used for any parameters that are not set. The parameter values in the init.ora file are only used when Oracle is started.

Some parameters can be changed dynamically with the set clause of the alter system commands; the number of such parameters has increased with recent versions of Oracle.

Some sites also use a `config.ora` file, referenced from `init.ora` with the `ifile` parameter, to store static parameters such as `db_name` and `db_block_size`.

Making Dynamic Parameter Changes Persistent

Oracle9i introduced a method of storing and maintaining configuration parameters based on a new *Server Parameter File* (`spfile`). As we have seen, tunable parameters can be changed dynamically with the `set` clause of the `alter system` statement. The `spfile` allows such changes to survive a database reboot. Without the `spfile`, all changes are lost when the database is shut down; unless the database administrator remembers to separately update the `init.ora` file, changes do not persistent across database reboots.

An `spfile` can be created from an `init.ora` file by the following statement run as `sysdba`:

```
SQL> create spfile='$ORACLE_HOME/dbs/spfileaccts.ora'
  2  from pfile='$ORACLE_HOME/dbs/initaccts.ora';

File created.
```

It is not actually necessary to supply the `spfile` name; if no name is specified, the name and location of the new `spfile` will default to `$ORACLE_HOME/dbs/spfile$ORACLE_SID.ora`. The `spfile` is a binary file that must not be manually edited; changes should be made with the `alter system` statement instead.

After an `spfile` is created, the database must be shut down and restarted before the file takes effect. If a startup command is issued without a `pfile` clause, the server parameter file will be used rather than the `init.ora` file. You can still boot Oracle with the `init.ora` file by supplying a `pfile` clause identifying the `init.ora` file. The new `SPFILE` configuration parameter can be used to specify the location of the `spfile`.

When parameters are modified with the `alter system` statement, a `scope` clause can be used to specify the scope of the change. Supported values are:

- **`scope=spfile`.** The change is made to the `spfile` only. Changes to both dynamic and static parameters take effect only when the database is next started.
- **`scope=memory`.** The change is applied to the running instance only and takes immediate effect for dynamic parameters. This option is not supported for static parameters.
- **`scope=both`.** The change is applied to both the `spfile` and the running instance and takes immediate effect for dynamic parameters. This option is not supported for static parameters.

The default scope is both if the database was started with an spfile, and memory if it was not.

A parameter can be returned to its system default value with the following statement:

```
alter system set parameter = '';
```

You can create an init.ora file from an spfile with the following command:

```
create pfile='$ORACLE_HOME/dbs/backup_initaccts.ora
        from spfile='$ORACLE_HOME/dbs/spfileaccts.ora'
```

The file names can be eliminated if default names are used for init.ora and spfile.

Finally, the current active parameters can be viewed with the show parameters statement or by querying the v$parameter view (or the v$parameter2 view). The v$spparameter view displays the current contents of the spfile, or NULL values if the spfile is not in use.

Viewing and Changing Hidden Parameters

As well as the init.ora parameters described above, Oracle includes a number of hidden init.ora parameters, each of which begins with an underscore (_). These hidden parameters can be set in the init.ora file just as for the normal parameters. There are occasions when modifying a hidden parameter can prove beneficial for performance reasons, and later in this chapter I identify some situations where modifying a hidden parameter might be helpful.

Let me issue an **Important Disclaimer**, though: the parameters are hidden by Oracle for a reason! Before changing them on a production system, discuss your plans with Oracle support. I will take no responsibility for database corruption or other problems resulting from your unsupported use of hidden parameters, and you should expect Oracle and Sun to take the same position.

That said, the following query will display hidden parameters for Oracle8 and later releases:

```
select a.ksppinm "name", a.ksppdesc "description",
        b.ksppstvl "current", b.ksppstdf "isdefault"
        from x$ksppi a, x$ksppcv b
        where a.indx = b.indx
        and substr(a.ksppinm,1,1) = '_'
        order by a.ksppinm;
```

The isdefault column shows whether the current value for this parameter is the default (true or false). Note that similar information can be obtained from Oracle7, although that release only provides the x$ksppi view, and not the x$ksppcv view.

On the book website (http://www.solarisdatabases.com) I have included a script called _params that simplifies this process for Oracle8 and later releases. If run with no parameters, it displays all hidden parameters. If a string is passed to the script, it displays all hidden parameters matching the string.

Monitoring Error Messages

Oracle writes error, warning, and notification messages to the alert${ORACLE_SID}.log file, located in the $ORACLE_HOME/rdbms/log directory (unless an alternate directory has been specified with the BACKGROUND_DUMP_DEST parameter). This file is often referred to simply as alert.log. The alert log is a good first place to visit when trying to understand and resolve problems with an Oracle instance.

Using Oracle Enterprise Manager

Oracle Enterprise Manager (OEM) provides access to database monitoring and administration capabilities with an intuitive graphical user interface. OEM displays the buffer cache hit rate and many other important Oracle metrics.

Since the Oracle8.1.6 release, the OEM console runs on Solaris as well as on Windows platforms. To invoke the OEM console, run the oemapp command from the command line (after first ensuring that your DISPLAY environment variable is set appropriately):

```
  oracle% oemapp console &
```

It may first be necessary to run the Enterprise Manager Configuration Assistant program, emca, to create a repository.

Rather than considering OEM in any detail, in this chapter I focus on the lower-level data provided by scripts in the hope that such a focus will offer more insight into the underlying mechanisms used by Oracle.

Monitoring Oracle System Tables

Oracle maintains a number of internal views that record statistics about the database and offers scripts that present the same information in a more understandable fashion.

v$ Views

Oracle's internal views have names starting with v$. Although they appear to be tables, they are actually internal memory structures that are not persistent—that is, they only exist while the instance is active.

A few examples of v$ views are given in the following list:

- **v$system_event:** Shows a summary of all the events waited for in the instance since it started.
- **v$session_event:** Shows a summary of all the events the session has waited for since it started.
- **v$session_wait:** Shows the current waits for a session. This view is an important starting point for finding current bottlenecks.
- **v$sysstat:** Shows system statistics.
- **v$sesstat:** Shows system session statistics.
- **v$session:** Shows user-session-related information.
- **v$parameter:** Shows session parameters. To see the current parameter settings, try running the following command as sysdba:

  ```
  select name, value from v$parameter
  ```

  The **v$system_parameter** view shows systemwide parameters for the instance.
- **v$waitstat:** Shows buffer wait statistics (the number of times a user process had to wait for various buffers).
- **v$filestat:** Shows file access statistics.

We will encounter a number of other v$ views later in this chapter.

The utlbstat and utlestat Scripts

Although all the v$ views can be accessed with standard SQL statements, Oracle provides a simpler mechanism in the form of two scripts, utlbstat.sql and utlestat.sql. The first is run at the start of a measurement interval, and the second at the end of the measurement interval. The results are saved in a file called report.txt in the current directory. Many of the more important v$ views are represented in this report. Before running the scripts, make sure that Oracle is collecting timed statistics. If the timed_statistics parameter is set to false, you can change it dynamically as sysdba with the following command:

```
alter system set timed_statistics = true;
```

The parameter can be reset to false in the same way after the scripts have been run.

You can also permanently set the timed_statistics parameter to true in init.ora. The CPU overhead associated with timed statistics is small, and Oracle recommends setting the parameter permanently.

The way to run the utlbstat and utlestat scripts is shown below:

```
oracle$ sqlplus "/ as sysdba"
<< Various messages deleted >>
SQL> @$ORACLE_HOME/rdbms/admin/utlbstat
<< Pause for a suitable period of time... >>
<< Database activity during this period will be reported >>
SQL> @$ORACLE_HOME/rdbms/admin/utlestat
```

A sample `report.txt` for Oracle9i is presented later in this chapter, along with detailed comments and monitoring suggestions.

The Statspack Scripts

Oracle8.1.6 also introduced the `statspack` scripts. These scripts report information similar to that reported by the `utlbstat` and `utlestat` scripts, although more data is collected and some useful ratios are calculated for you. The `utlbstat`/`utlestat` scripts will eventually be phased out—this chapter focuses on their output rather than `statspack` output because they cover a broader range of releases.

For detailed information about installing and running the `statspack` scripts, refer to `$ORACLE_HOME/rdbms/admin/spdoc.txt` in the Oracle9i release and `$ORACLE_HOME/rdbms/admin/statspack.doc` in the Oracle8i release.

After installation (carried out with the `spcreate.sql` script for Oracle9i and with the `statscre.sql` script for Oracle8i), as `sysdba` you create snapshots in the following way:

```
SQL> connect perfstat/perfstat
SQL> execute statspack.snap;
```

To create a report, run the `spreport.sql` script (Oracle9i) or the `statsrep.sql` script (Oracle8i). The following example shows the appropriate syntax for Oracle9i.

```
SQL> @?/rdbms/admin/spreport
```

As this example illustrates, the `?` character can be used instead of `$ORACLE_HOME` within `sqlplus`.

This script prompts for the IDs of two previously created snapshots and, after prompting for a report file name, creates a report based on activity occurring between the two snapshots.

Generating Explain Plans

Before retrieving data in response to a query, the database optimizer determines how best to access the data. In practice, especially for DSS queries, there is often more than one path the optimizer can choose (for example, either to retrieve the data with an index or directly from the base table). The sequence of steps the optimizer chooses is referred to as a query execution plan, or explain plan (the role of the database optimizer is discussed in detail in Chapter 8).

In an ideal world, the optimizer would always choose the optimal plan. The real world is rarely so straightforward, unfortunately. So because the query plan is so important, especially to DSS performance, it is often necessary to provide the optimizer with hints that can be embedded in SQL statements.

As previously stated, it is beyond the scope of this book to cover application and SQL tuning. Nonetheless, it is sometimes useful to know how to generate an execution plan for an SQL statement.

As of Oracle 7.3, generating an explain plan from `sqlplus` is as easy as running the `set autotrace on` command before running the query. Note that if the `plan_table` has not already been created, you will need to run the `$ORACLE_HOME/rdbms/admin/utlxplan.sql` script first:

```
SQL> @?/rdbms/admin/utlxplan

Table created.
```

An example of `autotrace` is shown below.

```
SQL> set autotrace on
SQL> select scale, power, company
  2  from tpcd
  3  where company like '%Sun%'
  4  order by scale, power;

     SCALE      POWER COMPANY
---------- ---------- ------------------------------
        30      702.8 Sun
       100    13738.7 Sun
       300     2009.5 Sun
       300     3270.6 Sun
       300     8113.2 Sun
      1000     8870.6 Sun
      1000    12931.9 Sun
      1000    70343.7 Sun
      1000   121824.7 Sun

9 rows selected.


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (ORDER BY)
   2    1     TABLE ACCESS (FULL) OF 'TPCD'


Statistics
----------------------------------------------------------
        203  recursive calls
          4  db block gets
         58  consistent gets
         17  physical reads
         60  redo size
        894  bytes sent via SQL*Net to client
        715  bytes received via SQL*Net from client
          4  SQL*Net roundtrips to/from client
          5  sorts (memory)
          0  sorts (disk)
          9  rows processed
```

Explain plans can also be generated with the utlxplan script. This method, which also works with earlier versions of Oracle, is illustrated below. Once again, if the plan_table has not already been created, you will need to run the utlxplan.sql script, as shown.

```
SQL> @$ORACLE_HOME/rdbms/admin/utlxplan

Table created.

SQL> explain plan
  2  set Statement_ID = ’TEST’
  3  for
  4  select a.invoice_date
  5  from gl_je_lines a, gl_je_headers b
  6  where je_line_num = 1
  7  and a.je_header_id = b.je_header_id
  8  order by invoice_date desc;

Explained.

SQL> select
  2  LPAD(’ ’,2*Level)||
  3  Operation||’ ’||Options||’ ’||
  4  decode(Object_Owner,NULL,’’,
  5  Object_Owner||’. ’||Object_Name)||’ ’||
  6  decode(Optimizer,NULL,’’,Optimizer)
  7  Q_Plan
  8  from PLAN_TABLE
  9  connect by prior ID = Parent_ID and Statement_ID =’TEST’
 10  start with ID = 0 and Statement_ID = ’TEST’;

Q_PLAN
------------------------------------------------------------
  SELECT STATEMENT   RULE
    SORT ORDER BY
      NESTED LOOPS
        TABLE ACCESS FULL GL. GL_JE_HEADERS ANALYZED
        TABLE ACCESS BY ROWID GL. GL_JE_LINES ANALYZED
          INDEX UNIQUE SCAN GL. GL_JE_LINES_U1 ANALYZED

6 rows selected.
```

When using this method of printing explain plans, it is simplest to execute the following SQL command between explain plans:

```
delete from PLAN_TABLE where Statement_ID = ’TEST’;
```

Note that the first statement, which runs the utlxplan script, only needs to be run once (it creates the PLAN_TABLE table).

As of Oracle 7.2, the select statement above can be enhanced as follows:

```
select
LPAD(’ ’,2*Level)||
Operation||’ ’||Options||’ ’||
decode(Object_Owner,NULL,’’,
Object_Owner||’. ’||Object_Name)||’ ’||
decode(Optimizer,NULL,’’,Optimizer)||’ ’||
decode(Cost,NULL,’’,
        ’ Cost=’||Cost||
        ’ Rows Expected=’||Cardinality)
Q_Plan
from PLAN_TABLE
connect by prior ID = Parent_ID and Statement_ID = ’TEST’
start with ID = 0 and Statement_ID = ’TEST’;
```

Oracle9i introduced a new view—v$sql_plan—that provides access to the execution plans for recently executed cursors. The information provided is similar to that produced by an explain plan statement. Unlike the explain plan statement, which shows a theoretical plan, the v$sql_plan view shows the actual plan that was used.

Calculating the Buffer Cache Hit Rate

As we saw in Chapter 7, the buffer cache hit rate plays an important role in database performance, especially for OLTP workloads. The Oracle buffer cache is sized according to the db_block_buffers parameter in init.ora (or the db_cache_size parameter in Oracle9i).

The statspack report shows the buffer cache hit rate (under Buffer Hit Ratio for Oracle8.1.6, and under Buffer Hit % for later releases). The report.txt file produced by the utlbstat and utlestat scripts does not calculate the hit rate, although all the necessary information is there.

The Buffer Cache Hit Rate Formula

The buffer cache hit rate can be calculated from the variables listed below (not all of them are used for all Oracle releases):

- **physical reads:** The number of read requests that required a block to be read from disk.
- **physical reads direct**: The number of read requests that read a block from disk, bypassing the buffer cache. Reads carried out during parallel table scans, for example, bypass the buffer cache.
- **physical reads direct (LOB)**: The number of large-object (LOB) read requests that read a block from disk, bypassing the buffer cache.
- **db block gets:** Incremented when blocks are read for update and when segment header blocks are read.
- **consistent gets:** The number of times a consistent read was requested for a block.

  This statistic measures the number of block accesses involving System Change Number (SCN) checks. The SCN is a unique number assigned by Oracle to data file and block headers in ascending sequence to identify transaction modifications. It is checked to ensure that data is up-to-date. If the SCN for a row has changed since the transaction started, then the row must have been updated by another transaction; the before-image of the row will have been stored in a rollback segment.

The SCN is incremented as changes are made to data. Each row (and the block in which it is stored) holds a copy of the SCN that was current when the row was last changed. When the block is flushed to disk by the Database Writers, the SCN on disk will match the SCN for the same block in the buffer cache.

Oracle uses the SCN to ensure that data remains consistent and to assist in recovery after a crash. During roll-forward recovery, if the SCN for a block on disk is the same or later than the SCN in the redo log, there is no need to roll forward the transaction.

Figure 1 shows the formula for calculating the cache hit rate before the Oracle8i releases.

Figure 1 *Oracle Buffer Cache Hit Rate Formula before Oracle8i*

$$cachehitrate = \left(1 - \left(\frac{physicalreads}{(dbblockgets + consistentgets)}\right)\right) \times 100$$

Figure 2 shows the formula for calculating the cache hit rate for the Oracle8.1.5 and Oracle8.1.6 releases.

Figure 2 *Oracle Buffer Cache Hit Rate Formula for Oracle8.1.5/8.1.6*

$$cachehitrate = \left(1 - \left(\frac{physicalreads - physicalreadsdirect}{(dbblockgets + consistentgets - physicalreadsdirect)}\right)\right) \times 100$$

For Oracle8.1.7 and Oracle9i, use the formula shown in Figure 3 to calculate the buffer cache hit rate.

Figure 3 *Oracle Buffer Cache Hit Rate Formula for Oracle8.1.7/9i*

$$\left(1 - \left(\frac{physicalreads - physicalreadsdirect - physicalreadsdirectLOB}{(dbblockgets + consistentgets - physicalreadsdirect - physicalreadsdirectLOB)}\right)\right) \times 100$$

Cache Hit Rate Prediction

Oracle9i introduced a new view—`v$db_cache_advice`—to help with the challenging task of determining the optimal size for the buffer cache. Before this view can be used, the following statement must be executed:

```
alter system set db_cache_advice = on;
```

This statement will cause approximately 100 bytes to be allocated in the shared pool per buffer and will also result in a small CPU overhead. The shared pool memory can be preallocated by setting the db_cache_advice parameter to ready or on in init.ora before the database is started. The default value for db_cache_advice is off.

After a workload has been running for a time, the view can be queried. The collecting of statistics is terminated when the db_cache_advice parameter is set to off or to ready.

The v$db_cache_advice view reports the estimated number of physical reads that would have been required for 20 different buffer cache sizes, ranging from 10% of the current size to 200% of the current size. The information helps you to assess the likely impact on the I/O subsystem of either decreasing or increasing the size of the buffer cache.

Monitoring Oracle with utlbstat/utlestat

To illustrate the process of monitoring Oracle, we examine an Oracle9i report.txt file created with the utlbstat.sql and utlestat.sql scripts (described in “The utlbstat and utlestat Scripts” on page 7). The report is interspersed with comments about some of the highlights; my objective is to explore the main statistics that might require action rather than to attempt to explain every item.

Note that although we follow the order used by report.txt, the best way to begin understanding instance behavior is to examine wait events. The statspack scripts recognize this by reporting the top five wait events almost at the beginning of the report.

The Library Cache

The first section of the report deals with the library cache. The library cache stores SQL and PL/SQL statements for reuse by other applications (in the SQL AREA), and also caches other objects for Oracle’s internal use.

```
SQL> column library  format a12 trunc;
SQL> column pinhitratio   heading ‘PINHITRATI’;
SQL> column gethitratio   heading ‘GETHITRATI’;
SQL> column invalidations heading ‘INVALIDATI’;
SQL> set numwidth 10;
SQL> Rem Select Library cache statistics.The pin hit rate should be high.
SQL> select namespace library,
  2           gets,
  3           round(decode(gethits,0,1,gethits)/decode(gets,0,1,gets),3)
  4              gethitratio,
  5           pins,
  6           round(decode(pinhits,0,1,pinhits)/decode(pins,0,1,pins),3)
  7              pinhitratio,
  8           reloads, invalidations
  9    from stats$lib;
  LIBRARY          GETS GETHITRATI      PINS PINHITRATI  RELOADS INVALIDATI
  ------------ -------- ---------- --------- ---------- -------- ----------
  BODY             1172          1      1172       .999        1          0
  CLUSTER             0          1         0          1        0          0
  INDEX             818       .001       818       .001        0          0
  JAVA DATA           0          1         0          1        0          0
  JAVA RESOURC        0          1         0          1        0          0
  JAVA SOURCE         0          1         0          1        0          0
  OBJECT              0          1         0          1        0          0
  PIPE                0          1         0          1        0          0
  SQL AREA       517004       .986   2246870       .991     9115       4878
  TABLE/PROCED    26899       .974   1739839       .999     1254          0
  TRIGGER             0          1         0          1        0          0

  11 rows selected.
```

Gets measure the number of times Oracle set up a reference to objects in the cache, and pins measure the number of times objects were referenced. The gethitratio and pinhitratio should be as close to 1 as possible (at least .95), and reloads should be no more than 2% of gets. These elements cannot be individually tuned, but increasing the size of the shared pool (the shared_pool_size parameter in init.ora) can help improve the hit ratios.

User Connections

The next section of the report deals with database connections.

```
SQL> column “Statistic”       format a27 trunc;
SQL> column “Per Transaction” heading “Per Transact”;
SQL> column ((start_users+end_users)/2) heading “((START_USER”
SQL> set numwidth 12;
SQL> Rem The total is the total value of the statistic between the time
SQL> Rem bstat was run and the time estat was run.  Note that the estat
SQL> Rem script logs on to the instance so the per_logon statistics will
SQL> Rem always be based on at least one logon.
SQL> select ‘Users connected at ‘,to_char(start_time, ‘dd-mon-yy
hh24:mi:ss’),’:’,start_users from stats$dates;

‘USERSCONNECTEDAT’        TO_CHAR(START_TIME       ‘     START_USERS
------------------------  ---------------------    -     ------------
Users connected at        17-aug-01 10:01:30       :              41

SQL> select ‘Users connected at ‘,to_char(end_time, ‘dd-mon-yy
hh24:mi:ss’),’:’,end_users from stats$dates;

‘USERSCONNECTEDAT’        TO_CHAR(END_TIME,’       ‘       END_USERS
------------------------  ---------------------    -     -----------
Users connected at        17-aug-01 10:31:24       :              41

SQL> select ‘avg # of connections: ‘,((start_users+end_users)/2) from
stats$dates;

‘AVG#OFCONNECTIONS:’      ((START_USER
---------------------     ------------

avg # of connections:               41
```

The number of connections at the start and end of the monitoring period and the average number of connections all help track user connectivity. Note that connections do not necessarily equate to users, though, since some users may have more than one connection and administrative scripts (including the one used to create this report) also count as connections. Conversely, transaction monitors allow multiple users to share a single connection.

The duration of the monitoring period is also shown at the end of the report. In this case it was almost exactly 30 minutes.

Database Statistics

The statistics below include some of the most important measures to be monitored. If you examine the SQL command that generated these results, you will notice that only statistics with non-zero values are reported. Consequently, if you run `utlbstat/utlestat` again later, you might find that new rows appear in this section of the report and other rows may have disappeared.

```
SQL> select n1.name “Statistic”,
  2      n1.change “Total”,
  3      round(n1.change/trans.change,2) “Per Transaction”,
  4      round(n1.change/((start_users + end_users)/2),2)  “Per Logon”,
  5      round(n1.change/(to_number(to_char(end_time,   ‘J’))*60*60*24 -
  6       to_number(to_char(start_time, ‘J’))*60*60*24 +
  7       to_number(to_char(end_time,   ‘SSSSS’)) -
  8       to_number(to_char(start_time, ‘SSSSS’)))
  9    , 2) “Per Second”
 10  from
 11       stats$stats n1,
 12       stats$stats trans,
 13       stats$dates
 14  where
 15        trans.name=’user commits’
 16   and  n1.change != 0
 17  order by n1.name;


                                               Per
Statistic                          Total  Transact   Per Logon  Per Second
---------------------------- ---------  --------   ---------  ----------
CR blocks created                  11909       .04      290.46        6.64
Cached Commit SCN reference       158232       .49     3859.32        88.2
DBWR buffers scanned              590396      1.82     14399.9      329.09
DBWR checkpoint buffers wri       704570      2.18    17184.63      392.74
DBWR checkpoints                       2         0         .05           0
DBWR free buffers found           402083      1.24      9806.9      224.13
DBWR lru scans                     22320       .07      544.39       12.44
DBWR make free requests            22320       .07      544.39       12.44
DBWR summed scan depth            590396      1.82     14399.9      329.09
DBWR transaction table writ           68         0        1.66         .04
DBWR undo block writes             32991        .1      804.66       18.39
```

```
SQL*Net roundtrips to/from      421556       1.3    10281.85     234.98
background checkpoints comp          2         0         .05          0
background checkpoints star          2         0         .05          0
background timeouts               3947       .01       96.27        2.2
branch node splits                1210         0       29.51        .67
buffer is not pinned count     7280085     22.48   177563.05    4058.02
buffer is pinned count         1530511      4.73    37329.54     853.13
bytes received via SQL*Net   101119623    312.28  2466332.27   56365.45
bytes sent via SQL*Net to c  235602857    727.59  5746411.15  131328.24
calls to get snapshot scn:      501750      1.55     12237.8     279.68
calls to kcmgas                 408813      1.26     9971.05     227.88
calls to kcmgcs                  83029       .26      2025.1      46.28
cleanouts and rollbacks - c       8645       .03      210.85       4.82
cleanouts only - consistent       4922       .02      120.05       2.74
cluster key scan block gets    6431181     19.86   156858.07    3584.83
cluster key scans              6431132     19.86   156856.88     3584.8
commit cleanout failures: b          1         0         .02          0
commit cleanout failures: b         69         0        1.68        .04
commit cleanout failures: c        303         0        7.39        .17
commit cleanout failures: c        676         0       16.49        .38
commit cleanouts               4006646     12.37    97723.07    2233.36
commit cleanouts successful    4005597     12.37    97697.49    2232.77
consistent changes               11959       .04      291.68       6.67
consistent gets               10345798     31.95   252336.54    5766.89
consistent gets - examinati    2841078      8.77    69294.59    1583.66
cursor authentications              26         0         .63        .01
data blocks consistent read      11954       .04      291.56       6.66
db block changes              13297163     41.06   324321.05    7412.02
db block gets                  9020588     27.86   220014.34     5028.2
deferred (CURRENT) block cl    2208212      6.82    53858.83    1230.89
dirty buffers inspected          42360       .13     1033.17      23.61
enqueue releases                423982      1.31    10341.02     236.34
enqueue requests                423997      1.31    10341.39     236.34
enqueue waits                    11430       .04      278.78       6.37
execute count                  2275386      7.03    55497.22    1268.33
free buffer inspected            42398       .13      1034.1      23.63
free buffer requested           763898      2.36    18631.66     425.81
hot buffers moved to head o     831102      2.57    20270.78     463.27
immediate (CR) block cleano      13567       .04       330.9       7.56
immediate (CURRENT) block c     530423      1.64    12937.15     295.66
leaf node splits                 66072        .2     1611.51      36.83
logons cumulative                    2         0         .05          0
messages received               320324       .99     7812.78     178.55
messages sent                   320326       .99     7812.83     178.55
native hash arithmetic exec    4836175     14.94   117955.49    2695.75
no work - consistent read g    6989419     21.58   170473.63       3896
opened cursors cumulative          121         0        2.95        .07
parse count (failures)               1         0         .02          0
parse count (hard)                   5         0         .12          0
parse count (total)                120         0        2.93        .07
physical reads                  678283      2.09    16543.49     378.08
physical reads direct               94         0        2.29        .05
physical writes                 935315      2.89    22812.56     521.36
physical writes direct              94         0        2.29        .05
physical writes non checkpo     603082      1.86    14709.32     336.17
prefetched blocks                 1558         0          38        .87
```

```
recursive calls                    2034976      6.28     49633.56    1134.32
redo blocks written                4683822     14.46    114239.56    2610.83
redo buffer allocation retr             72         0         1.76        .04
redo entries                       6864072      21.2    167416.39    3826.13
redo log space requests                 72         0         1.76        .04
redo size                       2276348068   7029.82  55520684.59 1268867.37
redo synch writes                   328737      1.02      8017.98     183.24
redo wastage                      46160068    142.55   1125855.32   25730.25
redo writes                         186021       .57       4537.1     103.69
rollback changes - undo rec          39610       .12        966.1      22.08
rollbacks only - consistent           3257       .01        79.44       1.82
rows fetched via callback           173830       .54      4239.76       96.9
serializable aborts                  12052       .04       293.95       6.72
session logical reads             19366372     59.81    472350.54   10795.08
session pga memory                50759728    156.76   1238042.15   28294.16
session pga memory max            50653084    156.43   1235441.07   28234.72
session uga memory                38770192    119.73    945614.44   21611.03
session uga memory max            38828836    119.91    947044.78   21643.72
shared hash latch upgrades          450999      1.39     10999.98     251.39
shared hash latch upgrades               2         0          .05          0
sorts (disk)                           857         0        20.90        .48
sorts (memory)                       71791       .22         1751      40.02
sorts (rows)                       1924399      5.94     46936.56    1072.69
summed dirty queue length           263847       .81      6435.29     147.07
switch current to new buffe             11         0          .27        .01
table fetch by rowid                201420       .62      4912.68     112.27
table scan blocks gotten              3002       .01        73.22       1.67
table scan rows gotten                 948         0        23.12        .53
table scans (short tables)              12         0          .29        .01
transaction rollbacks                 4917       .02       119.93       2.74
user calls                          421017       1.3     10268.71     234.68
user commits                        323813         1      7897.88      180.5
user rollbacks                       13640       .04       332.68        7.6
write clones created in for            356         0         8.68         .2

100 rows selected.
```

Note that four sets of values are reported for each statistic:

- **Total.** This value shows the total number of events of this type during the monitoring interval.
- **Per Transaction.** This column is normalized according to the number of `user commits` (you will notice a value of 1 for that row). Note that the rate of `user commits` can provide an alternate measure of application workload in the absence of higher-level information about business transactions (such as the number of invoices processed during a specified period of time).
- **Per Logon.** Normalizing the statistic according to the number of logons (user connections) can help in predicting the impact of changing the number of users and user connections. Bear in mind, though, that logged-on connections may not all be active.

- **Per Second.** This value helps put the totals in perspective. For example, the I/O capability of a disk is usually expressed in I/Os per second, so knowing that 935,315 physical writes were completed is not as useful as knowing that on average 521 physical writes were completed per second. Early versions of Oracle did not include this useful column.

The Buffer Cache Hit Rate

Using the Oracle9i cache hit rate equation presented earlier in this chapter and the information reported above, we can calculate the cache hit rate for the monitoring interval:

$$\begin{aligned}&\textit{Buffer Cache Hit Rate}\\&\quad= (1 - ((678283 - 94 - 0) \div (9020588 + 10345798 - 94 - 0)))\\&\qquad\times 100\\&\quad= 96.5\%\end{aligned}$$

Given a cache hit rate of 96.5%, the miss rate is 3.5%—a miss rate that could probably be reduced.

Is the hit rate acceptable? Given the rate of physical reads (approximately 380 per second) and the rate of physical writes (approximately 520 per second), the average disk I/O rate is 900 I/Os per second. That load could probably be handled by fifteen 7200 rpm disks or twelve 10000 rpm disks, although it would be wise to configure up to 50% more disks to allow for peaks of I/O activity.

If 20 to 25 disks are in use for the database and the I/O is balanced evenly across all the disks, it may not be necessary to try to improve the cache hit rate. On the other hand, if fewer disks are in use and they are heavily utilized, reducing the miss rate might significantly improve performance, provided adequate memory is available for the purpose (never increase the size of the buffer cache so much that applications begin to page). Remember, too, that increasing the cache size will make little change to the rate of physical writes.

The issues related to monitoring the buffer cache hit rate and sizing the buffer cache are explored in more detail in Chapter 7 of *Configuring and Tuning Databases on the Solaris Platform*, beginning with “Monitoring the Buffer Cache” on page 76.

Other Statistics to Monitor

We conclude this section of the `report.txt` file by considering a few highlights from the long list of statistics reported by `utlbstat` and `utlestat`.

- **`dirty buffers inspected`.** This statistic measures the number of times a shadow process found a dirty buffer on the least recently used (LRU) list. Normally the Database Writers find such dirty buffers and move them to a linked list of dirty buffer headers. If the Database Writers are working effectively, this statistic should be zero (and therefore not appear in the report) or have a low value. Adding more Database Writers (the `db_writer_processes` parameter in `init.ora`) should help resolve a problem of this type. In this case, dirty buffers have been found 24 times a second, or on average for one in eight transactions, suggesting that the number of Database Writers could be usefully increased.

- **`redo log space requests.`** This statistic measures the number of times shadow processes stalled waiting for log file space. A common myth is that the statistic reports the number of times a process stalls during commits because there was not enough room in the log buffer. Stalls can occur during checkpoints.
- **`sorts (disk)`** and **`sorts (memory)`.** The first of these statistics measures the number of sorts that spill to the temporary tablespace because they could not fit in the memory allocated by the `sort_area_size` parameter in `init.ora`. The second statistic shows the number of sorts that were able to complete in memory without resorting to the temporary tablespace. The report above shows 857 sorts to disk over a 30-minute period compared to 71,791 sorts in memory. So just over 1% of all sorts spilled to disk, at a rate of less than one per minute. There is little reason to increase `sort_area_size` in this case.
- **`table scans (short tables)`** and **`table scans (long tables)`.** The first of these statistics shows the number of table scans carried out on short tables (less than or equal to 5 blocks in length) or on tables that have been flagged as cached.

  Tables can be specified as cached when the table is created, or later with the `alter table` command from `sqlplus` (for example, `alter table customer cache;`). Normally, blocks read during a full table scan are marked as least recently used, and the space they consume is quickly reclaimed. By contrast, blocks read from cached tables during a table scan are treated as most recently used blocks. Caching small, heavily accessed tables can improve performance in some cases.

  The second statistic shows the number of table scans on larger tables (none appeared in the report). Large table scans should be avoided in OLTP environments since they impact overall system performance and lower the buffer cache hit rate. Creating appropriate indexes or modifying the application can overcome the problem.

Systemwide Wait Events

You can dynamically view the events reported in the next section of the `report.txt` file by querying the `v$system_event` and `v$session_event` views as `sysdba`. The report below breaks down the wait events into two categories: nonbackground processes and background processes, where background processes are Oracle system processes like `PMON`, `SMON`, and `LGWR`. The wait events are sorted in descending order of total time spent waiting (in units of hundredths of seconds).

```
SQL> column “Event Name” format a32 trunc;
SQL> set numwidth 13;
SQL> Rem System wide wait events for non-background processes (PMON,
SQL> Rem SMON, etc).  Times are in hundredths of seconds.  Each one of
SQL> Rem these is a context switch which costs CPU time.  By looking at
SQL> Rem the Total Time you can often determine what is the bottleneck
SQL> Rem that processes are waiting for.  This shows the total time spent
SQL> Rem waiting for a specific event and the average time per wait on
SQL> Rem that event.
SQL> select  n1.event “Event Name”,
  2           n1.event_count “Count”,
  3           n1.time_waited “Total Time”,
  4           round(n1.time_waited/n1.event_count, 2) “Avg Time”
  5       from stats$event n1
  6       where n1.event_count > 0
  7       order by n1.time_waited desc;

Event Name                                Count    Total Time     Avg Time
-------------------------------- ------------- ------------- -------------
db file sequential read                  679244       2231864          3.29
SQL*Net message from client              423411       2164652          5.11
log file sync                            329826        516878          1.57
enqueue                                   11472         71489          6.23
latch free                                 1736          2194          1.26
db file parallel read                        40           481         12.03
log file switch completion                   53           393          7.42
SQL*Net message to client                423405           327             0
control file sequential read                 83           119          1.43
buffer busy waits                           379           113            .3
SQL*Net break/reset to client              2284            61           .03

11 rows selected.

SQL> Rem System wide wait events for background processes (PMON,SMON,etc)
SQL> select  n1.event “Event Name”,
  2           n1.event_count “Count”,
  3           n1.time_waited “Total Time”,
  4           round(n1.time_waited/n1.event_count, 2) “Avg Time”
  5       from stats$bck_event n1
  6       where n1.event_count > 0
  7       order by n1.time_waited desc;

Event Name                                Count    Total Time     Avg Time
-------------------------------- ------------- ------------- -------------
rdbms ipc message                        331631        779378          2.35
db file parallel write                   229188        603152          2.63
pmon timer                                  597        175435        293.86
smon timer                                    5        150002       30000.4
log file parallel write                  186846         80079           .43
db file scattered read                      377          2168          5.75
control file parallel write                 613          1021          1.67
latch free                                  155           248           1.6
LGWR wait for redo copy                     747           142           .19
control file sequential read                 96           101          1.05
async disk IO                               100            63           .63
db file sequential read                       7            10          1.43
log file single write                         4             4             1
direct path read                             94             2           .02
log file sequential read                      2             1            .5
direct path write                            94             0             0

16 rows selected.
```

The SQL*Net message from client wait event simply means that the shadow process is waiting for the client to do something. Consequently, substantial wait times for this event do not usually indicate a problem (unless the waits are due to network delays). The converse event, SQL*Net message to client, shows the delay when shadow processes send messages to clients; large delays could indicate network problems.

Wait events that should be monitored include those in the following list:

- **free buffer waits.** A lot of time spent waiting for free buffers suggests that the Database Writers are not flushing dirty buffers fast enough to keep up with demand. This event does not appear in the report above, but if it should appear as a major wait event, try increasing the db_writer_processes parameter in init.ora.
- **buffer busy waits.** Buffer busy waits occur when shadow processes were unable to access a buffer because it was in use by another process. The report above shows a tiny number of waits of this type.

  If buffer busy waits are one of the top wait events in terms of percentage of time waited, check the v$waitstat view to find out what type of blocks are affected (this information is also presented later in report.txt, in "Buffer Busy Wait Statistics" on page 27).

  You can also check the v$session_wait view to find out the file ID (the P1 column) and the block ID (the P2 column) of the affected block. The file ID can be used to query the dba_extents view (you will need to add a where file_id = *n* clause, where *n* is the file ID from the v$session_wait view) to get the details of the segment that the block ID falls within.

  Each data block supports a limited number of concurrent accesses for update or delete operations; a table with a large number of rows per block and high concurrency can experience frequent buffer busy wait events as a result. The INITRANS parameter determines the number of concurrent accesses (the default is 1 for tables and 2 for indexes). If the segment identified in v$session_wait belongs to a table or index, you could increase the INITRANS storage parameter. The INITRANS parameter can only be set during table or index creation, so it may be necessary to drop and recreate the table or index.

  For tables subject to high insert concurrency, increase the FREELISTS storage parameter to improve performance if the buffer busy wait events are related to inserts. The FREELISTS parameter also must be specified at create time.
- **enqueues.** Although this wait event appears in the report above, the number of events and wait time do not suggest a problem. If enqueue waits represent a high proportion of the time spent waiting, you can try to identify the enqueue waited for. Oracle9i provides a view—v$enqueue_stat—for this purpose; see statspack for more information before Oracle9i. Enqueue waits are a symptom of some other problem.

Latch Wait Events

Latch wait information can be obtained from the `v$latch` view. When a latch is not available, in some cases the requesting process may spin (that is, consume CPU) for a time before trying again, depending on the nature of the latch request. If the latch is still unavailable, the process will go to sleep and try again when it wakes up. The next section of `report.txt` deals with latches of this type. The same information can also be obtained from the `name`, `gets`, `misses`, and `sleeps` columns in the `v$latch` view.

The subsequent section deals with no-wait latches. Processes unable to acquire latches requested in this way do not sleep, but time out and retry immediately. The `immediate_gets` and `immediate_misses` columns in `v$latch` also provide this information.

When monitoring latches, check the hit ratio, which indicates the degree of contention on the latch. Check also the number of gets, which indicates how hot (that is, how much in demand) a latch is, and the number of sleeps, which indicates the number of times the process had to sleep while waiting for the latch.

The worst-case scenario with latches is that a process will be preempted by the operating system while holding a high-contention latch. Database performance for some Solaris systems (particularly midrange systems) improves if the CPU allocation available to processes is increased. This issue and its resolution are discussed in “The TS Class” on page 220 of *Configuring and Tuning Databases on the Solaris Platform*.

Latches with Waits

The first section of the latch report deals with latches with waits.

```
SQL> column latch_name format a18 trunc;
SQL> set numwidth 11;
SQL> Rem Latch statistics. Latch contention will show up as a large value
for
SQL> Rem the ‘latch free’ event in the wait events above.
SQL> Rem Sleeps should be low.The hit_ratio should be high.
SQL> select name latch_name, gets, misses,
  2   round((gets-misses)/decode(gets,0,1,gets),3)
  3     hit_ratio,
  4   sleeps,
  5   round(sleeps/decode(misses,0,1,misses),3) “SLEEPS/MISS”
  6  from stats$latches
  7   where gets != 0
  8   order by name;

LATCH_NAME                    GETS    MISSES  HIT-RATIO    SLEEPS  SLEEPS/MISS
----------------------- --------  --------  ----------  -------- -----------
FIB s.o chain latc             8         0           1         0           0
FOB s.o list latch            75         0           1         0           0
active checkpoint         135993        82        .999         4        .049
```

```
cache buffers chai      67102082       5408           1        131         .024
cache buffers lru        1211707        386           1          8         .021
channel handle poo             4          0           1          0            0
channel operations           582          0           1          0            0
checkpoint queue l       5377441       1310           1         92          .07
child cursor hash             12          0           1          0           0
dml lock allocatio           110          0           1          0            0
enqueue hash chain        859206       1146        .999        117         .102
enqueues                 1047133        820        .999         23         .028
event group latch              2          0           1          0            0
hash table column              4          0           1          0            0
ktm global data                5          0           1          0            0
latch wait list              958          0           1          0            0
library cache            4660957      12857        .997        465         .036
list of block allo        892019        301           1         44         .146
loader state objec             4          0           1          0            0
messages                 1158227        551           1         42         .076
multiblock read ob           834          0           1          0            0
ncodef allocation             29          0           1          0            0
post/wait queue la        655696       1722        .997        152         .088
process allocation             2          0           1          0            0
process group crea             4          0           1          0            0
redo allocation          7233318      20538        .997        374         .018
redo copy                    136         80        .412         93        1.163
redo writing             1166363       1394        .999        140           .1
row cache objects         121417          7           1          0            0
sequence cache                 3          0           1          0            0
session allocation        334950        295        .999          7         .024
session idle bit         1009326        512        .999         26         .051
session switching             29          0           1          0            0
shared pool                  224          0           1          0            0
sort extent pool              34          0           1          0            0
transaction alloca       1263687       2378        .998         77         .032
transaction branch            29          0           1          0            0
undo global data         1376947       2983        .998         77         .026
user lock                      4          0           1          0            0

39 rows selected.
```

No-Wait Latches

The remainder of this section of the report deals with no-wait latches. It is followed by suggestions on latch monitoring.

```
SQL> set numwidth 16
SQL> Rem Statistics on no_wait gets of latches. A no_wait get does not
SQL> Rem wait for the latch to become free, it immediately times out.
SQL> select name latch_name,
  2        immed_gets nowait_gets,
  3        immed_miss nowait_misses,
  4        round((immed_gets/(immed_gets+immed_miss)), 3)
  5          nowait_hit_ratio
  6       from stats$latches
  7        where immed_gets + immed_miss != 0
  8        order by name;

LATCH_NAME           NOWAIT_GETS    NOWAIT_MISSES NOWAIT_HIT_RATIO
------------------ ---------------- ---------------- ----------------
cache buffers chai           806427               38                1
cache buffers lru            763359             1090             .999
process allocation                2                0                1
redo copy                   6861266            55595             .992
```

The hit_ratio should be close to 1. The following list indicates the main latches to monitor:

- **cache buffers chains.** When searching for a block in the cache, a shadow process uses a hashing algorithm to find the appropriate hash bucket and then follows a hash chain to scan for the block. Fewer hash buckets means longer hash chains, more searching, and higher contention on the cache buffers chains latch. Inefficient SQL statements, such as heavily accessed statements using indexes that are not highly selective, can cause high contention for this latch.

  Identify the scale of any potential problem with the following SQL (for Oracle8 and later releases) as sysdba:

  ```
  select count(*) from x$bh;
  select dbarfil “File”, dbablk “Block”, count(*)
      from x$bh group by dbafil, dbablk
      having count(*) > 1;
  ```

- **cache buffers lru chains.** This latch protects the LRU chain. High contention could indicate that the Database Writers are not operating efficiently, for example, due to a slow or overloaded I/O subsystem.
- **library cache.** A number of factors contribute to high library cache latch contention. Sometimes you can alleviate the contention simply by increasing the size of the shared pool (the shared_pool_size parameter in init.ora). Other changes that might be necessary to relieve library cache contention include those in the following list:
  - Keep large SQL statements into the shared pool (with the dbms_shared_pool.keep procedure).
  - Use bind variables to reduce SQL statement parsing. For example, the following two statements are parsed and stored independently in the shared pool, even though they are almost identical:

    ```
    select cust_name from customer
        where cust_id = 12345;
    select cust_name from customer
        where cust_id = 23456;
    ```

    The preferred approach is to use a bind variable (for example, :cust_id) and to assign the value for cust_id to the bind variable. The two statements can then be consolidated into a single statement:

    ```
    select cust_name from customer
        where cust_id = :cust_id;
    ```

    Use of bind variables reduces latch contention and also reduces the pressure on free space in the shared pool.
  - Fully qualify object names. For example, use:

    ```
              select * from accts.customer;
    ```

rather than:

```
            select * from customer;
```

- Flush the shared pool if fragmentation occurs (run the `alter system flush shared_pool` command as `sysdba`). Fragmentation problems are typically accompanied by the `ORA-4031` error message: `More shared memory is needed than was allocated in the shared pool`. Note that flushing the shared pool provides short-term relief at the cost of a short-term performance hit but does not solve the problem. The previous suggestions should provide longer-term solutions.

- **`redo copy.`** If the hit ratio is low for the `redo copy` latch, it may be possible to reduce the contention by increasing the number of `redo copy` latches with the hidden `_log_simultaneous_copies` parameter in `init.ora`. Normally this parameter is based on the number of CPUs on the system. Do not change this parameter in Oracle9i. For more information on hidden parameters, including caveats, refer to "Viewing and Changing Hidden Parameters" on page 5.

The second half of Chapter 22 will appear in the August edition of Sun BluePrints Online. The chapter is extracted from *Configuring and Tuning Databases on the Solaris Platform*, by Allan N. Packer, (c) 2002, Sun Microsystems Press.