Oracle RAC Performance Tuning

Oracle's Real Application Clusters is the most robust and complex of all Oracle environments and the inter-instance communications makes RAC tuning especially challenging. This book excerpt (http://www.rampant-books.com/book_2004_1_10g_grid.htm) show expert tips and techniques used by real-world RAC professionals for tuning and optimizing even the most complex RAC system.

This is an excerpt from the bestselling Oracle 10g RAC book Oracle 10g Grid & Real Application Clusters (http://www.rampant-books.com/book_2004_1_10g_grid.htm), by Mike Ault and Madhu Tumma.

The focus of this chapter is on RAC tuning issues. Tuning a RAC database is very similar to tuning a non-RAC database; however, there are some major differences that will be covered.

This chapter will also cover the views used to monitor the RAC environment. Finally, the Oracle Enterprise Manager features that automate some of the DBA's RAC monitoring tasks will be introduced.

Analysis of Performance Issues

The analysis of performance issues in RAC involves several key areas:

- Normal database tuning and monitoring.
- Monitoring RAC cluster interconnect performance.
- Monitoring workloads.
- Monitoring RAC-specific contention.

Normal database monitoring is covered thoroughly in any number of other texts. Thus, aspects of database tuning such as SQL tuning or standard SGA and internals tuning are not covered in this text other than the required extensions to normal database monitoring.

Monitoring RAC Cluster Interconnect Performance

The most important aspects of RAC tuning are the monitoring and tuning of the global services directory processes. The processes in the Global Service Daemon (GSD) communicate through the cluster interconnects. If the cluster interconnects do not perform properly, the entire RAC structure will suffer no matter how well everything else is tuned. The major processes of concern are the Global Enqueue Services (GES) and Global Cache Services (GCS) processes.

The level of cluster interconnect performance can be determined by monitoring GCS waits that show how well data is being transferred. The waits that need to be monitored are shown in *v$session_wait*, *v$obj_stats*, and *v$enqueues_stats*. The major waits to be concerned with for RAC are:

- *global cache busy*
- *buffer busy global cache*
- *buffer busy global cr*

In later versions of Oracle, the global cache is shortened to just *gc*. To find the values for these waits, the *gv$session_wait* view is used to identify objects that have performance issues. The *gv$session_wait* view contents are shown in the following results:

```
Description of GV$SESSION_WAIT
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------
 INST_ID                                            NUMBER
 SID                                                NUMBER
 SEQ#                                               NUMBER
 EVENT                                              VARCHAR2(64)
 P1TEXT                                             VARCHAR2(64)
 P1                                                 NUMBER
 P1RAW                                              RAW(4)
 P2TEXT                                             VARCHAR2(64)
 P2                                                 NUMBER
 P2RAW                                              RAW(4)
 P3TEXT                                             VARCHAR2(64)
 P3                                                 NUMBER
 P3RAW                                              RAW(4)
 WAIT_CLASS#                                        NUMBER
 WAIT_CLASS                                         VARCHAR2(64)
 WAIT_TIME                                          NUMBER
 SECONDS_IN_WAIT                                    NUMBER
 STATE                                              VARCHAR2(19)
```

New in 10g is the *wait_class* column which is used to restrict returned values based on 12 basic wait classes, one of which is the *cluster wait class*.

The following wait events indicate that the remotely cached blocks were shipped to the local instance without having been busy, pinned or requiring a log flush and can safely be ignored:

- *gc current block 2-way*
- *gc current block 3-way*
- *gc cr block 2-way*
- *gc cr block 3-way*

However, the object level statistics for *gc current blocks received* and *gc cr blocks received* enable the rapid identification of the indexes and tables which are shared by the active instances.

The columns *p1* and *p2* identify the file and block number of any object experiencing the above waits for the events, as shown in the following queries:

```
SELECT
  INST_ID,
  EVENT,
  P1 FILE_NUMBER,
  P2 BLOCK_NUMBER,
  WAIT_TIME
FROM
  GV$SESSION_WAIT
WHERE
  EVENT IN ('buffer busy global cr', 'global cache busy',
            'buffer busy global cache');
```

The output from this query should resemble the following:

```
INST_ID EVENT                   FILE_NUMBER BLOCK_NUMBER  WAIT_TIME
------- ---------------------- ----------- ------------ ----------
      1 global cache busy                 9          150 0
      2 global cache busy                 9          150 0
```

In order to find the object that corresponds to a particular file and block, the following query can be issued for the first combination on the above list:

```
SELECT
  OWNER,
  SEGMENT_NAME,
  SEGMENT_TYPE
FROM
  DBA_EXTENTS
WHERE
  FILE_ID = 9
  AND 150 BETWEEN BLOCK_ID AND BLOCK_ID+BLOCKS-1;
```

SEE CODE DEPOT FOR COMPLETE SCRIPT (http://www.dba-oracle.com/bp/bp_elec_adv_mon_tuning.htm)

In this example, there is no need to worry about the instance as this SELECT is issued from within the cluster because they all see the same tables, indexes and objects.

The output will be similar to:

```
OWNER      SEGMENT_NAME                 SEGMENT_TYPE
---------- ---------------------------- ---------------
SYSTEM     MOD_TEST_IND                 INDEX
```

Once the objects causing the contention are determined, they should be modified by:

- Reducing the rows per block.
- Adjusting the block size.
- Modifying *initrans* and *freelists*.

All of these object modifications reduce the chances of application contention for the blocks in the object. *Index leaf* blocks are usually the most contended objects in the RAC environment; therefore, using a smaller block size for index objects can decrease intra-instance contention for *index leaf* blocks.

Contention in blocks can be measured by using the *block transfer time*. To determine *block transfer time*, examine the statistics g*lobal cache cr block receive time* and *global cache cr blocks received*. The time is determined by calculating the ratio of *global cache cr block receive* time to *global cache cr blocks received*. The values for these statistics are taken from the *gv$sysstat* view shown below:

```
Description of GV$SYSSTAT
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------
 INST_ID                                            NUMBER
 STATISTIC#                                         NUMBER
 NAME                                               VARCHAR2(64)
 CLASS                                              NUMBER
 VALUE                                              NUMBER
 HASH                                               NUMBER
```

The following script shows this calculation.

```
column "AVG RECEIVE TIME (ms)" format 9999999.9
col inst_id for 9999
prompt GCS CR BLOCKS
select b1.inst_id, b2.value "RECEIVED",
b1.value "RECEIVE TIME",
((b1.value / b2.value) * 10) "AVG RECEIVE TIME (ms)"
from gv$sysstat b1, gv$sysstat b2
where

SEE CODE DEPOT FOR COMPLETE SCRIPT (http://www.dba-
oracle.com/bp/bp_elec_adv_mon_tuning.htm)


INST_ID   RECEIVED RECEIVE TIME AVG RECEIVE TIME (ms)
------- ---------- ------------ --------------------
      1       2791         3287                 14.8
      2       3760         7482                 19.9
```

If the transfer time is too high, or if one of the nodes in the cluster shows excessive transfer times, the cluster interconnects should be checked using system level commands to verify that they are functioning correctly. In the above SELECT result, instance two exhibits an average receive time that is 69% higher than instance one.

Another useful SELECT for determining latency measures the overall latency, including that for queue, build, flush and send time. These statistics are also found in the *gv$sysstat* view. The SELECT is shown below:

```
SELECT
   a.inst_id "Instance",
   (a.value+b.value+c.value)/d.value "LMS Service Time"
FROM
  GV$SYSSTAT A,
```

```
  GV$SYSSTAT B,
  GV$SYSSTAT C,
  GV$SYSSTAT D
WHERE
  A.name = 'global cache cr block build time' AND
  B.name = 'global cache cr block flush time' AND
  C.name = 'global cache cr block send time' AND
  D.name = 'global cache cr blocks served' AND
  SEE CODE DEPOT FOR COMPLETE SCRIPT (http://www.dba-
oracle.com/bp/bp_elec_adv_mon_tuning.htm)


 Instance LMS Service Time
--------- ----------------
        1       1.07933923
        2       .636687318
```

Why does instance two show a lower service time than instance one? The node containing instance two is the slower node, while the node containing instance one serves blocks faster. The reason is that the slower node two serves blocks to node one, so node one shows a slower service time even though it is a faster node! Be aware of this in a two node cluster. Where there are multiple nodes, the service times should average out. If not, the aberrant node is probably troubled.

The following code can be used to examine the individual components of the service time to determine the source of the problem:

```
SELECT
   A.inst_id "Instance",
   (A.value/D.value) "Consistent Read Build",
   (B.value/D.value) "Log Flush Wait",
   (C.value/D.value) "Send Time"
FROM
  GV$SYSSTAT A,
  GV$SYSSTAT B,
  GV$SYSSTAT C,
  GV$SYSSTAT D
WHERE
  A.name = 'global cache cr block build time' AND
  B.name = 'global cache cr block flush time' AND
  C.name = 'global cache cr block send time' AND
  D.name = 'global cache cr blocks served' AND
  SEE CODE DEPOT FOR COMPLETE SCRIPT (http://www.dba-
oracle.com/bp/bp_elec_adv_mon_tuning.htm)


 Instance Consistent Read Build Log Flush Wait  Send Time
--------- -------------------- -------------- ----------
        1             .00737234     1.05059755  .02203942
        2             .04645529      .51214820  .07844674
```

If problems are detected, operating system specific commands should be used to pinpoint the node having difficulties. For example, in the SUN Solaris environment, commands such as the following can be used:

```
netstat -l
netstat -s
sar -c
sar -q
vmstat
```

These commands are used to monitor the cluster interconnects for:

- A large number of processes in the run queue waiting for CPU or scheduling delays.
- Platform specific operating system parameter settings that affect IPC buffering or process scheduling.
- Slow, busy or faulty interconnects. In these cases, look for dropped packets, retransmits, or cyclic redundancy check (CRC) errors. Ensure that the network is private and that inter-instance traffic is not routed through a public network.

The DBA will be interested in the waits and statistics for tuning shown in Table 14.1.

Statistic or Wait	Description	Source
buffer busy global cache	A wait event that is signaled when a process waits for a block to be available because another process is already obtaining a resource for this block.	*gv$system_event, gv$session_wait*
buffer busy waits	A wait event that is signaled when a process cannot get a buffer due to another process using the buffer at that point in time.	*gv$system_event, gv$session_wait*
cache convert waits	A statistic showing the total number of waits for all up-convert operations, these are: *global cache null to S*, *global cache null to X*, and *global cache S to X*.	*gv$sysstat*
cache open waits	A statistic that shows the total number of waits for *global cache open S* and *global cache open X*.	*gv$sysstat*
consistent gets	The consistent gets statistic shows the number of buffers obtained in consistent read (CR) mode.	*gv$sysstat*
cr request retry	A statistic that quantifies when Oracle resubmits a consistent read request due to detecting that the holding instance is no longer available.	*gv$sysstat*
db block changes	The number of current buffers that where obtained in exclusive mode for DML.	*gv$sysstat*
db block gets	The number of current buffers that were obtained for a read.	gv$sysstat

Statistic or Wait	Description	Source
DBWR cross-instance writes	These are also forced writes and show the number of writes that an instance had to perform to disk in order to make a previously exclusively held block available for another instance. Generally, DBWR cross-instance writes are eliminated due to Cache Fusion unless a value greater than 0 (zero) is set for the *gc_files_to_locks* parameter.	gv$sysstat
global cache bg acks	This wait event can only occur during the startup or shutdown of an instance as the LMS process completes its operations.	*gv$system_event, gv$session_wait*
global cache busy	This is a wait event that accumulates when a session has waits for an ongoing operation on the resource to complete.	*gv$system_event, gv$session_wait*
global cache cr cancel wait	This is a wait event that accumulates when a session waits for the completion of an AST for a canceled block access request. The canceling of the request is part of the Cache Fusion Write Protocol.	*gv$system_event, gv$session_wait*
global cache converts	This is a statistic that shows the resource converts for buffer cache blocks. Whenever GCS resources are converted from NULL to SHARED, NULL to EXCLUSIVE or SHARED to EXCLUSIVE this statistic is incremented.	*gv$sysstat*
global cache convert time	This statistics shows the accumulated time for global conversions on GCS resources for all sessions.	*gv$sysstat*
global cache convert timeouts	Whenever a resource operation times out this statistic is incremented.	*gv$sysstat*
global cache cr block flush time	This statistic shows the time waited for a log flush whenever a CR request is served. This is part of the serve time.	*gv$sysstat*
global cache cr blocks received	If a process requests a consistent read for a data block that is not in its local cache it has to send a request to another instance, this statistic is incremented once the buffer has been received.	*gv$sysstat*
global cache cr block receive time	This statistic records the accumulated round trip time for all requests for consistent read blocks.	*gv$sysstat*

Statistic or Wait	Description	Source
global cache cr blocks served	This statistic shows the number of requests for a consistent read block served by LMS. This statistic is incremented by Oracle when the block is sent.	*gv$sysstat*
global cache cr block build time	This statistic shows the accumulated time that the LMS process needs to create a consistent read block on the holding instance	*gv$sysstat*
global cache cr block send time	This statistic shows the time needed by LMS to begin a send of a consistent read block. For send time, timing does not stop until send has completed. Only the time it takes to initiate the send is measured; the time elapsed before the block arrives at the requesting instance is not included.	*gv$sysstat*
global cache cr cancel wait	This is a wait event that happens when a session waits for a canceled CR request to complete its acquisition interrupt. The process of canceling the CR request is part of the Cache Fusion Write Protocol.	*gv$system_event, gv$session_wait*
global cache cr request	This is a wait event that happens because a process is made to wait for a pending CR request to complete. The process must wait for either shared access to a block to be granted before it can read the block from disk, or it waits for the LMS of the holding instance to send the block.	*gv$system_event, gv$session_wait*
global cache current block flush time	This is a statistic that shows the time it takes to flush the changes from a block to disk before the block can be shipped to the requesting instance.	*gv$sysstat*
global cache current block pin time	This is a statistic that shows the time it takes to pin a current block before shipping it to a requesting instance. The pinning of a block prevents further changes to the block while it is prepared to be shipped.	*gv$sysstat*
global cache current blocks received	This is a statistic that shows the number of current blocks received over the interconnect from the holding instance.	*gv$sysstat*
global cache current block receive time	This is a statistic that shows the current blocks accumulated round trip time for all requests.	*gv$sysstat*

Statistic or Wait	Description	Source
global cache current block send time	This is a statistic that shows the time it takes to send the current block over the interconnect to the requesting instance.	*gv$sysstat*
global cache current blocks served	This is a statistic that shows the number of current blocks shipped over the interconnect to the requesting instance.	*gv$sysstat*
global cache freelist wait	This is a statistic that shows Oracle must wait after it detects that the local element free list is empty.	*gv$sysstat*
global cache freelist waits	This is a statistic that shows the number of times that the resource element free list was found empty by Oracle.	*gv$sysstat*
global cache gets	This is a statistic that shows the number of buffer gets that caused the opening of a new GCS resource.	*gv$sysstat*
global cache get time	This is a statistic that shows the accumulated time for all sessions that was needed to open a GCS resource for a local buffer.	*gv$sysstat*
global cache null to S	A wait event that occurs whenever a session has to wait for a resource conversion to complete.	*gv$system_event, gv$session_wait*
global cache null to X	This is a wait event that occurs whenever a session has to wait for this type of resource conversion to complete.	*gv$system_event, gv$session_wait*
global cache open S	This is a wait event that occurs when a session has to wait for receiving permission for shared access to the requested resource.	*gv$system_event, gv$session_wait*
global cache open X	This is a wait event that occurs when a session has to wait for receiving exclusive access to the requested resource.	*gv$system_event, gv$session_wait*
global cache S to X	This is a wait event that occurs whenever a session has to wait for this type of resource conversion to complete.	*gv$system_event, gv$session_wait*
global cache pending ast	This is a wait event that can occur when a process waits for an acquisition interrupt before Oracle closes a resource element.	*gv$system_event, gv$session_wait*

Statistic or Wait	Description	Source
global cache pred cancel wait	This is a wait event that occurs when a session must wait for the acquisition interrupt to complete for a canceled predecessor read request. The canceling of a predecessor read request is a part of the Cache Fusion Write Protocol.	*gv$system_event, gv$session_wait*
global cache retry prepare	This is a wait event that occurs whenever Oracle cannot ignore or skip a failure to prepare a buffer for a consistent read or Cache Fusion request.	*gv$system_event, gv$session_wait*
global lock async converts	This is a statistic showing the number of resource converts from an incompatible mode.	*gv$sysstat*
global lock sync gets	This is a statistic showing the number of synchronous GCS resources that Oracle must open. These sync gets are usually for GES resources. Library cache resources are one example.	*gv$sysstat*
global lock async gets	This is a statistic showing the number of Ansynchronous GES resources that Oracle must open. Normally, *async gets* include the number of *global cache gets* and are only used for GES resources.	*gv$sysstat*
global lock get time	This is a statistic that shows the accumulated open time for all GES resources.	*gv$sysstat*
global lock sync converts	This statistic shows the number of GES resources that Oracle converted from an incompatible mode. Usually *sync converts* occur for GES resources only.	*gv$sysstat*
global lock convert time	This is a statistic that shows the accumulated *global lock sync converts* and *global lock async converts* time.	*gv$sysstat*
lock buffers for read	This is a statistic that shows the number of NULL to SHARED up converts.	*gv$sysstat*
lock gets per transaction	This is a statistic that shows the number of *global lock sync gets* and *global lock async gets per transaction*.	*gv$sysstat*
lock converts per transaction	This is a statistic that shows the number of *global local sync converts* and *global lock async converts per transaction*.	*gv$sysstat*

Statistic or Wait	Description	Source
messages flow controlled	This shows the number of messages that are intended to be sent directly but that were instead queued and delivered later by LMD/LMS.	*gv$dml_misc*
messages received	This statistics shows the number of messages received by the LMD process.	*gv$dml_misc*
messages sent directly	This statistic shows the number of messages sent directly by Oracle processes.	*gv$dml_misc*
messages sent indirectly	This statistics shows the number of explicitly queued messages.	*gv$dml_misc*
physical reads	This is a statistic that shows the total number of disk reads performed because a request for a data block could not be satisfied from a local cache.	*gv$sysstat*
physical writes	This statistics shows the number of write I/Os performed by the DBWNn processes. This number includes the number of DBWR cross instance writes (forced writes) in Oracle Database 10g when *gc_files_to_locks* is set. Setting *gc_files_to_locks* for a particular datafile will enable the use of the old ping protocol and will not leverage the Cache Fusion architecture.	*gv$sysstat*
remote instance undo block writes	This is a statistic that shows the number of undo blocks written to disk by DBWn due to a forced write.	*gv$sysstat*
remote instance undo header writes	This is a statistic that shows the number of rollback segment header blocks written to disk by DBWn due to a forced write.	*gv$sysstat*

Table 14.1: *Waits and Statistics for RAC Monitoring from Oracle RAC Manual Glossary*

Oracle has provided the *racdiag.sql* script that can help troubleshoot RAC slowdowns; however, it has limited applicability to RAC system tuning. It is suggested that STATSPACK be used, combined with the selects provided in this chapter for RAC tuning.

racdiag.sql is available at metalink.oracle.com.

The *class* column in the *gv$sysstat* view indicates the type of statistic. Oracle RAC related statistics are in classes eight, 32, and 40.

Undesirable Global Cache Statistics

The following are undesirable statistics or statistics for which the values should always be as near to zero as possible:

- *global cache blocks lost*: This statistic shows block losses during transfers. High values indicate network problems. The use of an unreliable IPC protocol such as UDP may result in the value for *global cache blocks lost* being non-zero. When this occurs, take the ratio of *global cache blocks lost* divided by *global cache current blocks served* plus *global cache cr blocks served*. This ratio should be as small as possible. Many times, a non-zero value for *global cache blocks lost* does not indicate a problem because Oracle will retry the block transfer operation until it is successful.
- *global cache blocks corrupt*: This statistic shows if any blocks were corrupted during transfers. If high values are returned for this statistic, there is probably an IPC, network, or hardware problem.

An example SELECT to determine if further examination is needed would be:

```
SELECT
      A.VALUE "GC BLOCKS LOST 1",
      B.VALUE "GC BLOCKS CORRUPT 1",
      C.VALUE "GC BLOCKS LOST 2",
      D.VALUE "GC BLOCKS CORRUPT 2"
FROM GV$SYSSTAT A, GV$SYSSTAT B, GV$SYSSTAT C, GV$SYSSTAT D
WHERE A.INST_ID=1 AND A.NAME='gc blocks lost'
  AND B.INST_ID=1 AND B.NAME='gc blocks corrupt'
  AND C.INST_ID=2 AND C.NAME='gc blocks lost'
  AND D.INST_ID=2 AND D.NAME='gc blocks corrupt';
```

A sample result from the above select should look like the following:

```
GC BLOCKS LOST 1 GC BLOCKS CORRUPT 1 GC BLOCKS LOST 2 GC BLOCKS CORRUPT 2
---------------- ------------------ ---------------- ------------------
               0                  0              652                  0
```

In this result, instance 2 is showing some problems with lost blocks, so it might be useful to look at the ratio described above:

```
SELECT A.INST_ID "INSTANCE", A.VALUE "GC BLOCKS LOST",
B.VALUE "GC CUR BLOCKS SERVED",
C.VALUE "GC CR BLOCKS SERVED",
A.VALUE/(B.VALUE+C.VALUE) RATIO
FROM GV$SYSSTAT A, GV$SYSSTAT B, GV$SYSSTAT C
WHERE A.NAME='gc blocks lost' AND
      B.NAME='gc current blocks served' AND
      C.NAME='gc cr blocks served' and
      SEE CODE DEPOT FOR COMPLETE SCRIPT (http://www.dba-
oracle.com/bp/bp_elec_adv_mon_tuning.htm)
```

```
  Instance gc blocks lost gc cur blocks served gc cr blocks served      RATIO
---------- -------------- -------------------- -------------------- ----------
         1              0                 3923                 2734          0
         2            652                 3008                 4380 .088251218
```

The question now becomes, how small is "as small as possible"? In this example, database instance one takes 22 seconds to perform a series of tests and instance two takes 25 minutes.

Investigation showed that the TCP receive and send buffers on instance two were set at 64K. Since this is an 8k-block size instance with a *db_file_multiblock_read_count* of 16, this was causing excessive network traffic because the system was using full table scans resulting in a read of 128K. In addition, the actual TCP buffer area was set to a small number. Setting these values for the TCP receive and send buffers is an operating specific operation. DBA's should talk this over with the system operator or check out:

http://www.psc.edu/networking/perf_tune.html

The following section covers example commands for various operating systems.

DEC Alpha (Digital UNIX)

```
$ dbx -k /vmunix
(dbx) assign sb_max = (u_long) 524288
(dbx) patch sb_max = (u_long) 524288
```

The first command changes the value for the operating system and the second patches the value into the kernel so it will not be lost at the next reboot. *mbclusters* can be modified to at least 832, and the kernel variables *tcp_sendspace* and *tcp_recvspace* cam also be modified in this manner. Under version 4.0, use the *sysconfig -r inet <variable> <value>* command to do this type of modification.

HPUX (11)

Use the *ndd* command to view and modify the various TCP variables. Use the *ndd -h <parm_name>* to get the method for changing the values.

AIX

Use the "no" command.

Linux

To check the values you CAT the contents of the various */proc/sys/net* files:

```
$cat /proc/sys/net/core/rmem_default
```

The values of interest are:

```
rmem_default
rmem_max
```

```
wmem_default
wmem_max
tcp_rmem
tcp_wmem
```

The value under Linux can be set by echoing the proper values into the files in */proc/sys/net*. Some example values that have been found to be beneficial are:

```
echo '100 5000 640 2560 150 30000 5000 1884 2'>/proc/sys/vm/bdflush
hdparm -m16 -c1 -d1 -a8 /dev/hda
hdparm -m16 -c1 -d1 -a8 /dev/hdb
echo '131071'>/proc/sys/net/core/rmem_default
echo '262143'>/proc/sys/net/core/rmem_max
echo '131071'>/proc/sys/net/core/wmem_default
echo '262143'>/proc/sys/net/core/wmem_max
echo '4096 65536 4194304'>/proc/sys/net/ipv4/tcp_wmem
echo '4096 87380 4194304'>/proc/sys/net/ipv4/tcp_rmem
```

The call to */proc/sys/vm/bdflush* actually improves memory handling. The two *hdparm* commands improve IDE disk performances. The most interesting ones here are the ones echoing values into */proc/sys/net/core* and */proc/sys/net/ipv4* subdirectory files.

SUN (2.x)

SUN uses the ndd command:

```
$ndd -set /dev/tcp tcp_max_buff xxx
$ndd -set /dev/tcp tcp_xmit_hiwat xxx
$ndd -set /dev/tcp tcp_recv_hiwat xxx
```

Windows NT/2000

Use the various settings in the registry under the key:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
```

Additional detailed notes are available at:

http://rdweb.cns.vt.edu/public/notes/win2k-tcpip.htm

After increasing both the buffer size and buffer space for the TCP area, the following results were obtained with the query above after the same series of tests were repeated:

```
  INSTANCE GC BLOCKS LOST GC CUR BLOCKS SERVED GC CR BLOCKS SERVED     RATIO
---------- -------------- -------------------- -------------------- ----------
         1              0                 3892                 3642          0
         2              0                 3048                 2627          0
```

By correcting the TCP issue, performance increased such that the tests now required only three minutes 20 seconds to complete on the poor performing node.

This provides an 800% improvement at a minimum recalling that the poor node is a single CPU using an old technology motherboard and bus structures.

Monitoring *current* Blocks

In addition to *cr blocks*, RAC performance is also a concern when processing *current mode blocks*. *Current mode blocks* suffer from latency as well as build and wait time concerns similar to *cr blocks*. The average latency for a *current mode block* is calculated with the SELECT:

```
column "AVG RECEIVE TIME (ms)" format 9999999.9
col inst_id for 9999
prompt GCS CURRENT BLOCKS
select b1.inst_id, b2.value "RECEIVED",
b1.value "RECEIVE TIME",
((b1.value / b2.value) * 10) "AVG RECEIVE TIME (ms)"
from gv$sysstat b1, gv$sysstat b2
where b1.name = 'gc current block receive time' and
b2.name = 'gc current blocks received' and b1.inst_id = b2.inst_id;

INST_ID   RECEIVED RECEIVE TIME AVG RECEIVE TIME (ms)
------ ---------- ------------ --------------------
     1      22694        68999                 30.4
     2      23931        42090                 17.6
```

The service time for receiving a current block is calculated in a similar fashion to the value for a *cr block*, except there is a pin time instead of a build time:

```
SELECT
   a.inst_id "Instance",
   (a.value+b.value+c.value)/d.value "Current Blk Service Time"
FROM
  GV$SYSSTAT A,
  GV$SYSSTAT B,
  GV$SYSSTAT C,
  GV$SYSSTAT D
WHERE
  A.name = 'gc current block pin time' AND
  B.name = 'gc current block flush time' AND
  C.name = 'gc current block send time' AND
  D.name = 'gc current blocks served' AND
SEE CODE DEPOT FOR COMPLETE SCRIPT (http://www.dba-
oracle.com/bp/bp_elec_adv_mon_tuning.htm)


 Instance Current Blk Service Time
--------- ------------------------
        1              1.18461603
        2              1.63126376
```

Instance two is requiring more time to service current blocks. How is the source of the problem determined? The overall service time can be decomposed to determine where the area of concern lies:

```
SELECT
   A.inst_id "Instance",
   (A.value/D.value) "Current Block Pin",
   (B.value/D.value) "Log Flush Wait",
   (C.value/D.value) "Send Time"
FROM
```

```
  GV$SYSSTAT A,
  GV$SYSSTAT B,
  GV$SYSSTAT C,
  GV$SYSSTAT D
WHERE
  A.name = 'gc current block build time' AND
  B.name = 'gc current block flush time' AND
  C.name = 'gc current block send time' AND
  D.name = 'gc current blocks served' AND
```

SEE CODE DEPOT FOR COMPLETE SCRIPT (http://www.dba-oracle.com/bp/bp_elec_adv_mon_tuning.htm)

```
 Instance Current Block Pin Log Flush Wait  Send Time
--------- ----------------- -------------- ----------
        1         .69366887     .472058762 .018196236
        2        1.07740715     .480549199 .072346418
```

In this case, most of the time difference comes from the pin time for the current block in instance two. High pin times could indicate problems at the I/O interface level.

A final set of statistics deals with the *average global cache convert time* and the *average global cache get times*. The following SELECT can be used to get this information from the RAC database:

```
select
  a.inst_id "Instance",
  a.value/b.value "Avg Cache Conv. Time",
  c.value/d.value "Avg Cache Get Time",
  e.value "GC Convert Timeouts"
from
   GV$SYSSTAT A,
   GV$SYSSTAT B,
   GV$SYSSTAT C,
   GV$SYSSTAT D,
   GV$SYSSTAT E
where
  a.name='gc convert time' and
  b.name='gc converts' and
  c.name='gc get time' and
  d.name='gc gets' and
  e.name='gc convert timeouts' and
  b.inst_id=a.inst_id and
  c.inst_id=a.inst_id and
  d.inst_id=a.inst_id and
  e.inst_id=a.inst_id
order by
  a.inst_id;
```

SEE CODE DEPOT FOR COMPLETE SCRIPT (http://www.dba-oracle.com/bp/bp_elec_adv_mon_tuning.htm)

```
 Instance Avg Cache Conv. Time Avg Cache Get Time GC Convert Timeouts
--------- ------------------- ------------------ -------------------
        1          1.85812072         .981296356                   0
        2          1.65947528         .627444273                   0
```

For this database, instance one has the highest convert and get times, as expected, since it is converting and getting from instance two, which is the slow instance. None of the times are excessive, >10-20 ms.

Some things to consider about these values are:

- High convert times indicate excessive global concurrency requirements. In other words, the instances are swapping a lot of blocks over the interconnect.
- Large values or rapid increases in the gets, converts, or average times indicate GCS contention.
- Latencies for resource operations may be high due to excessive system loads.
- The *gv$system_event* view can be used to review the *time_waited* statistics for various GCS events if the get or convert times become significant. STATSPACK is good for this.
- Values other than zero for the GC converts timeouts indicate system contention or congestion. Timeouts should not occur and indicate a serious performance problem.

Additional Wait Events of Concern

Most of the events reported in the dynamic performance views or in a STATSPACK report that show a high total time are actually normal. However, if monitored response times increase and the STATSPACK report shows a high proportion of wait time for cluster accesses, the cause of these waits needs to be determined. STATSPACK reports provide a breakdown of the wait events, with the five highest values sorted by percentages. The following specific RAC-related events should be monitored:

- *global cache open s*: A block was selected.
- *global cache open x*: A block was selected for IUD.
- *global cache null to s*: A block was transferred for SELECT.
- *global cache null to x*: A block was transferred for IUD.
- *global cache cr request*: A block was requested transferred for consistent read purposes.
- Global Cache Service Utilization for Logical Reads.

The following sections will provide more information on these events to help show why they are important to monitor.

The *global cache open s* and *global cache open x* Events

The initial access of a particular data block by an instance generates these events. The duration of the wait should be short, and the completion of the wait is most likely followed by a read from disk. This wait is a result of the blocks that are being requested and not being cached in any instance in the cluster database. This necessitates a disk read.

When these events are associated with high totals or high per-transaction wait times, it is likely that data blocks are not cached in the local instance and that the blocks cannot be obtained from another instance, which results in a disk read. At the same time, suboptimal *buffer cache hit ratios* may also be observed. Unfortunately, other than preloading heavily used tables into the buffer caches, there is little that can be done about this type of wait event.

The *global cache null to s* and *global cache null to x* Events

These events are generated by inter-instance block ping across the network. Inter-instance block ping is when two instances exchange the same block back and forth. Processes waiting for *global cache null to s* events are waiting for a block to be transferred from the instance that last changed it. When one instance repeatedly requests cached data blocks from the other RAC instances, these events consume a greater proportion of the total wait time. The only method for reducing these events is to reduce the number of rows per block to eliminate the need for block swapping between two instances in the RAC cluster.

The *global cache cr request* Event

This event is generated when an instance has requested a consistent read data block and the block to be transferred had not arrived at the requesting instance. Other than examining the cluster interconnects for possible problems, there is nothing that can be done about this event other than to modify objects to reduce the possibility of contention.

Global Cache Service Times

When global cache waits constitute a large proportion of the wait time, as listed on the first page of the STATSPACK or AWRRPT report, and if response time or throughput does not conform to service level requirements, the Global Cache Service workload characteristics on the cluster statistics page of the STATSPACK or AWRRPT reports should be examined. The STATSPACK or AWRRPT reports should be taken during heavy RAC workloads.

If the STATSPACK report shows that the average GCS time per request is high, it is the result of one of the following:

- Contention for blocks.
- System loads.
- Network issues.

The operating system logs and operating system statistics are used to indicate whether a network link is congested. A network link can be congested if:

- Packets are being routed through the public network instead of the private interconnect.
- The sizes of the run queues are increasing.

If CPU usage is maxed out and processes are queuing for the CPU, the priority of the GCS processes (LMSn) can be raised over other processes to lower GCS times. The load on the server can also be alleviated by reducing the number of processes on the database server, increasing capacity by adding CPUs to the server, or adding nodes to the cluster database.

New 10g RAC Waits

This section covers the most important Oracle Database 10g wait events to be aware of when analyzing performance reports. When user response times increase and a high proportion of time waited is due to the *global cache* (*gc*), the cause must be determined and corrected.

It is best to start with an ADDM report if that option has been purchased. This will analyze the routinely collected performance statistics with respect to their impact pinpoint the objects and SQL contributing most to the time waited. After that, the analysis can move on to the more detailed reports produced by the AWR and STATSPACK utilities.

The most important wait events for RAC fall into four main categories, arranged alphabetically these are:

- Block-oriented waits:
  - *gc current block 2-way*
  - *gc current block 3-way*
  - *gc cr block 2-way*
  - *gc cr block 3-way*
- Contention-oriented waits:
  - *gc current block busy*
  - *gc cr block busy*
  - *gc current buffer busy*
- **Load-oriented waits:**
  - *gc current block congested*
  - *gc cr block congested*
- **Message-oriented waits:**

- *gc current grant 2-way*
- *gc cr grant 2-way*

Block Oriented Wait Events

As their names imply, these block related wait events are the result of some combination of message and transfer event. Whether the wait occurred between a resource master and a requesting node as a single message and block transfer or was the result of a message passed from a node several times removed from the resource master causing two messages and a block transfer, the result is the same. A wait event of some duration occurred.

These wait events consist of several components:

- Physical network latency
- Request processing time
- Requesting process sleep time

By looking at the total and average wait times associated with these events, the DBA can be alerted when performance is being affected in a negative manner, such as when average times increase beyond statistical norms, something is wrong and needs investigation. Usually, the impact of interconnect malfunctions or load issues or the interactions of nodes against hot blocks are the culprit when these waits become significant.

Message Related Wait Events

The wait events dealing with messages usually indicate that the block requested was not present in any of the caches for the various nodes. This results in the need to issue a global grant and allow the requesting instance to read the block from disk in order to modify it. When messaging events result in high waits, it is usually due to frequently accessed SQL causing disk IO, due to cache areas being too small, in the event of *cr* grants, or the instances are inserting a lot of new records requiring format of disk blocks, if current grants are the cause.

Contention-Oriented Wait Events

Contention-oriented wait events are caused by:

- Waits for blocks pinned by other nodes
- Waits for changes to be flushed to disk
- Waits due to high concurrency for block access
- Intra-node cache-fusion contention

If high service times are experienced in the global cache, this type of wait can get worse. In general, this type of wait results from multiple nodes requiring read or write of the same block of data or index entries.

Load-Oriented Wait Events

These types of waits indicate that delays have occurred within the global cache services (GCS). Load-oriented events are usually caused by high loads on the nodes and their CPUs. This type of event is solved by:

- Increasing the available CPUs
- Load balancing
- Offloading processing until a lower load is available
- Adding a new node

For load-oriented events, the time is the total time for the round-trip, beginning when the request is made and ending when the block is received.

The following section will cover the STATSPACK report sections that are useful when dealing with RAC. The report is only in relation to the node/instance on which it was run. To compare other instances, the *statspack.snap* procedure and *spreport.sql* report script can be run on each node/instance that is to be monitored.

RAC and STATSPACK

The following is a STATSPACK report for the troubled instance. The sections covered will be those that involve RAC statistics. The first section deals with the top five timed events:

```
Top 5 Timed Events
~~~~~~~~~~~~~~~~~~
                                                     % Total
Event                                          Waits    Time (s) Ela Time
---------------------------------------------- ------------ ----------- --------
global cache cr request                              820         154    72.50
CPU time                                                          54    25.34
global cache null to x                               478           1      .52
control file sequential read                         600           1      .52
control file parallel write                          141           1      .28
          -------------------------------------------------------------
```

Observe the events in the report that are taking a majority of the *% total elapsed time* column that are greater than or near the *%total ela time* value for *cpu time*. The *cpu time* statistic should be the predominant event as it denotes processing time. If *cpu time* is not the predominant event, the events that exceed *cpu time's* share of the total elapsed time need to be investigated. In the above report section, *global cache cr request* events are dominating the report. This indicates that transfer times are excessive from the other instances in the cluster to this instance. The excessive transfer times could be due to network problems or buffer cache sizing issues.

After making the network changes and adding an index, the STATSPACK wait report for instance one looks like:

```
Top 5 Timed Events
~~~~~~~~~~~~~~~~~~                                                  % Total
Event                                             Waits    Time (s) Ela Time
--------------------------------------------- ------------ ----------- --------
CPU time                                                          99    64.87
global cache null to x                             1,655          28    18.43
enqueue                                               46           8     5.12
global cache busy                                    104           7     4.73
DFS lock handle                                       38           2     1.64
```

The number one wait is now *cpu time*, followed by *global cache null to x*, which indicates the major wait has been shifted from intra-cache to I/O-based as *global cache null to x* indicates a read from disk.

The next report in the STATSPACK listing shows the workload characteristics for the instance for which the report was generated:

```
Cluster Statistics for DB: MIKE  Instance: mike2  Snaps: 25 -26

Global Cache Service - Workload Characteristics
-----------------------------------------------
Ave global cache get time (ms):                          3.1
Ave global cache convert time (ms):                      3.2

Ave build time for CR block (ms):                        0.2
Ave flush time for CR block (ms):                        0.0
Ave send time for CR block (ms):                         1.0
Ave time to process CR block request (ms):               1.3
Ave receive time for CR block (ms):                     17.2

Ave pin time for current block (ms):                     0.2
Ave flush time for current block (ms):                   0.0
Ave send time for current block (ms):                    0.9
Ave time to process current block request (ms):          1.1
Ave receive time for current block (ms):                 3.1

Global cache hit ratio:                                  1.7
Ratio of current block defers:                           0.0
% of messages sent for buffer gets:                      1.4
% of remote buffer gets:                                 1.1
Ratio of I/O for coherence:                              8.7
Ratio of local vs remote work:                           0.6
Ratio of fusion vs physical writes:                      1.0
```

In the above report, the statistics in relation to the other instances in the cluster should be examined. The possible causes of any statistics that are not in line with the other cluster instances should be investigated. By making the network changes and index changes stated before, the workload was increased by a factor of greater than three, and the response time was still less than in the original STATSPACK. The following is the same section from the STATSPACK report taken after the network changes. Almost all statistics show an increase:

```
Cluster Statistics for DB: MIKE  Instance: mike2  Snaps: 105 -106

Global Cache Service - Workload Characteristics
-----------------------------------------------
Ave global cache get time (ms):                          8.2
```

```
Ave global cache convert time (ms):                      16.5

Ave build time for CR block (ms):                         1.5
Ave flush time for CR block (ms):                         6.0
Ave send time for CR block (ms):                          0.9
Ave time to process CR block request (ms):                8.5
Ave receive time for CR block (ms):                      18.3

Ave pin time for current block (ms):                     13.7
Ave flush time for current block (ms):                    3.9
Ave send time for current block (ms):                     0.8
Ave time to process current block request (ms):          18.4
Ave receive time for current block (ms):                 17.4

Global cache hit ratio:                                   2.5
Ratio of current block defers:                            0.2
% of messages sent for buffer gets:                       2.2
% of remote buffer gets:                                  1.6
Ratio of I/O for coherence:                               2.8
Ratio of local vs remote work:                            0.5
Ratio of fusion vs physical writes:                       0.0
```

The next report shows the global enqueue service statistics. The global enqueue services (GES) control the inter-instance locks in Oracle Database 10g RAC. These times should all be in the less than 15 millisecond range and the ratio should be near one. If they are not, it shows possible network or memory problems. Application locking issues can also cause this and the enqueue report, which is shown later in the STATSPACK listing, should be consulted to further diagnose the possible problems:

```
Global Enqueue Service Statistics
---------------------------------
Ave global lock get time (ms):                            0.9
Ave global lock convert time (ms):                        1.3
Ratio of global lock gets vs global lock releases:        1.1
```

The next STATSPACK report deals with GCS and GES messaging. Queue times greater than 20-30 ms should be considered excessive and should be watched:

```
GCS and GES Messaging statistics
--------------------------------
Ave message sent queue time (ms):                         1.8
Ave message sent queue time on ksxp (ms):                 2.6
Ave message received queue time (ms):                     1.2
Ave GCS message process time (ms):                        1.2
Ave GES message process time (ms):                        0.2
% of direct sent messages:                               58.4
% of indirect sent messages:                              4.9
% of flow controlled messages:                           36.7
-------------------------------------------------------------
```

The next section of the STATSPACK report also deals with the global enqueue services statistics. Blocked converts are one thing to look for. There are several in this report example. Blocked converts means the instance requested a block from another instance and for one reason or another was unable to obtain the conversion of the block. This can indicate that users are going after the same records, and it may be desirable to prevent that from occurring.

Blocked converts can also indicate insufficient freelists. This should not be the issue if Oracle Database 10g bitmap freelists are used. Block contention can be reduced through *freelists, ini_trans*, and limiting rows per block to avoid conversions being blocked.

If there are excessive message processing times, thought should be given to tuning the network to increase bandwidth, or perhaps upgrading the NIC's to faster, high bandwidth versions.

```
GES Statistics for DB: MIKE  Instance: mike2  Snaps: 25 -26

Statistic                                Total   per Second    per Trans
--------------------------------- ---------------- ------------ ------------
dynamically allocated gcs resourc                0          0.0          0.0
dynamically allocated gcs shadows                0          0.0          0.0
flow control messages received                   0          0.0          0.0
flow control messages sent                       0          0.0          0.0
gcs ast xid                                      0          0.0          0.0
gcs blocked converts                           904          2.1        150.7
gcs blocked cr converts                      1,284          2.9        214.0
gcs compatible basts                             0          0.0          0.0
gcs compatible cr basts (global)                 0          0.0          0.0
gcs compatible cr basts (local)                 75          0.2         12.5
gcs cr basts to PIs                              0          0.0          0.0
gcs cr serve without current lock                0          0.0          0.0
gcs error msgs                                   0          0.0          0.0
gcs flush pi msgs                               21          0.0          3.5
gcs forward cr to pinged instance                0          0.0          0.0
gcs immediate (compatible) conver                4          0.0          0.7
gcs immediate (null) converts                   79          0.2         13.2
gcs immediate cr (compatible) con                4          0.0          0.7
gcs immediate cr (null) converts                 3          0.0          0.5
gcs msgs process time(ms)                    3,193          7.3        532.2
gcs msgs received                            2,586          5.9        431.0
gcs out-of-order msgs                            0          0.0          0.0
gcs pings refused                                0          0.0          0.0
gcs queued converts                              0          0.0          0.0
gcs recovery claim msgs                          0          0.0          0.0
gcs refuse xid                                   0          0.0          0.0
gcs retry convert request                        0          0.0          0.0
gcs side channel msgs actual                    65          0.1         10.8
gcs side channel msgs logical                1,383          3.2        230.5
gcs write notification msgs                      0          0.0          0.0
gcs write request msgs                           0          0.0          0.0
gcs writes refused                               0          0.0          0.0
ges msgs process time(ms)                      136          0.3         22.7
ges msgs received                              578          1.3         96.3
implicit batch messages received                90          0.2         15.0
implicit batch messages sent                    12          0.0          2.0
lmd msg send time(ms)                           55          0.1          9.2
lms(s) msg send time(ms)                         8          0.0          1.3
messages flow controlled                       806          1.8        134.3
messages received actual                     1,980          4.5        330.0
messages received logical                    3,164          7.3        527.3
messages sent directly                       1,281          2.9        213.5
messages sent indirectly                       108          0.2         18.0
msgs causing lmd to send msgs                  231          0.5         38.5
msgs causing lms(s) to send msgs                97          0.2         16.2
msgs received queue time (ms)                3,842          8.8        640.3
msgs received queued                         3,164          7.3        527.3
msgs sent queue time (ms)                      202          0.5         33.7
msgs sent queue time on ksxp (ms)            4,337          9.9        722.8
msgs sent queued                               111          0.3         18.5
msgs sent queued on ksxp                     1,658          3.8        276.3
process batch messages received                269          0.6         44.8
process batch messages sent                    191          0.4         31.8
          -------------------------------------------------------------
```

The next section of the report deals with waits. Waits are a key tuning indicator. The predominant wait is for *global cache cr request*, which was caused by the network not being tuned properly, as already mentioned. The second highest wait is the *global cache null to x*, which, if severe, indicates problems with the I/O subsystem. In this case, the total time waited was one second or less, hardly a concern when compared with the 154 second wait on *global cache cr request*. The waits with the highest total time should be tuned first. Waits such as SQL*Net waits and any

having to do with *smon*, *pmon*, or *wakeup* timers can be safely ignored in most situations.

```
Wait Events for DB: MIKE  Instance: mike2  Snaps: 25 -26
-> s  - second
-> cs - centisecond -     100th of a second
-> ms - millisecond -    1000th of a second
-> us - microsecond - 1000000th of a second
-> ordered by wait time desc, waits desc (idle events last)

Avg
                                                        Total Wait   wait    Waits
Event                               Waits   Timeouts   Time (s)   (ms)     /txn
---------------------------- ------------ ---------- ---------- ------ --------
global cache cr request              820        113        154    188    136.7
global cache null to x               478          1          1      2     79.7
control file sequential read         600          0          1      2    100.0
control file parallel write          141          0          1      4     23.5
enqueue                               29          0          1     18      4.8
library cache lock                   215          0          0      2     35.8
db file sequential read               28          0          0      7      4.7
LGWR wait for redo copy               31         16          0      4      5.2
ksxr poll remote instances           697        465          0      0    116.2
global cache open x                   48          0          0      2      8.0
CGS wait for IPC msg                 899        899          0      0    149.8
log file parallel write              698        697          0      0    116.3
latch free                            24          0          0      2      4.0
global cache s to x                   41          0          0      1      6.8
log file sync                          3          0          0     13      0.5
DFS lock handle                       30          0          0      1      5.0
global cache open s                   16          0          0      1      2.7
global cache null to s                 9          0          0      1      1.5
library cache pin                    133          0          0      0     22.2
KJC: Wait for msg sends to c          17          0          0      0      2.8
db file parallel write                19          0          0      0      3.2
cr request retry                      27         27          0      0      4.5
gcs remote message                10,765      9,354        840     78  1,794.2
virtual circuit status                15         15        440  29307      2.5
ges remote message                 9,262      8,501        421     45  1,543.7
wakeup time manager                   14         14        408  29135      2.3
SQL*Net message from client        4,040          0        216     53    673.3
SQL*Net message to client          4,040          0          0      0    673.3
          -------------------------------------------------------------
```

The next report deals with enqueues. These are the normal system enqueues. The non-RAC related ones have been removed from this listing. Enqueues are high level locks used to protect memory areas. When the report is reviewed, the enqueues with the highest totals should be of the most interest. In the following report, all of the enqueues of concern are again dealing with message times and cache block transfers. These types of enqueues again point to network tuning.

```
Enqueue activity for DB: MIKE  Instance: mike2  Snaps: 25 -26
-> Enqueue stats gathered prior to 10g should not be compared with 10g data
-> ordered by Wait Time desc, Waits desc

Instance Activity Stats for DB: MIKE  Instance: mike2  Snaps: 25 -26

Statistic                                     Total     per Second    per Trans
--------------------------------- ------------------ -------------- ------------
gcs messages sent                             1,570            3.6        261.7
ges messages sent                               805            1.9        134.2
global cache blocks lost                         88            0.2         14.7
global cache convert time                       171            0.4         28.5
global cache converts                           528            1.2         88.0
global cache cr block build time                 28            0.1          4.7
global cache cr block flush time                  2            0.0          0.3
global cache cr block receive tim             1,158            2.7        193.0
global cache cr block send time                 117            0.3         19.5
global cache cr blocks received                 674            1.6        112.3
global cache cr blocks served                 1,147            2.6        191.2
global cache current block pin ti                12            0.0          2.0
global cache current block receiv               170            0.4         28.3
global cache current block send t                57            0.1          9.5
global cache current blocks recei               541            1.2         90.2
global cache current blocks serve               653            1.5        108.8
global cache defers                               0            0.0          0.0
global cache get time                            57            0.1          9.5
global cache gets                               183            0.4         30.5
global cache skip prepare failure                37            0.1          6.2
global lock async converts                        0            0.0          0.0
global lock async gets                          197            0.5         32.8
global lock convert time                          4            0.0          0.7
```

```
global lock get time                          290          0.7        48.3
global lock releases                        3,064          7.0       510.7
global lock sync converts                      30          0.1         5.0
global lock sync gets                       3,120          7.2       520.0
```

The next three reports deal with latches. Latches are low level lock structures that protect memory areas. There is usually no concern with latches unless high sleeps or misses are observed. Generally, adjusting internal parameters or tuning SQL code tunes latches. An example would be adjusting the *_kgl** parameters to tune the library related or shared pool locks. Sometimes, increasing the shared pool size can also help to relieve latch issues. For some latches, the number of latches available are derived from the size of the shared pool and the settings for related initialization parameters.

```
Latch Activity for DB: MIKE  Instance: mike2  Snaps: 25 -26
->"Get Requests", "Pct Get Miss" and "Avg Slps/Miss" are statistics for
  willing-to-wait latch get requests
->"NoWait Requests", "Pct NoWait Miss" are for no-wait latch get requests
->"Pct Misses" for both should be very close to 0.0

                                    Pct    Avg   Wait                 Pct
                                Get Get   Slps   Time       NoWait NoWait
Latch                       Requests Miss /Miss    (s)     Requests   Miss
------------------------ ------------ ------ ------ ------ ------------ ------
KCL bast context freelis         1,150    0.0             0            0
KCL freelist parent latc           713    0.0             0            0
KCL gc element parent la         8,399    0.0             0          166    0.0
KCL name table parent la         1,780    0.0             0          144    0.0
KJC message pool free li           655    0.0             0          194    0.0
KJCT flow control latch          2,238    0.0    1.0      0           91    0.0
gcs opaque info freelist         1,922    0.0             0            0
gcs resource freelist               77    0.0             0            0
gcs resource hash                5,719    0.0    1.0      0            0
gcs shadows freelist               681    0.0             0            0
ges caches resource list         3,518    0.0             0        2,130    0.0
ges deadlock list                  411    0.0             0            0
ges domain table                 4,206    0.0             0            0
ges enqueue table freeli         6,271    0.0             0            0
ges group parent                 4,238    0.0             0            0
ges group table                  6,409    0.0             0            0
ges process hash list              207    0.0             0            0
ges process parent latch        18,354    0.0    1.0      0            0
ges process table freeli             4    0.0             0            0
ges resource hash list          10,294    0.1    1.0      0        1,248    0.0
ges resource table freel         5,703    0.0             0            0
ges timeout list                    72    0.0             0            0

Latch Sleep breakdown for DB: MIKE  Instance: mike2  Snaps: 25 -26
-> ordered by misses desc

                                    Get                          Spin &
Latch Name                       Requests      Misses      Sleeps Sleeps 1->4
-------------------------- -------------- ----------- ----------- ------------
ges resource hash list             10,294          13          13 0/13/0/0/0
redo allocation                    41,394           4           4 0/4/0/0/0
KJCT flow control latch             2,238           1           1 0/1/0/0/0
cache buffers lru chain             3,055           1           1 0/1/0/0/0
gcs resource hash                   5,719           1           1 0/1/0/0/0
ges process parent latch           18,354           1           1 0/1/0/0/0
         -------------------------------------------------------------
Latch Miss Sources for DB: MIKE  Instance: mike2  Snaps: 25 -26
-> only latches with sleeps are shown
-> ordered by name, sleeps desc

                                                  NoWait              Waiter
Latch Name               Where                    Misses     Sleeps   Sleeps
------------------------ ------------------------ ------- ---------- --------
KJCT flow control latch  kjcts_sedeqv: dequeue a ve       0          1        0
cache buffers lru chain  kcbzgb: wait                     0          1        1
gcs resource hash        kjbmpconvert                     0          1        1
ges resource hash list   kjrmas1: lookup master nod       0         13        0
redo allocation          kcrfwr                           0          4        2
         -------------------------------------------------------------
```

The next report deals with the data dictionary cache area of the shared pool. If there are no conflicts or releases, the *dc* caches are sized correctly. The dictionary cache area is sized by properly sizing the shared pool.

```
Dictionary Cache Stats for DB: MIKE  Instance: mike2  Snaps: 25 -26
                                      GES          GES          GES
```

```
Cache                        Requests    Conflicts     Releases
------------------------- ------------ ------------ ------------
dc_global_oids                      0            0            0
dc_object_ids                      22            0            0
dc_objects                         97            0            0
dc_profiles                         0            0            0
dc_rollback_segments                0            0            0
dc_segments                         3            0            0
dc_tablespace_quotas                2            0            0
dc_tablespaces                      0            0            0
dc_user_grants                      0            0            0
dc_usernames                        0            0            0
dc_users                            0            0            0
------------------------------------------------------------
```

The next report gives a breakdown of enqueues by type of enqueue. This enqueue type report will help isolate the cause of enqueue problems. The types of enqueue are shown in Table 14.2

Type	Description
BL	Buffer Cache Management
CF	Controlfile Transaction
CI	Cross-instance Call Invocation
CU	Bind Enqueue
DF	Datafile
DL	Direct Loader Index Creation
DM	Database Mount
DR	Distributed Recovery
DX	Distributed TX
FS	File Set
IN	Instance Number
IR	Instance Recovery
IS	Instance State
IV	Library Cache Invalidation
JQ	Job Queue
KK	Redo Log "Kick"
L[A-P]	Library Cache Lock
MR	Media Recovery
N[A-Z]	Library Cache Pin
PF	Password File
PI	Parallel Slaves
PR	Process Startup
PS	Parallel Slave Synchronization
Q[A-Z]	Row Cache
RT	Redo Thread
SC	System Commit Number
SM	SMON
SQ	Sequence Number Enqueue
SR	Synchronized Replication
SS	Sort Segment
ST	Space Management Transaction
SV	Sequence Number Value

Type	Description
TA	Transaction Recovery
TM	DML Enqueue
TS	Temporary Segment (also TableSpace)
TT	Temporary Table
TX	Transaction
UL	User-defined Locks
UN	User Name
US	Undo Segment: Serialization
WL	Being Written Redo Log
XA	Instance Attribute Lock
XI	Instance Registration Lock

Table 14.2: *List of Enqueues*

In the report, the majority enqueue is the TM or DML related enqueue. However, its average wait time is only 2.43 milliseconds. The transaction recovery (TA) enqueue has a whopping 497 millisecond wait time. This high value was driven by several rollbacks in the test procedure. Once the enqueue causing the problems are determined, standard tuning techniques can be used to resolve them.

```
Enqueue activity for DB: MIKE  Instance: mike2  Snaps: 25 -26
-> Enqueue stats gathered prior to 10g should not be compared with 10g data
-> ordered by Wait Time desc, Waits desc

                                                        Avg Wt         Wait
Eq     Requests    Succ Gets Failed Gets       Waits   Time (ms)     Time (s)
-- ------------ ------------ ----------- ----------- ------------ ------------
TA            1            1           0           1       497.00            0
TM        1,582        1,582           0          14         2.43            0
HW           13           13           0           5         2.60            0
FB            4            4           0           4         3.00            0
TT            3            3           0           3         2.33            0
          -------------------------------------------------------------
```

The final report section to be reviewed is the library cache report dealing with the GES. GES invalid requests and GES invalidations could indicate insufficient sizing of the shared pool, resulting in GES contention.

```
Library Cache Activity for DB: MIKE  Instance: mike2  Snaps: 25 -26
->"Pct Misses"  should be very low

                    GES Lock      GES Pin      GES Pin   GES Inval GES Invali-
Namespace           Requests     Requests     Releases    Requests     dations
--------------- ------------ ------------ ------------ ----------- -----------
BODY                       1            0            0           0           0
CLUSTER                    4            0            0           0           0
INDEX                     84            0            0           0           0
SQL AREA                   0            0            0           0           0
TABLE/PROCEDURE          617          192            0          77           0
TRIGGER                    0            0            0           0           0
          -------------------------------------------------------------
```

This section on the STATSPACK reports has only covered the reports that dealt with the RAC environment. This does not mean that the rest of the report can be ignored. The sections on SQL use are critical in light of the need to find the code that may actually be causing the problems. The other waits, latches, enqueues, and

statistics that deal with the mundane parts of the Oracle environment are also important to review, monitor, and tune.

Global Cache Services (GCS) Monitoring

The use of the GCS relative to the number of buffer cache reads, or logical reads can be estimated by dividing the sum of GCS requests (*global cache gets* + *global cache converts* + *global cache cr blocks received* + *global cache current blocks received*) by the number of logical reads (*consistent gets* + *db block gets*) for a given statistics collection interval. A global cache service request is made in Oracle when a user attempts to access a buffer cache to read or modify a data block and the block is not in the local cache. A remote cache read, disk read or change access privileges is the inevitable result. These are logical read related. Logical reads form a superset of the global cache service operations. The calculation for *global cache hit ratio* since instance startup is:

```
SELECT
  a.inst_id "Instance",
  (A.VALUE+B.VALUE+C.VALUE+D.VALUE)/(E.VALUE+F.VALUE) "GLOBAL CACHE HIT RATIO"
FROM
  GV$SYSSTAT A,
  GV$SYSSTAT B,
  GV$SYSSTAT C,
  GV$SYSSTAT D,
  GV$SYSSTAT E,
  GV$SYSSTAT F
WHERE
  A.NAME='gc gets'
  AND B.NAME='gc converts'
  AND C.NAME='gc cr blocks received'
  AND D.NAME='gc current blocks received'
  AND E.NAME='consistent gets'
  AND F.NAME='db block gets'
SEE CODE DEPOT FOR COMPLETE SCRIPT (http://www.oracle-script.com/)


  Instance GLOBAL CACHE HIT RATIO
---------- ---------------------
         1              .02403656
         2             .014798887
```

The instance with the best access to the drives, or the faster I/O path, will likely have the best *cache hit ratio*. This is due to the way Oracle's RAC caching algorithm works as it may decide that the cost of doing a local read is higher than reading into the other cache and siphoning it across the cluster interconnect. In formula form:

```
(gc gets + gc converts + gc cr blocks received +
gc current blocks received) / (consistent gets + db block gets)
```

Blocks frequently requested by local and remote users will be very hot. If a block is hot, its transfer is delayed for a few milliseconds to allow the local users to complete their work. The following ratio provides a rough estimate of how prevalent this is:

```
SELECT
```

```
  A.INST_ID "Instance",
  A.VALUE/B.VALUE "BLOCK TRANSFER RATIO"
FROM
  GV$SYSSTAT A, GV$SYSSTAT B
WHERE
  A.NAME='gc defers'
  AND B.NAME='gc current blocks served'
 SEE CODE DEPOT FOR COMPLETE SCRIPT (http://www.oracle-script.com/)


  Instance BLOCK TRANSFER RATIO
---------- --------------------
         1           .052600105
         2           .078004479
```

If the above SELECT generates a ratio of more than 0.3, a fairly hot data set is indicated. If this is the case, blocks involved in busy waits should be analyzed. The following columns should be queried to find the blocks involved in busy waits:

- *name*
- *kind*
- *forced_reads*
- *forced_writes*

For example:

```
col instance format 99999999
col name format a20
col kind format a10
set lines 80 pages 55
Select
INST_ID "Instance",
NAME,
KIND,
sum(FORCED_READS) "Forced Reads",
sum(FORCED_WRITES) "Forced Writes"
FROM GV$CACHE_TRANSFER
SEE GRID CODE DEPOT FOR DOWNLOAD (http://www.rampant-
books.com/book_2004_1_10g_grid.htm)


Instance NAME                 KIND       Forced Reads Forced Writes
--------- -------------------- ---------- ------------ -------------
        1 MOD_TEST_IND         INDEX               308             0
        1 TEST2                TABLE                64             0
        1 AQ$_QUEUE_TABLES     TABLE                 5             0
        2 TEST2                TABLE               473             0
        2 MOD_TEST_IND         INDEX               221             0
        2 AQ$_QUEUE_TABLES     TABLE                 2             0
```

These values come from the *gv$cache_transfer* view. Alternatively, the *cr_requests* and *current_requests* columns in *gv$cr_block_server* can be examined. Also, the values shown for the *global cache busy*, *buffer busy global cache*, and *buffer busy global cr* statistics from the *gv$sysstat* view should be examined.

```
SELECT
  INST_ID,
  sum(CR_REQUESTS) "CR Requests",
  sum(CURRENT_REQUESTS) "Current Requests"
FROM
```

```
  GV$CR_BLOCK_SERVER
GROUP BY
  INST_ID;

   INST_ID CR Requests Current Requests
---------- ----------- ---------------
         1       28940            2244
         2       31699             837

SELECT
  inst_id "Instance",
  event "Wait Event",
  total_waits,
  time_waited
FROM
  GV$SYSTEM_EVENT
WHERE
      event in (
       'global cache busy',
       'buffer busy global cache',
       'buffer busy global CR')
SEE CODE DEPOT FOR COMPLETE SCRIPT (http://www.oracle-script.com/)


 Instance Wait Event               TOTAL_WAITS TIME_WAITED
--------- ----------------------- ----------- -----------
        1 buffer busy global CR             1           0
        1 global cache busy              1073        7171
        2 global cache busy               973        7524
```

If a problem is discovered, the object causing the problem should be identified along with the instance that is accessing the object, and how the object is being accessed. If necessary, the contention can be alleviated by:

- Reducing hot spots by spreading the accesses to index blocks or data blocks.
- Using Oracle hash or range partitions wherever applicable, just as it would be done in single instance Oracle databases.
- Reducing concurrency on the object by implementing load balancing or resource management. For example, decrease the rate of modifications to that object by using fewer database processes.

In RAC, as in a single instance Oracle database, blocks are only written to disk for aging, cache replacement, or checkpoints. When a data block is replaced from the cache due to aging or when a checkpoint occurs and the block was previously changed in another instance but not written to disk, Oracle sends a message to notify the other instance that Oracle will perform a fusion write to move the data block to disk.

These fusion writes are monitored with the following ratio. It reveals the proportion of writes that Oracle manages.

```
SELECT
 a.inst_id "Instance",
 A.VALUE/B.VALUE "Cache Fusion Writes Ratio"
FROM
  GV$SYSSTAT A,
  GV$SYSSTAT B
WHERE
```

```
      a.name='DBWR fusion writes'
  AND b.name='physical writes'
  AND b.inst_id=a.inst_id
ORDER BY
  A.INST_ID;

Instance Cache Fusion Writes Ratio
--------- -------------------------
        1                .216290958
        2                .131862042
```

The larger this ratio is, the higher the number of written blocks that have been copied with their previous changes between the RAC instances. A large ratio is the result of:

- Insufficiently sized caches.
- Insufficient checkpoints.
- Large numbers of buffers written due to cache replacement or checkpointing.

For example, 0.21 means that 21% of the buffers written to disk were globally dirty. A fusion write does not involve an additional write to disk. A fusion write does require messaging to arrange the transfer with the other instances. This indicates that fusion writes are in fact a subset of all the instance's physical writes.

Use of the v$*cache_transfer* Views

The *v$cache_transfer* and *v$file_cache_transfer* views are used to examine RAC statistics. The types of blocks that use the cluster interconnects in a RAC environment are monitored with the *v$* cache transfer series of views:

- *v$cache_transfer:* This view shows the types and classes of blocks that Oracle transfers over the cluster interconnect on a per-object basis. The *forced_reads* and *forced_writes* columns can be used to determine the types of objects the RAC instances are sharing. Values in the *forced_writes* column show how often a certain block type is transferred out of a local buffer cache due to the current version being requested by another instance. The following columns are the same for all of the views.
- *v$class_cache_transfer:* -- This view can be used to identify the class of blocks that experience cache transfers. *v$class_cache_transfer* has a *class* column showing the class of a block; therefore, this view can be used to assess block transfers per class of object.
- *v$file_cache_transfer:* This view can be used to monitor the blocks transferred per file. The *file_number* column identifies the datafile that contained the blocks transferred.
- *v$temp_cache_transfer:* -- This view can be used to monitor the transfer of temporary tablespace blocks. The view contains a *file_number* column that is

used to track, by the tempfile file number, the number of blocks transferred. This view has the same structure as the *v$temp_cache_transfer* view.

The contents of the *v$cache_transfer* view are shown below.

```
Description of the V$CACHE_TRANSFER view

Name                      Type
------------------------  --------------
FILE#                     NUMBER
BLOCK#                    NUMBER
CLASS#                    NUMBER
STATUS                    VARCHAR2(5)
XNC                       NUMBER
FORCED_READS              NUMBER
FORCED_WRITES             NUMBER
NAME                      VARCHAR2(30)
PARTITION_NAME            VARCHAR2(30)
KIND                      VARCHAR2(15)
OWNER#                    NUMBER
GC_ELEMENT_ADDR           RAW(4)
GC_ELEMENT_NAME           NUMBER
```

The *v$cache_transfer* view shows the types and classes of blocks that Oracle transfers over the cluster interconnect on a per-object basis. The *forced_reads* and *forced_writes* columns are used to determine which types of objects the RAC instances are sharing.

The *v$file_cache_transfer* view is used to identify files that have experienced cache transfers. For example, while *v$cache_transfer* has a *name* column showing the name of an object, the *v$file_cache_transfer* view has the *file_number* column to show the file numbers of the datafiles that are the source of blocks transferred; therefore, this view can be used to assess block transfers per file. The *v$file_cache_transfer* view contents are shown below:

```
Description of the V$FILE_CACHE_TRANSFER View
 Name                           Type
 ------------------------------ ----------
 FILE_NUMBER                    NUMBER
 X_2_NULL                       NUMBER
 X_2_NULL_FORCED_WRITE          NUMBER
 X_2_NULL_FORCED_STALE          NUMBER
 X_2_S                          NUMBER
 X_2_S_FORCED_WRITE             NUMBER
 S_2_NULL                       NUMBER
 S_2_NULL_FORCED_STALE          NUMBER
 RBR                            NUMBER
 RBR_FORCED_WRITE               NUMBER
 RBR_FORCED_STALE               NUMBER
 NULL_2_X                       NUMBER
 S_2_X                          NUMBER
 NULL_2_S                       NUMBER
 CR_TRANSFERS                   NUMBER
 CUR_TRANSFERS                  NUMBER
```

Even though the shared disk architecture virtually eliminates forced disk writes, the *v$cache_transfer* and *v$file_cache_transfer* views may still show the number of block mode conversions per block class or object. Values in the *forced_writes* column, however, will be zero.

Monitoring the GES Processes

The monitoring of the global enqueue services (GES) process is performed using the *gv$enqueue_stat* view. The contents of the *gv$enqueue_stat* view are shown below:

```
Description of the view GV$ENQUEUE_STAT

Name                                      Type
----------------------------------------- ----------
INST_ID                                   NUMBER
EQ_TYPE                                   VARCHAR2(2)
TOTAL_REQ#                                NUMBER
TOTAL_WAIT#                               NUMBER
SUCC_REQ#                                 NUMBER
FAILED_REQ#                               NUMBER
CUM_WAIT_TIME                             NUMBER
```

An example SELECT to retrieve all of the enqueues with *total_wait* number greater than zero would be:

```
select
   *
from
   gv$enqueue_stat
where
   total_wait#>0
order by
   inst_id,
cum_wait_time desc;
```

SEE CODE DEPOT FOR COMPLETE SCRIPT (http://www.oracle-script.com/)

```
   INST_ID EQ TOTAL_REQ# TOTAL_WAIT#  SUCC_REQ# FAILED_REQ# CUM_WAIT_TIME
---------- -- ---------- ----------- ---------- ----------- -------------
         1 TX      31928          26      31928           0        293303
         1 PS        995         571        994           1         55658
         1 TA       1067         874       1067           0         10466
         1 TD        974         974        974           0          2980
         1 DR        176         176        176           0           406
         1 US        190         189        190           0           404
         1 PI         47          27         47           0           104
         1 CF     499314          23     499314           0            47
         1 TM      41928           8      41928           0            35
         1 MR         93          13         93           0            21
         1 HW        637           6        637           0             8
         1 XR          4           2          4           0             5
         1 DM          4           2          4           0             4
         1 SR          1           1          1           0             3
         1 SW          3           1          3           0             3
         1 TS          2           1          2           0             3
         2 TA       1064        1015       1064           0        437648
         2 PS       2208         325       1597         611        104273
         2 TX     440843          18     440843           0         62787
         2 US        197          41        197           0          8551
         2 IR        193          29        193           0          4593
         2 TT       4393         131       4393           0          3363
         2 CF     507497         540     507497           0          1726
         2 TM    1104694         101    1104694           0           766
         2 DM          5           2          5           0           483
         2 HW        444          41        444           0           108
         2 PI         90          18         90           0            81
```

```
    2 DL         32         18         32          0          55
    2 DR        176         23        176          0          52
    2 RT          4          4          3          1          12
    2 FB         10          2         10          0           6
    2 IA          1          1          1          0           4
    2 PG          1          1          1          0           3
    2 TS          4          1          4          0           3
```

According to Oracle, the enqueues of interest, as shown in the *eq_type* column of the *gv$enqueue_stat* view in the RAC environment are:

- SQ Enqueue: This indicates that there is contention for sequences. In almost all cases, executing an ALTER SEQUENCE command can increase the cache size of sequences used by the application. When creating sequences for a RAC environment, DBAs should use the NOORDER keyword to avoid an additional cause of SQ enqueue contention that is forced ordering of queued sequence values.
- TX Enqueue*:* This is usually an application related issue pertaining to row locking. Real Application Clusters processing can magnify the effect of TX enqueue waits. Performance bottlenecks can also appear on leaf blocks of right growing indexes as TX enqueue waits while the index block splits are occuring. TX enqueue performance issues can be resolved by setting the value of the *initrans* parameter for a TABLE or INDEX to be equal to the number of CPUs per node multiplied by the number of nodes in the cluster multiplied by 0.75. Another technique is to determine the number of simultaneous accesses for DML for the objects experiencing TX enqueues, and setting *initrans* to that value. Oracle Corporation recommends avoiding setting this parameter greater than 100. Another parameter that can reduce TX enqueues is *maxtrans*. *maxtrans* determines the maximum number of transactions that can access a block. *maxtrans* will default to 255 and it is a good practice to reset this to less than 100.

PS (Parallel Slave Synchronization) and TA (Transaction Recovery) enqueues also seem to have some importance in the environment. Therefore, start with a wide sweep and then focus on the waits that are causing performance issues in the environment.

There are other interesting views that provide information on tuning that the DBA in a RAC environment should be aware of, for example:

- *gv$segment_statistics:* Provides statistics such as *buffer busy waits* on a per segment basis. This allows tracking of exactly which segments, indexes or tables, are causing the *buffer busy waits* or other statistics to increment. To select against the *gv$segment_statistics* view, the user will want to SELECT for a specific statistic name where the value is greater than a predetermined limit. The contents of *gv$segment_statistics* are shown below:

```
Description of gv$segment_statistics
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
```

```
INST_ID                                                   NUMBER
OWNER                                                     VARCHAR2(30)
OBJECT_NAME                                               VARCHAR2(30)
SUBOBJECT_NAME                                            VARCHAR2(30)
TABLESPACE_NAME                                           VARCHAR2(30)
TS#                                                       NUMBER
OBJ#                                                      NUMBER
DATAOBJ#                                                  NUMBER
OBJECT_TYPE                                               VARCHAR2(18)
STATISTIC_NAME                                            VARCHAR2(64)
STATISTIC#                                                NUMBER
VALUE                                                     NUMBER
```

Another useful view shows which file IDs which have been remastered, which happens when they are transferred from one instance to another, this view is called *gv$gcspfmaster_info* and its contents are shown below:

```
Description of gv$gcspfmaster_info
 Name                                      Null?    Type
 ----------------------------------------- -------- -------
 INST_ID                                            NUMBER
 FILE_ID                                            NUMBER
 CURRENT_MASTER                                     NUMBER
 PREVIOUS_MASTER                                    NUMBER
 REMASTER_CNT                                       NUMBER
```

The *file_id* column corresponds to the data file ID. The current and previous masters refer to the instances that are either the current or previous master of the specified file. A view related to the *gv$gcspfmaster_info* view is the *gv$gchvmaster_info* view which shows the same information but for the PCM hash value IDs for specific resources that have been remastered. This views contents are shown below:

```
 Description of gv$gcshvmaster_info
 Name                                      Null?    Type
 ----------------------------------------- -------- -------
 INST_ID                                            NUMBER
 HV_ID                                              NUMBER
 CURRENT_MASTER                                     NUMBER
 PREVIOUS_MASTER                                    NUMBER
 REMASTER_CNT                                       NUMBER
```

To select against these views, it may be desirable to restrict on the *remaster_cnt* value being greater than a predetermined limit.

The *gv$sqlarea* view has also been enhanced in Oracle Database 10g RAC. The column *cluster_wait_time* in *gv$sqlarea* represents the wait time incurred by individual SQL statements for global cache events and will identify the SQL which may need to be tuned based on its contribution to RAC contention.

Monitoring and Tuning using OEM

The first part of this chapter covered the manual aspects of RAC tuning, showing techniques and providing scripts to obtain the needed information from the various underlying Oracle views. This section will cover the tools provided by

Oracle in the Oracle Enterprise manager (OEM) Performance Monitor. In order to use OEM to monitor a RAC environment, the following must be in place:

- OEM installed.
- OEM repository.
- Oracle Intelligent Agents running.

Normally, if the standard installation is used, OEM will be installed as a matter of course. If possible, the OEM and its repository should be set up on a separate system, such as a small Windows-based server. At the least, a small repository database should be created on one of the RAC servers. This can be used for the RMAN repository as well. The OEM client software can be installed on any desktop as long as that desktop has SQL*Net connectivity to the OEM repository.

The most difficult part of the setup for OEM is probably getting the intelligent agents to work properly with the RAC environment.

Configuring the Oracle Intelligent Agent with RAC

In order to be sure that the agent is installed properly in Oracle Database 10g, the following Metalink documents should be reviewed:

- EM 10g Database Control Release Notes 10.1.0.2.0 Note: 266770.1
- EM 10g GRID Control Release Notes 10.1.0.2.0 Note: 266769.1
- How to Log and Trace the EM 10g Management Agents Note: 229624.1
- How To Install The Downloadable Central Management Agent in EM 10g Grid Control Note: 235287.1

The nodes and databases must be properly discovered and viewable from the OEM webpage before they can be monitored.

Using EM in Oracle Database 10g

In 10g, Oracle is moving to the web based interface for EM and will soon deprecate the Java based version. Monitoring and management functions should be migrated into the HTML Web based version of EM as soon as possible. The following sections will provide a quick look at the screens in the new HTML version that the DBA should be using for RAC monitoring and management.

The Cluster Performance Page

In the HTML based web version of the EM, the Cluster Performance Page is used to display the usage statistics for all RAC hosts or for individual RAC hosts. This

information allows you to add, suspend, or redistribute resources as the need arises. Figure 14.1 shows this page.

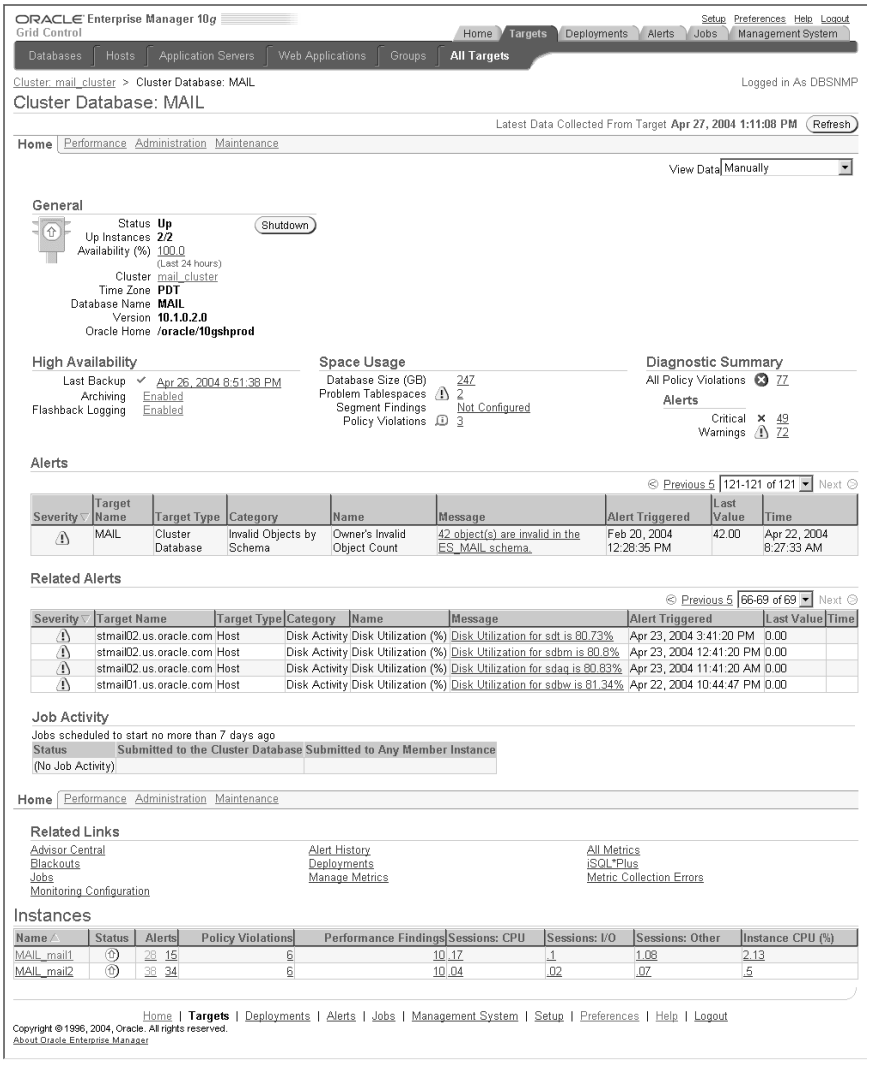

Figure 14.1: *Example Cluster Performance Page*

The Cluster Database Performance Page

The Oracle Database 10g EM Cluster Database Performance Page displays statistics via charts that show run queue length, paging rate, service time, and the database throughput for each RAC host or RAC instance. The page is also used to access the detailed information for the Wait Class Page for Service Time and the Top Sessions Page for Database Throughput. Figure 14.2 shows an example of this page.

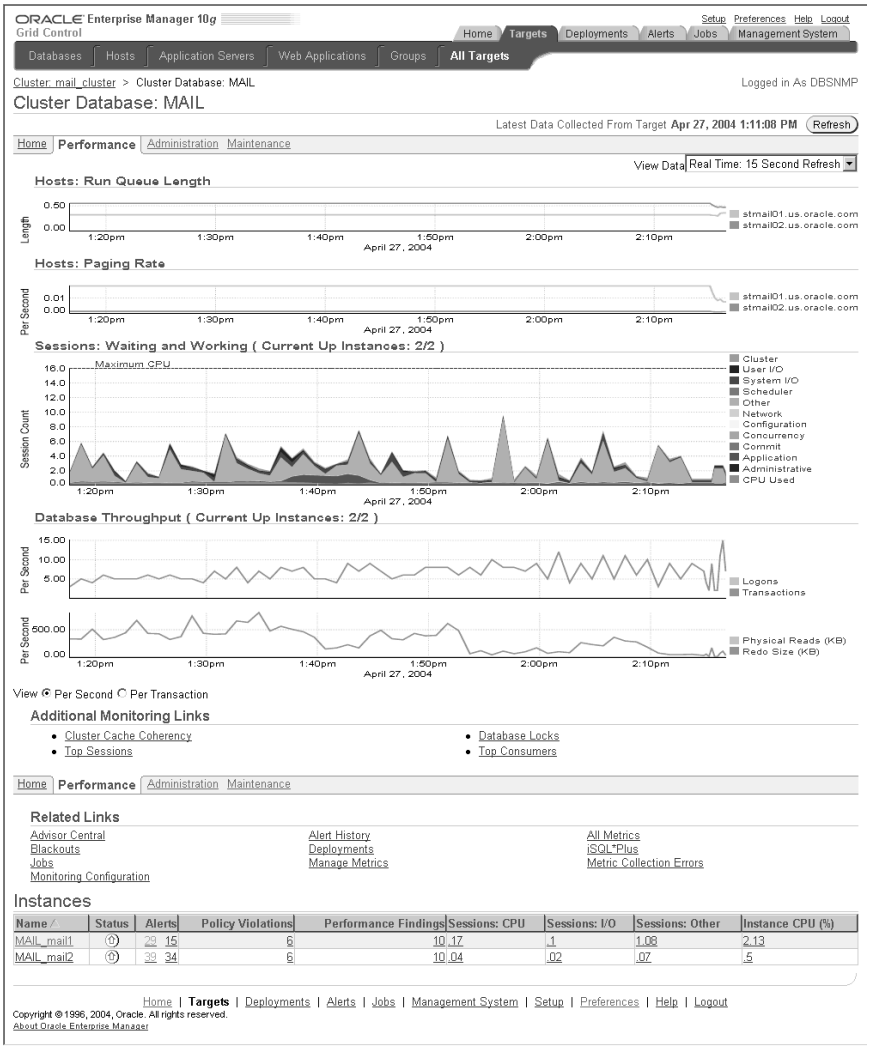

Figure 14.2: *Example Cluster Database Performance Page*

The Cluster Cache Coherency Instances Page

The Cluster Cache Coherency Instances Page is used to provide real-time monitoring of global cache statistics. The Cluster Cache Coherency Instances Page will display tables of metrics from the following groups for all cluster instances:

- Block Access Statistics
- Global Cache Convert, Global Cache Current Block Request, Global Cache CR
- Block Request
- Top 5 Library Cache Lock and Top 5 Row Cache Lock

Figure 14.3 shows the Cluster Cache Coherency Page:

Figure 14.3: *Example Cluster Cache Coherency Page*

While the Enterprise Manager monitors events at the database and instance levels and any available node can monitor database events, only one node at a time monitors the entire database while each node monitors events for its local instances.

By using the various screens and reports in OEM Performance Manager for RAC, the status of virtually any section of the RAC environment can be seen. The screens and reports provide suggestions to correct problems as they occur.

Other Items to Monitor for RAC

In RAC, it is important to remember that multiple nodes are accessing the same database files. If the DBA only monitors from a single instance using the *v$* views, the effects from the other nodes will not be visible. The *gv$* views must be used to examine data from all nodes in order to get a complete picture of the RAC environment's performance.

An excellent example of this is the monitoring of file IO rates and file IO timing:

```
rem NAME: fileio.sql
rem
rem FUNCTION: Reports on the file io status of all of the
rem FUNCTION: datafiles in the database.

rem HISTORY:
rem WHO             WHAT           WHEN
rem Mike Ault               Created        1/5/2003
```

```
rem
column sum_io1 new_value st1 noprint
column sum_io2 new_value st2 noprint
column sum_io new_value divide_by noprint
column Percent format 999.999 heading 'Percent|Of IO'
column brratio format 999.99 heading 'Block|Read|Ratio'
column bwratio format 999.99 heading 'Block|Write|Ratio'
column phyrds heading 'Physical | Reads'
column phywrts heading 'Physical | Writes'
column phyblkrd heading 'Physical|Block|Reads'
column phyblkwrt heading 'Physical|Block|Writes'
column name format a45 heading 'File|Name'
column file# format 9999 heading 'File'
column dt new_value today noprint
select to_char(sysdate,'ddmonyyyyhh24miss') dt from dual;
set feedback off verify off lines 132 pages 60 sqlbl on trims on
rem
select
     nvl(sum(a.phyrds+a.phywrts),0) sum_io1
from
     sys.gv_$filestat a;
select nvl(sum(b.phyrds+b.phywrts),0) sum_io2
from
        sys.gv_$tempstat b;
select &st1+&st2 sum_io from dual;
rem
@title132 'File IO Statistics Report'
spool rep_out\&db\rac_fileio&&today
select
     a.inst_id, a.file#,b.name, a.phyrds, a.phywrts,
     (100*(a.phyrds+a.phywrts)/&divide_by) Percent,
     a.phyblkrd, a.phyblkwrt, (a.phyblkrd/greatest(a.phyrds,1)) brratio,
       (a.phyblkwrt/greatest(a.phywrts,1)) bwratio
from
     sys.gv_$filestat a, sys.gv_$dbfile b
where
     a.inst_id=b.inst_id and
     a.file#=b.file#
union
select
     c.inst_id,c.file#,d.name, c.phyrds, c.phywrts,
     (100*(c.phyrds+c.phywrts)/&divide_by) Percent,
     c.phyblkrd, c.phyblkwrt,(c.phyblkrd/greatest(c.phyrds,1)) brratio,
       (c.phyblkwrt/greatest(c.phywrts,1)) bwratio
from
     sys.gv_$tempstat c, sys.gv_$tempfile d
where
        c.inst_id=d.inst_id and
     c.file#=d.file#
order by
     1,2
/
spool off
pause Press enter to continue
set feedback on verify on lines 80 pages 22
clear columns
ttitle off
```

The output from the above script looks like the following:

```
Date: 01/22/04
Page:   1
Time: 01:38 PM                                    File IO Statistics Report
TSTDBMRA
                                                    tstdb database
```

INST_ID	File	File Name	Physical Reads	Physical Writes	Percent Of IO	Block Reads	Block Writes	Physical Read Ratio	Physical Write Ratio	Block Read Ratio	Block Write Ratio
2	1	/od04_01/oradata/tstdb/system01.dbf	1731	2540	.009	2717	2540	1.57	1.00		
2	1	/od04_01/oradata/tstdb/temp01.dbf	195386	135805	.723	1688145	1724988	8.64	12.70		
2	2	/od04_01/oradata/tstdb/undotbs01.dbf	524	523	.002	524	523	1.00	1.00		
2	3	/od04_01/oradata/tstdb/drsys01.dbf	524	523	.002	524	523	1.00	1.00		
2	4	/od04_01/oradata/tstdb/indx01.dbf	524	523	.002	524	523	1.00	1.00		
2	5	/od04_01/oradata/tstdb/tools01.dbf	524	523	.002	524	523	1.00	1.00		
2	6	/od04_01/oradata/tstdb/undotbs02.dbf	545	23867	.053	545	23867	1.00	1.00		
2	7	/od04_01/oradata/tstdb/undotbs03.dbf	524	523	.002	524	523	1.00	1.00		
2	8	/od04_01/oradata/tstdb/users01.dbf	524	523	.002	524	523	1.00	1.00		
2	9	/od04_01/oradata/tstdb/xdb01.dbf	530	523	.002	545	523	1.03	1.00		
2	10	/od04_01/oradata/tstdb/tstdb_globald01.dbf	525	523	.002	525	523	1.00	1.00		
2	11	/od04_01/oradata/tstdb/tstdb_globalx01.dbf	525	523	.002	525	523	1.00	1.00		
2	12	/od04_01/oradata/tstdb/tstdb_reportd01.dbf	524	523	.002	524	523	1.00	1.00		
2	13	/od04_01/oradata/tstdb/tstdb_reportx01.dbf	524	523	.002	524	523	1.00	1.00		
2	14	/od04_01/oradata/tstdb/nomadd01.dbf	524	523	.002	524	523	1.00	1.00		
2	15	/od04_01/oradata/tstdb/TA1d01.dbf	524	523	.002	524	523	1.00	1.00		
2	16	/od04_01/oradata/tstdb/TA1x01.dbf	524	523	.002	524	523	1.00	1.00		
2	17	/od04_01/oradata/tstdb/SRCd01.dbf	131430	523	.288	3762539	523	28.63	1.00		
2	18	/od04_01/oradata/tstdb/SRCx01.dbf	5410	523	.013	5410	523	1.00	1.00		
2	19	/od04_01/oradata/tstdb/REEd01.dbf	524	523	.002	524	523	1.00	1.00		
2	20	/od04_01/oradata/tstdb/REEx01.dbf	524	523	.002	524	523	1.00	1.00		
2	21	/od04_01/oradata/tstdb/CRWd01.dbf	524	523	.002	524	523	1.00	1.00		
2	22	/od04_01/oradata/tstdb/CRWx01.dbf	524	523	.002	524	523	1.00	1.00		
2	23	/od04_02/oradata/tstdb/LWEd01.dbf	519	519	.002	519	519	1.00	1.00		
2	24	/od04_02/oradata/tstdb/LWEx01.dbf	519	519	.002	519	519	1.00	1.00		
2	25	/od04_01/oradata/tstdb/perfstat01.dbf	110	110	.000	110	110	1.00	1.00		
3	1	/od04_01/oradata/tstdb/system01.dbf	5870952	43328	12.920	5879481	43328	1.00	1.00		
3	1	/od04_01/oradata/tstdb/temp01.dbf	2459053	1219824	8.036	22005243	15402399	8.95	12.63		
3	2	/od04_01/oradata/tstdb/undotbs01.dbf	62411	601	.138	62411	601	1.00	1.00		
3	3	/od04_01/oradata/tstdb/drsys01.dbf	475816	601	1.041	475816	601	1.00	1.00		
3	4	/od04_01/oradata/tstdb/indx01.dbf	604	601	.003	604	601	1.00	1.00		
3	5	/od04_01/oradata/tstdb/tools01.dbf	835	643	.003	1553	643	1.86	1.00		
3	6	/od04_01/oradata/tstdb/undotbs02.dbf	608	707	.003	608	707	1.00	1.00		
3	7	/od04_01/oradata/tstdb/undotbs03.dbf	88095	547959	1.389	88095	547959	1.00	1.00		
3	8	/od04_01/oradata/tstdb/users01.dbf	3907	4289	.018	6098	5497	1.56	1.28		
3	9	/od04_01/oradata/tstdb/xdb01.dbf	4370138	601	9.548	4370317	601	1.00	1.00		
3	10	/od04_01/oradata/tstdb/tstdb_globald01.dbf	1547848	29866	3.446	1941544	29866	1.25	1.00		
3	11	/od04_01/oradata/tstdb/tstdb_globalx01.dbf	4353943	6356	9.525	4354433	6357	1.00	1.00		
3	12	/od04_01/oradata/tstdb/tstdb_reportd01.dbf	604	601	.003	604	601	1.00	1.00		
3	13	/od04_01/oradata/tstdb/tstdb_reportx01.dbf	604	601	.003	604	601	1.00	1.00		
3	14	/od04_01/oradata/tstdb/nomadd01.dbf	288384	601	.631	288384	601	1.00	1.00		
3	15	/od04_01/oradata/tstdb/TA1d01.dbf	338417	601	.741	338417	601	1.00	1.00		
3	16	/od04_01/oradata/tstdb/TA1x01.dbf	963876	601	2.107	963876	601	1.00	1.00		
3	17	/od04_01/oradata/tstdb/SRCd01.dbf	3075710	936826	8.765	9782425	971945	3.18	1.04		

```
        3    18 /od04_01/oradata/tstdb/SRCx01.dbf               1315213      94012    3.078    1550400
275893    1.18    2.93
        3    19 /od04_01/oradata/tstdb/REEd01.dbf               1191132        601    2.603    1191132
601    1.00    1.00
        3    20 /od04_01/oradata/tstdb/REEx01.dbf               3109339        601    6.794    3109339
601    1.00    1.00
        3    21 /od04_01/oradata/tstdb/CRWd01.dbf                   604        601     .003        604
601    1.00    1.00
        3    22 /od04_01/oradata/tstdb/CRWx01.dbf                   604        601     .003        604
601    1.00    1.00
        3    23 /od04_02/oradata/tstdb/LWEd01.dbf               7042322    3913365   23.933   88147193
4346731   12.52    1.11
        3    24 /od04_02/oradata/tstdb/LWEx01.dbf               1381676     508355    4.129    2064523
1265528    1.49    2.49
        3    25 /od04_01/oradata/tstdb/perfstat01.dbf               647       1845     .005        672
1845    1.04    1.00
```

The I/O balance is off between the two instances, two and three. If only instance two or only instance three were researched, the possible I/O problem would not have been evident.

Another I/O related statistic is the I/O timing. I/O timing would show if there are latency problems between the nodes. The following code shows an example file I/O timing report for RAC:

```
rem Purpose: Calculate IO timing values for datafiles
col inst_id format 9999999 heading 'Instance'
col name format a50 heading 'File Name'
set lines 132 pages 45
start title132 'IO Timing Analysis'
spool rep_out\&db\rac_io_time
select  f.inst_id,f.FILE# ,d.name,PHYRDS,PHYWRTS,READTIM/PHYRDS,WRITETIM/PHYWRTS
from gv$filestat f, gv$datafile d
where   f.inst_id=d.inst_id and
SEE CODE DEPOT FOR COMPLETE SCRIPT (http://www.oracle-script.com/)
```

An example output from the report above is shown below.

```
Date: 02/02/04                                                                          Page:   1
Time: 08:59 AM                          IO Timing Analysis                              PERFSTAT
                                           tstdb database

Instance FILE# File Name                                    PHYRDS PHYWRTS READTIM/PHYRDS WRITETIM/PHYWRTS
-------- ----- ---------------------------------------- ------- ------- -------------- --
       2    10 /od04_01/oradata/tstdb/tstdb_globald01.dbf     592      11           21.8               0
       1    10 /od04_01/oradata/tstdb/tstdb_globald01.dbf     632      21           20.4               0
       2    23 /od04_02/oradata/tstdb/LWEd01.dbf           100027    4023           5.94       .177479493
       1    17 /od04_01/oradata/tstdb/SRCd01.dbf            77626       6           3.61               0
       2    24 /od04_02/oradata/tstdb/LWEx01.dbf             1801     341           1.61       .263929619
       3    10 /od04_01/oradata/tstdb/tstdb_globald01.dbf  299320    6370           1.58       .195918367
       3    23 /od04_02/oradata/tstdb/LWEd01.dbf           294166   31246           1.44       1.53120399
       1    18 /od04_01/oradata/tstdb/SRCx01.dbf             1879       6           1.43               0
       3    24 /od04_02/oradata/tstdb/LWEx01.dbf           196574   35080           1.30       1.57374572
       2    17 /od04_01/oradata/tstdb/SRCd01.dbf            58099      61           1.16               0
       3     1 /od04_01/oradata/tstdb/system01.dbf         688550    2071           1.10       .125060357
       3    18 /od04_01/oradata/tstdb/SRCx01.dbf           186020       4           1.09               0
       3    17 /od04_01/oradata/tstdb/SRCd01.dbf           504230      36           1.06       1.02777778
       1    24 /od04_02/oradata/tstdb/LWEx01.dbf                8       6           .875       .333333333
       1    11 /od04_01/oradata/tstdb/tstdb_globalx01.dbf      45      10      .755555556               0
       1    23 /od04_02/oradata/tstdb/LWEd01.dbf               79      17      .683544304       .529411765
       3     7 /od04_01/oradata/tstdb/undotbs03.dbf            60   15243      .583333333       .460145641
       1     2 /od04_01/oradata/tstdb/undotbs01.dbf            29    2453      .551724138       .043212393
       2     6 /od04_01/oradata/tstdb/undotbs02.dbf            33    2501      .515151515       .019992003
       2    11 /od04_01/oradata/tstdb/tstdb_globalx01.dbf      65      12      .461538462               0
       1     7 /od04_01/oradata/tstdb/undotbs03.dbf             7       6      .428571429               0
       1    16 /od04_01/oradata/tstdb/TA1x01.dbf                7       6      .428571429               0
       1     1 /od04_01/oradata/tstdb/system01.dbf           1416     248      .399717514       .008064516
       2     1 /od04_01/oradata/tstdb/system01.dbf           2357     366      .391599491       .013661202
       2    25 /od04_01/oradata/tstdb/perfstat01.dbf          198       6      .328282828               0
       3     5 /od04_01/oradata/tstdb/tools01.dbf             174       8      .293103448               0
       1     6 /od04_01/oradata/tstdb/undotbs02.dbf             7       6      .285714286               0
       1    15 /od04_01/oradata/tstdb/TA1d01.dbf                7       6      .285714286               0
       1    13 /od04_01/oradata/tstdb/tstdb_reportx01.dbf       7       6      .285714286               0
       3     4 /od04_01/oradata/tstdb/indx01.dbf                7       4      .285714286               0
       2     5 /od04_01/oradata/tstdb/tools01.dbf               7       6      .285714286               0
       2     3 /od04_01/oradata/tstdb/drsys01.dbf               7       6      .285714286               0
```

```
     1    20 /od04_01/oradata/tstdb/REEx01.dbf               7       6     .285714286               0
     3     6 /od04_01/oradata/tstdb/undotbs02.dbf            5      18             .2               0
     3     8 /od04_01/oradata/tstdb/users01.dbf           6731       4     .199227455               0
     3    14 /od04_01/oradata/tstdb/nomadd01.dbf         24614     192     .188835622       .588541667
     3    25 /od04_01/oradata/tstdb/perfstat01.dbf       56010       4     .185841814               0
     3     3 /od04_01/oradata/tstdb/drsys01.dbf          51063       4     .181422948               0
     2    18 /od04_01/oradata/tstdb/SRCx01.dbf            3562       6     .179955081               0
     3    11 /od04_01/oradata/tstdb/tstdb_globalx01.dbf  468503      81     .179932679       .074074074
     3     2 /od04_01/oradata/tstdb/undotbs01.dbf            6      10     .166666667               0
     3     9 /od04_01/oradata/tstdb/xdb01.dbf           475840       4     .147650471               0
     1     4 /od04_01/oradata/tstdb/indx01.dbf               7       6     .142857143               0
     1     9 /od04_01/oradata/tstdb/xdb01.dbf                7       6     .142857143               0
     1    14 /od04_01/oradata/tstdb/nomadd01.dbf             7       6     .142857143               0
     1    12 /od04_01/oradata/tstdb/tstdb_reportd01.dbf      7       6     .142857143               0
     1    21 /od04_01/oradata/tstdb/CRWd01.dbf               7       6     .142857143               0
     2     4 /od04_01/oradata/tstdb/indx01.dbf               7       6     .142857143               0
     3    22 /od04_01/oradata/tstdb/CRWx01.dbf               7       4     .142857143               0
     2    21 /od04_01/oradata/tstdb/CRWd01.dbf               7       6     .142857143               0
     2    20 /od04_01/oradata/tstdb/REEx01.dbf               7       6     .142857143               0
     2    15 /od04_01/oradata/tstdb/TA1d01.dbf               7       6     .142857143               0
     2    12 /od04_01/oradata/tstdb/tstdb_reportd01.dbf      7       6     .142857143               0
     2     9 /od04_01/oradata/tstdb/xdb01.dbf                7       6     .142857143               0
     2     8 /od04_01/oradata/tstdb/users01.dbf              7       6     .142857143               0
     2     2 /od04_01/oradata/tstdb/undotbs01.dbf            7       6     .142857143               0
     1    25 /od04_01/oradata/tstdb/perfstat01.dbf           7       6     .142857143               0
     3    19 /od04_01/oradata/tstdb/REEd01.dbf          109796       4     .133611425               0
     3    15 /od04_01/oradata/tstdb/TA1d01.dbf           40327       4     .132839041               0
     3    20 /od04_01/oradata/tstdb/REEx01.dbf          333992       4     .121095715               0
     3    16 /od04_01/oradata/tstdb/TA1x01.dbf          103495       4     .120218368               0
```

At this point, it is still important to look for unbalanced timings between the instances.

The final example will look at system events. The following code shows the report script.

```
col event format a30 heading 'Event Name'
col waits format 999,999,999 heading 'Total|Waits'
col average_wait format 999,999,999 heading 'Average|Waits'
col time_waited format 999,999,999 heading 'Time Waited'
col total_time new_value divide_by noprint
col value new_value val noprint
col percent format 999.990 heading 'Percent|Of|Non-Idle Waits'
col duration new_value millisec noprint
col p_of_total heading 'Percent|of Total|Uptime' format 999.9999
set lines 132 feedback off verify off pages 50
 select to_number(sysdate-startup_time)*86400*1000 duration from v$instance;
select
sum(time_waited) total_time
from gv$system_event
where total_waits-total_timeouts>0
     and event not like 'SQL*Net%'
     and event not like 'smon%'
     and event not like 'pmon%'
     and event not like 'rdbms%'
         and event not like 'PX%'
         and event not like 'sbt%'
         and event not in ('gcs remote message','ges remote message',
                           'virtual circuit status','dispatcher timer') ;
select max(value) value from gv$sysstat where name ='CPU used when call started';
@title132 'RAC System Events Percent'
break on report
compute sum of time_waited on report
spool rep_out/&db/rac_sys_events
select    inst_id,
          name event,
          0 waits,
   0 average_wait,
   value time_waited,
   value/(&&divide_by+&&val)*100 Percent,
   value/&&millisec*100 p_of_total
from gv$sysstat
where name ='CPU used when call started'
union
```

```
select inst_id,
       event,
       total_waits-total_timeouts waits,
       time_waited/(total_waits-total_timeouts) average_wait,
       time_waited,
       time_waited/(&&divide_by+&&val)*100 Percent,
       time_waited/&&millisec*100 P_of_total
from gv$system_event
where total_waits-total_timeouts>0
    and event not like 'SQL*Net%'
    and event not like 'smon%'
    and event not like 'pmon%'
    and event not like 'rdbms%'
       and event not like 'PX%'
       and event not like 'sbt%'
       and event not in ('gcs remote message','ges remote message',
                         'virtual circuit status','dispatcher timer')
```

SEE GRID CODE DEPOT FOR DOWNLOAD (http://www.rampant-books.com/book_2004_1_10g_grid.htm)

Example results from the script above are shown below.

```
Date: 02/02/04                                                          Page:   1
Time: 01:51 PM                    RAC System Events Percent             TSTDBMRA
                                       tstdb database
```

INST_ID	Event Name	Total Waits	Average Waits	Time Waited	Percent Of Non-Idle Waits	Percent of Total Uptime
1	io done	222,168	2	532,399	7.930	.1285
1	CPU used when call started	0	0	360,648	5.372	.0870
1	imm op	168	1,812	304,377	4.533	.0735
1	control file parallel write	134,810	1	160,829	2.395	.0388
1	control file sequential read	748,737	0	106,655	1.589	.0257
1	i/o slave wait	377,955	0	99,104	1.476	.0239
1	enqueue	574,470	0	56,854	.847	.0137
1	IPC send completion sync	6,328	7	44,580	.664	.0108
1	wait for master scn	272,879	0	25,184	.375	.0061
1	DFS lock handle	65,619	0	18,470	.275	.0045
1	library cache pin	2,027	8	16,750	.249	.0040
1	db file sequential read	56,356	0	10,377	.155	.0025
1	name-service call wait	190	49	9,280	.138	.0022
1	direct path read	119,524	0	9,210	.137	.0022
1	log file parallel write	68,692	0	7,989	.119	.0019
1	global cache cr request	71,664	0	7,130	.106	.0017
1	process startup	145	35	5,112	.076	.0012
1	async disk IO	497,496	0	3,636	.054	.0009
1	db file scattered read	3,749	0	1,738	.026	.0004
1	switch logfile command	17	82	1,399	.021	.0003
2	CPU used when call started	0	0	625,945	9.323	.1511
2	control file parallel write	134,052	1	155,664	2.318	.0376
2	enqueue	1,146,971	0	149,334	2.224	.0360
2	control file sequential read	736,589	0	89,883	1.339	.0217
2	wait for master scn	274,211	0	24,081	.359	.0058
2	global cache cr request	308,585	0	21,361	.318	.0052
2	DFS lock handle	70,138	0	16,284	.243	.0039
2	db file sequential read	78,344	0	16,000	.238	.0039
2	log file parallel write	70,637	0	9,560	.142	.0023
2	db file scattered read	50,454	0	8,247	.123	.0020
2	IPC send completion sync	59,587	0	5,567	.083	.0013
2	name-service call wait	97	53	5,116	.076	.0012
2	direct path read	67,032	0	4,462	.066	.0011
2	process startup	68	43	2,904	.043	.0007
2	CGS wait for IPC msg	4,344	0	1,632	.024	.0004
2	library cache pin	3,939	0	1,384	.021	.0003
2	db file parallel read	3,664	0	789	.012	.0002
2	log file sequential read	71	11	757	.011	.0002
2	row cache lock	8,193	0	649	.010	.0002
3	CPU used when call started	0	0	3,171,613	47.238	.7655
3	db file sequential read	3,838,010	0	571,051	8.505	.1378
3	global cache cr request	2,670,668	0	388,165	5.781	.0937
3	control file parallel write	134,107	1	177,376	2.642	.0428
3	library cache pin	11,677	12	142,391	2.121	.0344
3	control file sequential read	979,741	0	122,439	1.824	.0296
3	IPC send completion sync	2,378	20	47,029	.700	.0114
3	db file scattered read	123,285	0	27,301	.407	.0066
3	global cache busy	257	105	27,044	.403	.0065
3	direct path read	135,560	0	23,154	.345	.0056
3	DFS lock handle	75,839	0	18,137	.270	.0044
3	name-service call wait	197	49	9,683	.144	.0023
3	log file parallel write	84,689	0	9,356	.139	.0023
3	latch free	1,983	4	7,881	.117	.0019

```
         3 process startup                      127          48        6,037          .090     .0015
         3 global cache s to x                26,158           0        3,521          .052     .0008
         3 global cache open x                20,776           0        3,452          .051     .0008
         3 row cache lock                     28,131           0        2,916          .043     .0007
         3 log file sequential read              654           4        2,541          .038     .0006
         3 pipe get                              125          19        2,420          .036     .0006
                                                                   ------------
sum                                                                 7,700,701
```

The above example has been reduced to 20 events per node to make displaying the report easier. Node three is using more CPU cycles than the other nodes and is also using more *db file reads* and more *global cache cr request* waits. At this point, node three should be reviewed for bad SQL.

Conclusion

Quite a bit of territory has been covered on RAC tuning in this chapter. The following are a few general guidelines:

- The cluster interconnect network should be tuned to get an optimal transfer rate and buffering.
- RAC objects should be tuned for:
  - *initrans*.
  - Freelists.
  - Rows per block (pctfree).
- Freelists and *init_trans* should be used whenever possible.
- Larger shared pools should be used rather than those used with normal Oracle, to allow for the Global Directory.
- Cache transfer speeds and intra-instance pings should be monitored to determine latencies.
- Holistic monitoring should be performed at all times

Using these guidelines, and the scripts and techniques covered in this chapter, the RAC environment can be made to perform optimally.

References

Oracle Real Application Clusters Administrator's Guide 10g, Release 1 (10.1), Part No. B10765-01, December 2003

Oracle Real Application Clusters Deployment and Performance Guide 10g, Release 1 (10.1), Part No. B10768-01, December 2003

This is an excerpt from Oracle 10g Grid & Real Application Clusters (http://www.rampant-books.com/book_2004_1_10g_grid.htm), by Mike Ault and Madhu Tumma.

Oracle10g Grid and RAC is the first-of-its-kind reference for Oracle Grid computing. Covering all areas of Oracle Grid computing, this book is indispensable for any Oracle DBA who is charged with configuring and implementing Oracle10g Grid with server blades.

This text presents a complete guide to the installation, configuration and design of Oracle Grid and 10g RAC. It supplies expert internals of shared disk technology, raw devices and RAID, and exposes the internal configuration methods for server blades. The text also demonstrates the use of Oracle Grid using the Enterprise Manager Grid control utility.