9

Performance Tuning

Performance tuning of any application, including the database, is an iterative process. This means that to maintain a healthy database, one must constantly monitor and fine-tune it. During certain periods, an aggressive performance tuning of both the application and database may be required. At other times, only routine continuous monitoring and maintenance may be needed. During this time, system hiccups may be discovered and solutions tried and tested.

The goal of a DBA or the application developer is to provide efficient, well-performing applications with good response time. In order for the application to provide a good response, the system, database, and SQL queries should be well tuned. Systems are tuned based on data collected during periods of poor performance; the evidence and the data collected may provide an indication of where the actual problem resides. For continuous monitoring and tuning of systems, a process or method should be adopted that helps streamline the activity. As in most repeatable situations, a methodology should be adopted, and once it has been validated and approved, it needs to be practiced. This methodology should be iterated every time there is a need to tune the system.

In this chapter, we will look into a scientific approach to troubleshooting, performance tuning, and maintaining a healthy database system. Tuning a RAC implementation has many aspects, and the techniques will vary depending on whether the RAC cluster is preproduction or live. Since a RAC configuration comprises one or more instances connected to a shared database, tuning a RAC configuration ideally starts with tuning the individual instances prior to the deployment of the production cluster. Individual instances in the cluster should be tuned using the same techniques used for single-instance databases. Once the individual instances are tuned, the other tiers, network, interconnect, cluster manager, and so on, should be incorporated into the tuning process.

9.1 Methodology

Problem-solving tasks of any nature need to be approached in a systematic and controlled manner. There needs to be a defined procedure or an action plan, and this procedure needs to be followed step by step from start to finish. During every step of the process, data is collected and analyzed, and the results are fed into the next step, which in turn is performed using a similar systematic approach. Hence, methodology is the procedure or process followed from start to finish, from identification of the problem to problem solving and documentation. A methodology is a procedure or process that is repeatable as a whole or in increments through iterations. During all of this analysis, the cause or reasons for a behavior or problem should be based on quantitative analysis and not on guesswork.

The performance tuning methodology can be broadly categorized into seven steps:

1. *Problem statement.* Identify or state the specific problem in hand (e.g., poor response time or poorly performing SQL statement).
2. *Information gathering.* Gather all information relating to the problem identified in step one. For example, when a user complains of poor performance, it may be a good idea to interview him or her to identify what kind of function the user was performing and at what time of the day (there may have been another contending application at that time, which may have caused the slow performance).
3. *Area identification.* Once the information concerning the performance issue is gathered, the next step is to identify the area of the performance issue. For example, the module in the application that belongs to a specific service type may be causing the performance issue.
4. *Area drilldown.* Drill down further to identify the cause or area of the performance issue. For example, identify the SQL statement or the batch application running at the wrong time of day.
5. *Problem resolution.* Work to resolve the performance issue (e.g., tune the SQL query).
6. *Testing against baseline.* Test to see if the performance issue has been resolved. For example, request that the user who complained test the performance.

7. *Repeating the process.* Now that the identified problem has been resolved, attempt to use the same process with the next problem.

While each of these steps is very broad, a methodical approach will help identify and solve the problem in question, namely, performance. Which area of the system is having a performance problem? Where do we start? Should the tuning process start with the operating system, network, database, instance, or application? Often the users of the application tier complain that the system has a poor response time. Users access an application, and the application in turn communicates with the database to store and retrieve information. When the user who made the request does not get the response in a sufficiently small amount of time, he or she complains that the system is slow.

Starting with poor end user response time may assist in tuning a system in production, but in other scenarios, we may need to tune bottom up (e.g., starting with the hardware platform, tuning the storage subsystem, tuning the database configuration, tuning the instance). Addressing the performance issues using this approach can bring some amount of change or performance improvement to the system with less or no impact on the actual application code. However, if the application is poorly written (e.g., a bad SQL query), tuning the underlying layers will have only a marginal effect.

As a general rule, it is more effective to take a "top-down" approach to database tuning since improvements in the upper layers (e.g., the application) will change the demand experienced by the lower layers (such as the storage system). When an application SQL is poorly tuned, it may cause excessive physical I/O demand, which in turn leads to poor disk service times. Tuning the SQL will both reduce the demand and eliminate the problems at all layers of the application. On the other hand, improving the performance of poorly tuned SQL by optimizing the storage subsystem, perhaps by buying more spindles, is a relatively expensive and ultimately ineffective measure. You also risk the embarrassing situation of having requested expensive hardware upgrades, which are later rendered unnecessary by the creation of an index or the addition of a hint to a problematic SQL.

Therefore, it is usually wise to perform tuning activities in the following order:

1. Tune the application, focusing on reducing its demand for database services. Primarily, this is done through SQL tuning, addi-

tion of indexes, rewording of SQLs or application-level caching. The observable outcome of this stage is a reduction in the rate of logical I/O demands (consistent reads and db (database) block reads) by the application.

2. Eliminate any contention for shared resources, such as locks, latches, freelists, and so on. Contention for these resources may be preventing the application from exhibiting its full demand for database services.

3. Use memory to minimize the amount of logical demand that turns into physical disk I/Os. This involves tuning the buffer cache to maximize the number of blocks of data that can be found in memory and tuning the `PGA_AGGREGATE_TARGET` to minimize I/O resulting from disk sorts and hash joins.

4. Finally, when the physical I/O demand has been minimized, distribute the load as evenly as possible across your disk spindles, and if there are insufficient spindles for the I/O demand you are now observing, add additional disk devices to improve overall I/O bandwidth.

The top-down or bottom-up methodology discussed previously is good for an already existing production application that needs to be tuned. Typically, we find an application's performance has degraded over time, possibly because (1) applications have degraded in performance due to new functionality that was not sufficiently tuned; (2) the user base has increased and the current application does not support the extended user base; and (3) the volume of data in the underlying database has increased, but the storage has not changed to accept the increased I/O load.

While these are issues with an existing application and database residing on existing hardware, a more detailed testing and tuning methodology should be adopted when migrating from a single instance to a clustered database environment. Before migrating the actual application and production enabling the new hardware, the following basic testing procedure should be adopted.

As mentioned earlier, testing of the RAC environment should start with tuning a single-instance configuration. Only when the performance characteristics of the application are satisfactory should tuning on the clustered configuration begin. To perform these tests, all nodes in the cluster except one should be shut down, and the single-instance node should be tuned. Only after the single instance has been tuned and appropriate performance

measurements equal to the current configuration or more are obtained should the next step of tuning be started. Tuning the cluster should be performed by adding one instance at a time to the mix. Performance should be measured in detail to ensure that the expected scalability and availability are obtained. If such performance measurements are not obtained, the application should not be deployed into production, and only after the problem areas are identified and tuned should deployment occur.

Note: RAC cannot magically bring performance improvements to an application that is already performing poorly on a single-instance configuration.

Caution: The rule of thumb is if the application cannot scale on a single-instance configuration when the number of CPUs on the server is increased from two to four to eight, the application will not scale in a RAC environment. Indeed, migrating to RAC can conceivably diminish performance of an application that cannot scale in an SMP (multi-CPU) environment.

While running performance tests on the instances by adding one node at a time to the cluster, the following phases should be included in the testing plan:

1. *Load Testing Phase I*. During this phase, a standard performance benchmarking software load will test the database environment, not including the application schemas and data. The purpose of this test is to verify the database and operating system performance characteristics. Based on the load tests and the statistics collected, the database and environment should be tuned. Once tuned, the testing should be repeated until a maximum or until such point when no or minimal performance gains are noticed. Load-testing tools such as Benchmark Factory (BMF), illustrated in Figure 9.1, or free-load testing tools such as Swingbench[1] or Hammerora[2] provide standard test categories and can be used to test the environment during this phase.

1. The latest version of the Swingbench software can be downloaded from www.dominicgiles.com.
2. The latest version of the Hammerora software can be downloaded from http://hammerora.sourceforge.net.

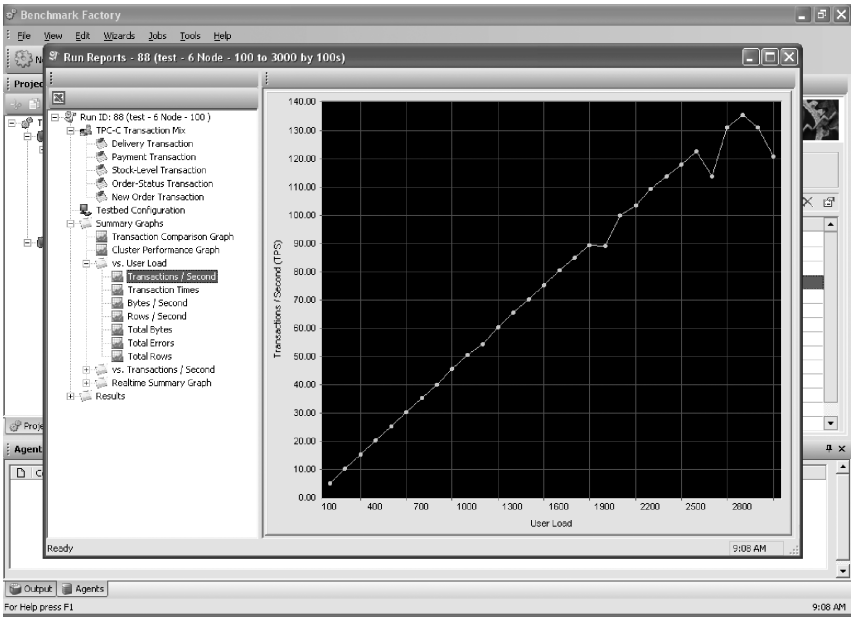


Figure 9.1
Benchmark Factory (BMF)

Once a stable environment has been reached, the next step is to test all failure points because, after all, one primary reason to migrate to a clustered environment is availability.

2. *Availability test*. During this step of testing, the various failure points will have to be tested to ensure that the RAC database will continue to function either as a single instance or as a cluster, depending on where the failure has occurred. For example, from where the node failure occurred, the remaining nodes in the cluster should continue to function. Similarly, when a network switches to the storage array and fails, the redundant switch should continue to operate. Tests should be performed during load, meaning that failures should be simulated, considering that they can happen in live production environments with user activity.

Note: This is a critical test and should not be compromised. All failure points should be tested until the expected results are achieved.

3. *Load Testing Phase II*. During this step, a load test should be performed against a production schema (on the new hardware plat-

form) that contains a copy of the actual data from the current live production environment. The purpose of this test is to tune the instance and the database for application workloads not interfacing with the business application. For such a test, an extract of the SQL queries from the application can be used. One method to extract these queries from a live system without user intervention is to extract them using Oracle event 10046 and parsing the trace files generated through an application to extract the queries with their respective bind values. Sample steps to complete phase II are as follows:

a. In a live production environment, enable Oracle Event Trace 10046 at level 4 after connecting to the server as user sys.

```
ALTER SYSTEM SET EVENTS '10046 TRACE NAME CONTEXT
FOREVER, LEVEL 4';
```

This command generates a trace file in the directory identified by the parameter `USER_DUMP_DEST`.

Caution: Depending on the activity on the production servers, the number of trace files and their contents could be large and consume a considerable amount of disk space. Please ensure sufficient disk space is available before attempting this step.

b. Concatenate all the trace files generated by the event in the user dump destination directory into one file.

```
cat *.trc  > SQLQueries.trc
```

c. Using parsing software (sample Perl script provided in Appendix B), replace the bind variables with bind values found in the trace file.

d. Using the queries extracted from step c, perform a load test simulating the estimated user workload iterating the queries and measuring response times. Remember, this step is also an iterative process, which means that the user load should be gradually increased through iterations, and during each iteration, statistics should be collected. Then,

based on the analysis, the instance and database parameters and, most importantly, the SQL queries should be tuned. This test can be performed using either a homegrown tool or third-party software such as BMF or hammerora.

Note: Performance should be monitored on all the tiers of the database server (i.e., operating system, instance, and database) during both load-testing phases using various performance-monitoring tools, which are discussed later in this chapter along with other performance-tuning methods.

Once the database layer has been tested and tuned simulating user work behavior, the next step is to perform an actual user acceptance test.

4. *User acceptance testing*. In this step, an organized set of users is requested to perform day-to-day operations using the standard application interface against the new environment. During this test phase, the database environment should be monitored, and data should be collected and analyzed and the environments tuned. With this step, almost all problem areas of the new environment should be identified and fixed.
5. *Day-in-a life test*. This is the final test where the environment is put through an actual user test by the application users simulating a typical business day.

Through these various stages of testing, all problem areas should be identified and fixed before going live into a production environment. Please note that RAC will not perform any miracles to improve the performance of the application. All applications that do not scale on a single-instance database environment will not scale in a clustered environment. Therefore, it is important to ensure that the application is performing well in the clustered environment during these testing cycles before going live.

Note: One of the most common failures during preproduction benchmarking is failure to simulate expected table data volumes. Many SQL statements will increase their I/O requirements and elapsed times as table volumes increase. In the case of a full table scan, the relationship will be approximately linear: if you double the size of the table, you double the SQLs' I/O and elapsed time. Indexed-based queries may show better scal-

ability, though many indexed queries will perform range scans that will grow in size with the number of rows in the underlying table. Therefore, a valid benchmark will use a database in which the long-term row counts in key tables have been simulated.

Identification and tuning of the database depends on the type of application, the type of user access patterns, the size of the database, the operating system, and so on. In the next sections of this chapter, the various tuning areas and options are discussed.

9.2 Storage subsystem

Shared storage in a RAC environment is a critical component of the overall architecture. Seldom is importance given to the storage system relative to the size of the database, the number of nodes in the cluster, and so on. Common problems found among customers are as follows:

- *When increasing the number of nodes participating in the cluster, seldom is any thought given to the number of nodes versus the number of interfaces to the storage subsystem and the capacity of the I/O path*. Due to limitations of the hardware, it has been observed on several occasions that the number of slots for the host bus adapter (HBA) and the network interface card (NIC) is insufficient to provide a good I/O capacity, and so is the number of ports on a switch and the number of controllers in a disk array. Care should be taken to ensure that the number of HBAs is equal to the number of disk controllers. Using the disk controller slots to accommodate more disk arrays will have a negative impact on the total throughput.

  For example, on a 16-port fiber channel switch, the ideal configuration is have eight HBAs and eight disk controllers, giving a total throughput of 8 × 200 MB = 1.6 GB/sec.[3] Now, if the number of HBAs is reduced to four to provide room for additional storage, then the total throughput drops by 50% (4 × 200 MB = 800 MB/sec).

  Another area of concern is tuning the operating system to handle the I/O workload. For example, Figure 9.2 is an output from an EM database console that illustrates a high I/O activity against the storage array.

3. Assuming the maximum theoretical payload of 2 Gb/s Fiber Channel is 200 MB/sec.

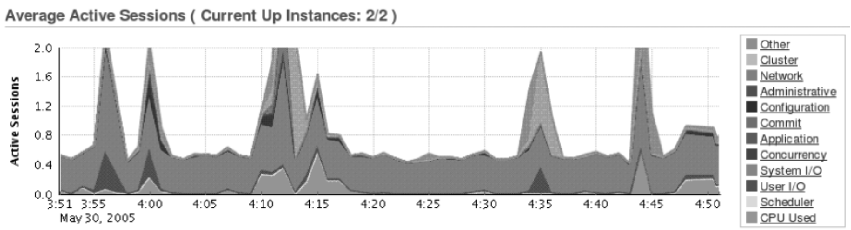


Figure 9.2
EM Active Session Showing High I/O Activity

Apart from poorly written SQL queries, high I/O activity against the storage system can occur for a number of reasons:

- Bad configuration of the SAN
- Low disk throughput
- High contention against the storage area
- Bad I/O channel
- High queue lengths

The storage system should be verified beforehand to ensure that all disks in the storage array are of high-performing capacity. While it may be difficult to have the entire storage array contain disks of the same performance characteristics, care should be taken to ensure that disks within a disk group (in the case of ASM) or volume group (in the case of third-party volume managers) are of identical capacity and performance characteristics because a poor-performing disk in a disk group can create inconsistent I/O activity. When using ASM, performance characteristics of the individual disks within a disk group can be monitored using EM, as illustrated in Figure 9.3.

- *Disk I/O can also be improved by configuring Oracle to use asynchronous I/O*. Asynchronous I/O (AIO) can be enabled by installing the following operating system-specific patches:

```
[root@oradb3 root]$ rpm -ivf libaio-0.3.96-3.i386.rpm
[root@oradb3 root]$ rpm -i-f libaio-devel-0.3.96-3.i386.rpm
```

Then, recompile the Oracle kernel using the following commands:

```
make -f ins_rdbms.mk async_on
make -f ins_rdbms.mk oracle
```

Subsequently, the following two parameters have to be set to the appropriate values:

```
DISK_ASYNCH_IO = TRUE  (default)
FILESYSTEMIO_OPTIONS=ASYNCH
```

In a RAC environment, by sharing blocks between instances using the cluster interconnect, Oracle will avoid physical I/O if possible. However, if it must be done to force the data to be saved to disks, then the goal should be to make I/O asynchronous and to eliminate random I/O operations because `DBWRn` processes often have to write out large batches of "dirty" blocks to disk. If AIO is not available, you may see "free buffer" or "write complete" waits as sessions wait for the `DBWRn` to catch up with the changes made by user sessions.

Note: Oracle Wait Interface (OWI) is discussed later in this chapter.

AIO allows a process to submit I/O requests without waiting for their completion. Enabling this feature allows Oracle to submit AIO requests, and while the I/O request is being processed, Oracle can pick up another thread and schedule a new I/O operation.

Note: In Oracle Database 10*g* Release 2, during installation Oracle will compile the kernel using the asynch parameters if the appropriate operating system packages are installed. AIO is not supported for NFS servers.

- *Poor performance in Linux environments, particularly with OLAP queries, parallel queries, backup and restore operations, or queries that perform large I/O operations, can be due to inappropriate setting of certain operating system parameters*. For example, by default on Linux environments, large I/O operations are broken into 32K-segment chunks, separating system I/O operations into smaller sizes. To allow Oracle to perform large I/O operations, certain default values at the operating system level should be configured appropriately. The following

steps will help users identify the current parameter settings and make appropriate changes:

1. Verify if the following parameters have been configured:

```
# cat /proc/sys/fs/superbh-behavior
# cat /proc/sys/fs/aio-max-size
# cat /proc/sys/fs/aio-max-nr
# cat /proc/sys/fs/aio-nr
```

aio-max-size

The `aio-max-size` parameter specifies the maximum block size that one single AIO write/read can do. Using the default value of 128K will chunk the AIO done by Oracle.

aio-nr and aio-max-nr

`aio-nr` is the running total of the number of events specified on the `io_setup` system call for all currently active AIO contexts. If `aio-nr` reaches `aio-max-nr`, then `io_setup` will fail. `aio-nr` shows the current systemwide number of AIO requests. `aio-max-nr` allows you to change the maximum value `aio-nr` can increase to.

Increasing the value of the `aio-max-size` to 1,048,576 and `aio_max_ns` parameters to 56K also helps the performance of the ASM disks because ASM performs I/O in 1-MB chunks.

2. Update the parameters by adding the following lines to the `/etc/sysctl.conf file`:

```
fs.superbh-behavior = 2
fs.aio-max-size = 1048576
fs.aio-max-nr = 512
```

This change will set these kernel parameters across reboots. To change them dynamically on a running system, issue the following commands as user `root`:

```
# echo 2 > /proc/sys/fs/superbh-behavior
# echo 1048576 > /proc/sys/fs/aio-max-size
# echo 512 > /proc/sys/fs/aio-max-nr
```

- *When configuring disk groups or volume groups, care should be taken in identifying disks of the same performance characteristics*. Such verification can be done using either the simple `dd` command or any disk calibration tool, such as Orion,[4] for example;

```
dd bs=1048576 count=200 if=/dev/sdc  of=/dev/null
```

  This command will copy 200 blocks by reading one block at a time up to a maximum of 1,048,576 bytes from an input device and writing it to an output device. When testing disks for Oracle database, the block size should represent the Oracle block size times the value defined using the parameter `MULTI_BLOCK_READ_COUNT` to obtain optimal disk performance.

  The following is the description of the various options used with the `dd` command:

  - `bs=bytes`. Reads that many bytes of data at a time.
  - `count=blocks`. Copies the number of blocks specified by the count parameter.
  - `if=file`. Specifies the input file to read data from (e.g., a disk).
  - `of=file`. Specifies the output device of the file where the data will be written.

- *When testing disk performance characteristics, user concurrency should be considered from multiple nodes in a RAC environment*. User concurrency can also be simulated by running multiple `dd` commands. By using standard operating system commands such as `vmstat`, the concurrency level can be increased gradually to determine the highest throughput rate and beyond where there is a point of zero increase.
- *Selection of disk volume managers also plays an important part in the overall performance of the database*. This is where the use of ASM comes into play. Deploying databases on ASM will help in automatic distribution of files based on the same methodology, and Oracle will perform the automatic placement of files based on the importance of data.

4. Orion can be downloaded from the Oracle technology network at http://otn.oracle.com.

9.3 Automatic Storage Management

Above, we briefly touched on tuning the operating system to help improve the I/O subsystem. ASM performs placement of files across various disks automatically; however, ASM cannot improve the performance of existing poorly performing disks. We discussed earlier that it would be ideal to have all disks in a disk group with the same performance characteristics to

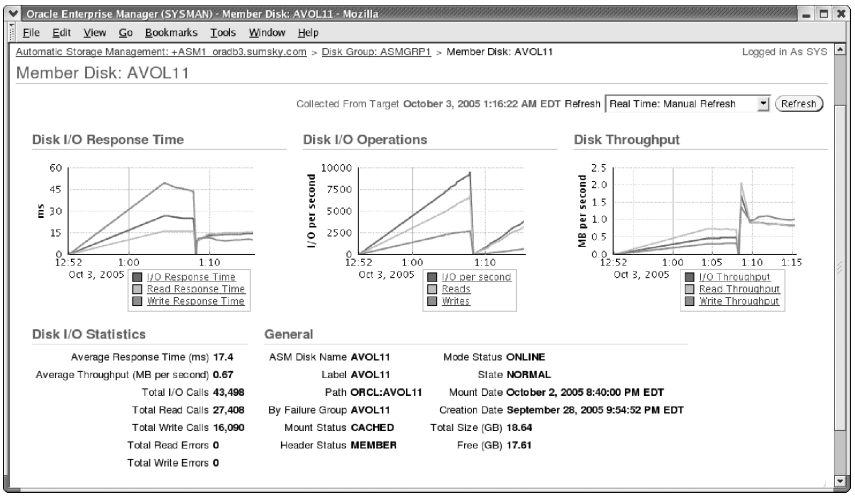


Figure 9.3 *I/O Performance at the ASM Disk Level*

provide consistent performance. The performance characteristics of individual disks illustrated in Figure 9.3 within a disk group can be monitored using EM.

In Chapter 3, we discussed how an ASM instance and an RDBMS instance will interact for various reasons. During this process of communication and during the various administrative functions performed by ASM on the disk groups, ASM will require resources. Like in a RDBMS instance, despite ASM being a lightweight instance, it also contains an SGA. For example, the default SGA is

```
SQL> show sga

Total System Global Area   92274688 bytes
Fixed Size                  1217884 bytes
Variable Size              65890980 bytes
ASM Cache                  25165824 bytes
```

Note: The ASM cache is defined by the `DB_CACHE_SIZE` parameter.

Note: The SGA is broken into the shared pool, large pool, and shared pool reserved size. Default values for these parameters are:

```
SHARED_POOL_SIZE = 48M
LARGE_POOL_SIZE = 12M
SHARED_POOL_RESERVED_SIZE = 24M
SGA_MAX_SIZE = 88M
```

The SGA for the ASM instance is sized very small. Based on the number of instances or databases communicating with the ASM instance, usually, the default SGA is sufficient. However, when the application performs high I/O activity or when the ASM instance supports more than six Oracle instances, adding resources to the ASM instance is helpful to improve performance (e.g., increasing the `LARGE_POOL_SIZE` to help in the communications between ASM and its clients) [27]. ASM and its functionality are discussed extensively in Chapter 3.

9.4 Cluster interconnect

This is a very important component of the clustered configuration. Oracle depends on the cluster interconnect for movement of data between the instances. Chapter 2 provides a detailed explanation of how global data movement occurs.

Testing the cluster interconnect should start with a test of the hardware configuration. This basic test should ensure that the database is using the correct IP addresses or NICs for the interconnect. The following query provides a list of IP addresses registered with Oracle:

```
COL PICKED_KSXPIA FORMAT A15
COL INDX FORMAT 99999
SELECT * FROM X$KSXPIA;

ADDR     INDX INST_ID PUB_KSXPIA PICKED_KSXPIA   NAME_KSXPIA  IP_KSXPIA
-------- ---- ------- ---------- --------------- ------------ -------------
3FE47C74    0       1 N          OCR             bond1        10.168.2.130
3FE47C74    1       1 Y          OCR             bond0        192.168.2.30
```

In the output, `bond0` is the public interface (identified by the value `Y` in column `PUB_KSXPIA`), and `bond1` is the private interface (identified by the value `N` in column `PUB_KSXPIA`). If the correct IP addresses are not visible, this indicates incorrect installation and configuration of the RAC environment.

Column `PICKED_KSXPIA` indicates the type of clusterware implemented on the RAC cluster, where the interconnect configuration is stored, and the cluster communication method that RAC will use. The valid values in this column are

- OCR. Oracle Clusterware is configured.
- OSD. It is operating system dependent, meaning a third-party cluster manager is configured, and Oracle Clusterware is only a bridge between Oracle RDBMS and the third-party cluster manager.
- CI. The interconnect is defined using the `CLUSTER_INTERCONNECT` parameter in the instance.

Alternatively interconnect information registered by all participating nodes in the cluster can be verified from `GV$CLUSTER_INTERCONNECTS` view. Cluster interconnect can also be verified by using the `ORADEBUG` utility (discussed later) and verifying the trace file for the appropriate IP address.

Note: Failure to keep the interconnect interfaces private will result in the instances' competing with other network processes when requesting blocks from other cluster members. The network between the instances needs to be dedicated to cluster coordination and not used for any other purpose.

`CLUSTER_INTERCONNECTS`

This parameter provides Oracle with information on the availability of additional cluster interconnects that can be used for cache fusion activity. The parameter overrides the default interconnect settings at the operating system level with a preferred cluster traffic network. While this parameter does provide certain advantages over systems where high interconnect latency is noticed by helping reduce such latency, configuring this parameter can affect the interconnect high-availability feature. In other words, an interconnect failure that is normally unnoticeable will instead cause an Oracle cluster failure as Oracle still attempts to access the network interface.

Best Practice: NIC pairing/bonding should be a preferred method to using the `CLUSTER_INTERCONNECTS` parameter to provide load-balancing and failover of the interconnects.

9.5 Interconnect transfer rate

The next important verification is to determine the transfer rate versus the actual implemented packet size to ensure the installation has been carried out per specification.

The speed of the cluster interconnect depends solely on the hardware vendor and the layered operating system. Oracle depends on the operating system and the hardware for sending packets of information across the cluster interconnect. For example, one type of cluster interconnect supported in Sun 4800s is the UDP protocol. However, Solaris in this specific version has an operating system limitation of a 64-KB packet size for data transfer. To transfer 256 KB worth of data across this interconnect protocol would take more than four round trips. Comparing this to another operating system (e.g., Linux), the maximum supported packet size is 256K. On a high-transaction system where there is a large amount of interconnect traffic, because of user activity on the various instances participating in the clustered configuration, limitations on the packet size can cause serious performance issues.

Tools such as `IPtraf` on Linux environments (Figure 9.4) or `glance` on HP-UX environments or utilities such as `netstat` should help monitor network traffic and transfer rates between instance and client configurations.

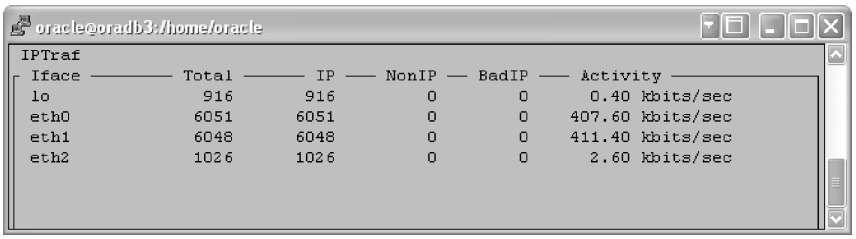
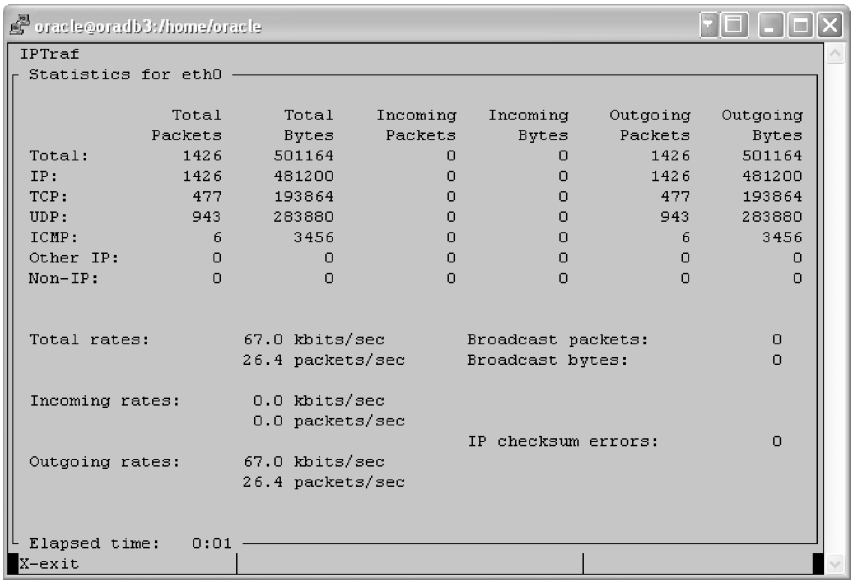


Figure 9.4 *IPTraf General Network Traffic*

`IPTraf` also helps to look into a specific network and monitor its performance in detail by the type of protocol used for network traffic. For example, in Figure 9.5, network traffic by protocol (TCP and UDP) is displayed, giving outgoing and incoming rates.

Figure 9.5 *IPTraf Statistics for eth0*



After the initial hardware and operating-system-level tests to confirm the packet size across the interconnect, subsequent tests could be done from the Oracle database to ensure that there is not any significant added latency from using cache-to-cache data transfer or the cache fusion technology. The query below provides the average time to receive a consistent read (CR) block on the system:

```
set numwidth 20
column "AVG CR BLOCK RECEIVE TIME (ms)" format 9999999.9
select
      b1.inst_id,
      b2.value "GCS CR BLOCKS RECEIVED",
      b1.value "GCS CR BLOCK RECEIVE TIME",
      ((b1.value / b2.value) * 10) "AVG CR BLOCK RECEIVE TIME (ms)"
from     gv$sysstat b1,
gv$sysstat b2
where b1.name = 'gc cr block receive time'
and      b2.name = 'gc cr blocks received'
and      b1.inst_id = b2.inst_id ;

INST_ID GCS CR BLOCKS RECEIVED GCS CR BLOCK RECEIVE TIME AVG CR BLOCK RECEIVE TIME (ms)
------- ---------------------- ------------------------ ------------------------------
      1                   2758                   112394                         443.78
      2                   1346                     1457                           10.8
```

Note: The data in the GV$SYSSTAT view is cumulative since the last time the Oracle instance was bounced. This does not reflect the true performance of the interconnect or give a true picture of the latency in transferring data. To get a more realistic picture of the performance, it would be good to bounce all of the Oracle instances and test again.

In the output above, it can be noticed that the AVG CR BLOCK RECEIVE TIME for instance 1 is 443.78 ms; this is significantly high when the expected average latency as recommended by Oracle should not exceed 15 ms. A high value is possible if the CPU has limited idle time, and the system typically processes long-running queries. However, it is possible to have an average latency of less than 1 ms with user-mode IPC. Latency can also be influenced by a high value for the DB_MULTI_BLOCK_READ_COUNT parameter. This is because this parameter determines the size of the block that each instance would request from the other during read transfers. and a requesting process can issue more than one request for a block depending on the setting of this parameter and may have to wait longer. This kind of high latency requires further investigation of the cluster interconnect configuration, and tests should be performed at the operating system level to ensure this is not something from Oracle or the parameter.

Note: Sizing of the DB_MULTI_BLOCK_READ_COUNT parameter should be based on the interconnect latency and the packet sizes as defined by the hardware vendor, and after considering the operating system limitations.

If the network interconnect is correctly configured as outlined earlier, then it is unlikely that the interconnect itself will be responsible for high receive times as revealed by GV$SYSSTAT. The actual time taken to transfer a block across the interconnect hardware will normally be only a small fraction of the total time taken to request the block on the first instance, convert any locks that may exist on the block, prepare the block for transfer, verify the receipt of the block, and update the relevant global cache structures. So, while it is important to ensure that the interconnect hardware is correctly configured, it should not be concluded that the interconnect is misconfigured if it is determined that block transfers are slow.

The EM Cluster Cache Coherency screen (Figure 9.6) is also a tool to monitor cluster interconnect performance. The figure displays three important matrixes:

1. *Global cache block access latency*. This represents the elapsed time from when the block request was initiated until it finishes. However, when a database block of any class is unable to locate a buffered copy in the local cache, a global cache operation is initiated by checking if the block is present in another instance. If it is found, it is shipped to the requestor.
2. *Global cache block transfer rate*. If a logical read fails to find a copy of the buffer in the local cache, it attempts to find the buffer in the database cache of a remote instance. If the block is found, it is shipped to the requestor. The global cache block transfer rate indicates the number of blocks received.
3. *Block Access Statistics*. This indicates the number of blocks read and the number of blocks transferred between instances in a RAC cluster.

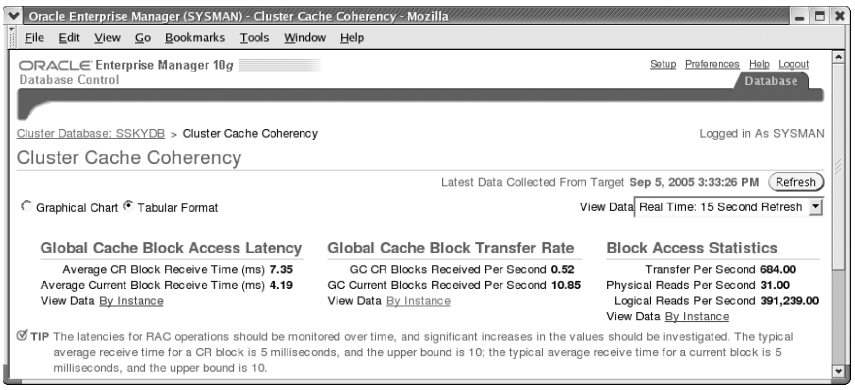


Figure 9.6 *EM Cluster Cache Coherency*

Latencies on the cluster interconnect can be caused by the following:

- No dedicated interconnect for cache fusion activity has been configured.
- A large number of processes in the run queues are waiting for CPU or as a result of processor scheduling delays.
- Incorrect platform-specific operating system parameter settings affect IPC buffering or process scheduling.
- Slow, busy, or faulty interconnects create slow performance.

One primary advantage of the clustered solution is to save on physical I/O against a storage system, which is expensive. This means that the latency of retrieving data across the interconnect should be significantly lower compared to getting the data from disk. For the overall performance of the cluster, the interconnect latency should be maintained at 6 to 8 ms. *The average latency of a consistent block request is the average latency of a consistent-read request round-trip from the requesting instance to the holding instance and back to the requesting instance.*

When such high latencies are experienced over the interconnect, another good test is to perform a test at the operating system level by checking the actual ping time. This will help to determine if there are any issues at the operating system level. After all, the performance issue may not be from data transfers within the RAC environment. Figure 9.7 (taken from Quest Software's Spotlight on RAC product) provides a comparison of the cluster latency versus the actual ping time monitored at the operating system level. This helps determine the latency encountered at the database level versus any overheads at the operating system level.

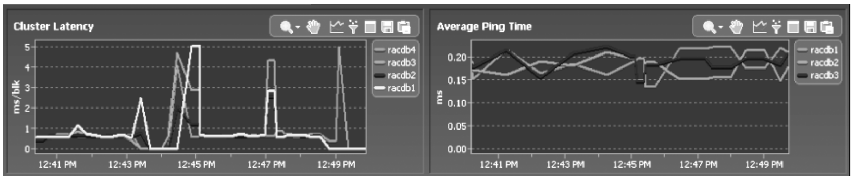


Figure 9.7 *Cluster Latency versus Average Ping Time*

Apart from the basic packet transfer tests that can be performed at the operating system level, other checks and tests can be done to ensure that the cluster interconnect has been configured correctly.

- There are redundant, private, high-speed interconnects between the nodes participating in the cluster. Implementing NIC bonding or pairing will help interconnect load-balancing and failover when one of the interconnects fails. The configuring of bonding or pairing of NICs is discussed in Chapter 4.
- The user network connection does not interfere with the cluster interconnect traffic (i.e., they are isolated from each other).

At the operating system level, the `netstat` and `ifconfig` commands display network-related data structures. The output below for `netstat -i`

indicates that there are four network adapters configured, and NIC pairing is implemented:

```
[oracle@oradb3 oracle]$ netstat -i
Kernel Interface table
Iface       MTU Met    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
bond0      1500   0     3209      0      0      0     4028      0      0      0 BMmRU
bond0:1    1500   0     4390      0      0      0     6437      0      0      0 BMmRU
bond1      1500   0     7880      0      0      0    10874      0      0      0 BMmRU
eth0       1500   0     1662      0      0      0     2006      0      0      0 BMsRU
eth1       1500   0     1547      0      0      0     2022      0      0      0 BMsRU
eth2       1500   0     4390      0      0      0     6437      0      0      0 BMRU
eth3       1500   0     3490      0      0      0     4437      0      0      0 BMRU
lo        16436   0     7491      0      0      0     7491      0      0      0 LRU
```

- `bond0` is the public interconnect created using the bonding functionality (bonds `eth0` and `eth1`).
- `bond0:1` is the VIP assigned to `bond0`.
- `bond1` is the private interconnect alias created using the bonding functionality (bonds `eth2` and `eth3`).
- `eth0` and `eth1` are the physical public interfaces; however, they are bonded/paired together (`bond0`).
- `eth2` and `eth3` are the physical private interfaces; however, they are bonded/paired together (`bond1`).
- `lo0` indicates that there is a loopback option configured. Verification of whether Oracle is using the loopback option should be made using the `ORADEBUG` command and is discussed later in this section. The use of the loopback IP depends on the integrity of the routing table defined on each of the nodes. Modification of the routing table can result in the inoperability of the interconnect.

In the `netstat` output above, MTU is set at 1,500 bytes. MTU definitions do not include the data-link header. However, packet size computations include data-link headers. Maximum packet size displayed by the various tools is MTU plus the data-link header length. To get the maximum benefit from the interconnect, MTU should be configured to the highest possible value supported. For example, a setting as high as 9K using jumbo frames will help improve interconnect bandwidth and data transmission. Jumbo frame configuration is covered in Chapter 4.

Checks can also be done from the Oracle instance to ensure proper configuration of the interconnect protocol. If the following commands

are executed as user **sys**, a trace file is generated in the user dump destination directory that contains certain diagnostic information concerning the UDP /IPC configurations:

```
SQL> ORADEBUG SETMYPID
       ORADEBUG IPC
       EXIT
```

The following is an extract from the trace file concerning IPC. The output confirms that the cluster interconnect is being used for instance-to-instance message transfer.

```
admno 0x4768d5f0 admport:
SSKGXPT 0xe453ec4 flags SSKGXPT_READPENDING     info for network 0
        socket no 7     IP 10.168.2.130         UDP 31938
        sflags SSKGXPT_UP
        info for network 1
        socket no 0     IP 0.0.0.0      UDP 0
        sflags SSKGXPT_DOWN
        active 0        actcnt 1
context timestamp 0
        no ports
```

Note:

- The above output protocol used is UDP. On certain operating systems, such as Tru64, the trace output does not reveal the cluster interconnect information.
- ASM in a cluster environment will also use the interconnect for its interinstance cache transfer activity. The same verification step can also be performed from the ASM instance to ensure that both are correct.

Both the RDBMS and ASM alert logs are another source for this information.

```
Cluster communication is configured to use the following interface(s) for this
instance
  10.168.2.130
Sun Oct  2 21:34:13 2005
cluster interconnect IPC version: Oracle UDP/IP
IPC Vendor 1 proto 2
```

Best Practice: Set the interconnect network parameters to the maximum allowed by the operating system.

9.6 SQL*Net tuning

Similar to the network buffer settings for the cluster interconnects, buffer sizes and network parameters for the public interface should also be considered during the performance optimization process.

Network delays in receiving user requests and sending data back to users affect the overall performance of the application and environment. Such delays translate into SQL*Net-related wait events (OWI is discussed later).

Like the MTU settings for the interconnects, the MTU settings for the public interface can also be set to jumbo frame sizes, provided the entire network stack starting with the origin of the user request to the database tier all support this configuration. If any one tier does not support jumbo frames, this means the entire network stepping down to the default configuration of 1,500 bytes.

Like the MTU settings, the Session Data Unit (SDU) settings for the SQL*Net connect descriptor can also be tuned. Optimal SDU settings can be determined by repeated data/buffer requests by enabling SQL*Net and listener trace at both the client and server levels.

SQL*Net tracing can be enabled by adding the following parameters to the `SQLNET.ora` file on the client machines located in the `$ORACLE_HOME/network/admin` directory:

```
trace_level_client=16
trace_file_client=client
trace_unique_client=true
trace_timestamp_client=ON
```

Listener tracing on the servers can be enabled by adding the following parameters to the `listener.ora` file located in the `$ORACLE_HOME/network/admin` directory:

```
trace_level_server=16
trace_file_server=server
trace_timestamp_server=ON
```

Trace files are generated in `$ORACLE_HOME/network/log` directories on the respective systems. The appropriate parameters should then be added to the connection descriptor on the client system. For example, the following SDU settings in the TNS connection descriptor will set the value of the SDU to 8K:

```
SSKYDB =
  (DESCRIPTION =
      (SDU = 8192)
      (FAILOVER = ON)
     (ADDRESS = (PROTOCOL = TCP)(HOST = oradb1-vip.sumsky.net)(PORT = 1521))
     (ADDRESS = (PROTOCOL = TCP)(HOST = oradb2-vip.sumsky.net)(PORT = 1521))
     (ADDRESS = (PROTOCOL = TCP)(HOST = oradb3-vip.sumsky.net)(PORT = 1521))
     (ADDRESS = (PROTOCOL = TCP)(HOST = oradb4-vip.sumsky.net)(PORT = 1521))
     (LOAD_BALANCE = YES)
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = SSKYDB)
      (FAILOVER_MODE =
         (TYPE = SELECT)(METHOD = BASIC)(RETRIES = 10)(DELAY = 3)
    )
   )
  )
```

Similar settings should also be applied to the listener to ensure that the bytes received by the server are also of a similar size. For example, the following SDU settings on the listener will set the receive value to 8K:

```
SID_LIST_LISTENER =
(SID_DESC =
      (SDU=8192)
      (SID_NAME = SSKY1)
    ))
```

9.6.1 Tuning network buffer sizes

As a basic installation and configuration requirement, network buffer size requirements were discussed in Chapter 4. These parameter values are the bare minimum required for RAC functioning. Continuous monitoring and

measuring of network latencies can help increase these buffer sizes even further, provided the operating system supports such an increase.

TCP uses a congestion window scheme to determine how many packets can be transmitted at any one time. The maximum congestion window size is determined by how much buffer space the kernel has allocated for each socket. If the buffers are too small, the TCP congestion window will never completely open; on the other hand, if the buffers are too large, the sender can overrun the receiver, causing the TCP window to shut down.

Apart from the `wmem_max` and `rmem_max` parameters discussed in Chapter 4, certain TCP parameters should also be tuned to improve TCP network performance.

`tcp_wmem`

This variable takes three different values, which hold information on how much TCP send buffer memory space each TCP socket has to use. Every TCP socket has this much buffer space to use before the buffer is filled up. Each of the three values is used under different conditions.

The first value in this variable sets the minimum TCP send buffer space available for a single TCP socket; the second value sets the default buffer space allowed for a single TCP socket to use; and the third value sets the kernel's maximum TCP send buffer space. The `/proc/sys/net/core/wmem_max` value overrides this value; hence, this value should always be smaller than that value.

`tcp_rmem`

The `tcp_rmem` variable is pretty much the same as `tcp_wmem` except in one large area: tells the kernel the TCP receive memory buffers instead of the transmit buffer, which is defined in `tcp_wmem`. This variable takes three different values, like the `tcp_wmem` variable.

`tcp_mem`

The `tcp_mem` variable defines how the TCP stack should behave when it comes to memory usage. It consists of three values, just like the `tcp_wmem` and `tcp_rmem` variables. The values are measured in memory pages (in short, pages). The size of each memory page differs depending on hardware and configuration options in the kernel, but on standard i386 computers, this is 4 KB, or 4,096 bytes. On some newer hardware, this is set to 16, 32, or even 64 KB. All of these values have no real default since they are calcu-

lated at boot time by the kernel and should, in most cases, be good for you and most usages you may encounter.

9.6.2 Device queue sizes

As with tuning the network buffer sizes, it is important to look into the size of the queue between the kernel network subsystems and the driver for the NIC. Inappropriate sizing can cause loss of data due to buffer overflows, which in turn causes retransmission, consumes resources, and delays performance.

There are two queues to consider in this area, the `txqueuelen`, which is related to the transmit queue size, and the `netdev_backlog`, which determines the receive queue size. These values can be manually defined using the `ifconfig` command on Linux and Unix systems. For example, the following command will reset the `txqueuelen` to 2,000:

```
/sbin/ifconfig eth0 txqueuelen 2000
```

Similarly, the receive queue size can be increased by setting the following parameter:

`/proc/sys/net/core/netdev_max_backlog = 2000` in the `/etc/sysctl.conf` file.

Note: Tuning the network should also be considered when implementing Standby or Streams solutions that involve movement of large volumes of data across the network to the remote location.

9.7 SQL tuning

Irrespective of having high-performing hardware, a high-performing storage subsystem, or an abundance of resources available on each of the nodes in the cluster, RAC cannot perform magic to help with poorly-performing queries. Actually, poorly-performing queries can be a serious issue when you move from a single-instance configuration to a clustered configuration. In certain cases, a negative impact on the overall performance of the system will be noticed. When tuning queries, be it in a single-instance configuration or a clustered configuration, the following should be verified and fixed.

9.7.1 Hard parses

Hard parses are very costly for the Oracle's optimizer. The amount of validation that has to be performed during a parse consumes a significant number of resources. The primary reason for a hard parse is the uniqueness of the queries present in the library cache or SGA. When a user or session executes a query, the query is parsed and loaded in the library cache after Oracle has generated a hash value for the query. Subsequently, when another session or user executes the same query, depending on the extent of its similarity to the query already present in the library cache, it is reused, and there is no parse operation involved. However, if it is a new query, it has to go through the Oracle parsing algorithm; this is considered a hard parse and is very costly. The total number of hard parses can be determined using the following query:

```
SELECT PA.INST_ID,
       PA.SID,
       PA.VALUE "Hard Parses",
       EX.VALUE "Execute Count"
FROM   GV$SESSTAT PA,
       GV$SESSTAT EX
WHERE  PA.SID=EX.SID
AND    PA.INST_ID=EX.INST_ID
AND    PA.STATISTIC#=(SELECT STATISTIC#
                      FROM   V$STATNAME
                      WHERE  NAME ='parse count (hard)')
AND    EX.STATISTIC#=(SELECT STATISTIC#
                      FROM   V$STATNAME
                      WHERE  NAME ='execute count')
AND    PA.VALUE > 0;
```

Besides when a query is executed for the first time, other reasons for hard parse operations are as follows:

1. *There is insufficient allocation of the SGA*. When numerous queries are executed, they have to be flushed out to give space for new ones. This repeated loading and unloading can create high hard parse operations. The number of reloads can be determined using the following query:

```
SELECT INST_ID,
```

```
SQL_TEXT,
LOADS
   FROM   GV$SQLSTATS
   WHERE  LOADS > 100;
```

The solution to this problem is to increase the size of the shared pool using the parameter `SHARED_POOL_SIZE`. The ideal configuration of the shared pool can be determined by querying the `V$SHARED_POOL_ADVICE` view.

2. *Queries that use literals in the `WHERE` clause, making every query executed unique to Oracle's optimizer, cause it to perform hard parse operations*. The solution to these issues is to use bind variables instead of hard-coded values in the queries. If the application code cannot be modified, the hard parse rate can be reduced by setting the parameter `CURSOR_SHARING` to `FORCE` (or `SIMILAR`). Furthermore, soft parse rates can be reduced by setting `SESSION_CACHED_CURSORS` to a nonzero value.

   Hard parsing should be minimized, largely to save on resources and make those resources available for other purposes.

V$/GV$SQLSTATS

This view provides the same information available in `V$SQL` and `V$SQLAREA`. However, accessing this view is much more cost-effective compared to the others. Accessing data from `GV$SQLSTATS` will not require the process to obtain any operating system latches and gives improved response times.

9.7.2 Logical reads

When data is read from physical storage (disk), it is placed into the buffer cache before filtering through the rows that match the criteria specified in the `WHERE` clause. Rows thus read are retained in the buffer, assuming other sessions executing similar queries may require the same data, reducing physical I/O. Queries not tuned to perform minimal I/O operations will retrieve a significantly larger number of rows, causing Oracle to traverse through the various rows, filtering what is not required instead of directly accessing rows that match. Such operations cause a significant amount of overhead and consume a large number of resources in the system.

Reading from buffer, or logical reads or logical I/O operations (LIO), is cheaper compared to reading data from disk. However, in Oracle's architecture, high LIOs are not cheap enough that they can be ignored because

when Oracle needs to read a row from buffer, it needs to place a lock on the row in buffer. To obtain a lock, Oracle has to request a latch from the operating system. Latches are not available in abundance. Often when a latch is requested, one is not immediately available because other processes are using them. When a latch is requested, the requesting process will go into a sleep mode and after a few nanoseconds, will wake up and request the latch again. This time it may or may not obtain the latch and may have to sleep again. These attempts to obtain a latch generally lead to high CPU consumption on the host and cache buffer chains latch contention as sessions fight for access to the same blocks. When Oracle has to scan a large number of rows in the buffer to retrieve only a few rows that meet the search criteria, this can prove costly.

SQLs that issue high logical read rates in comparison to the actual number of database rows processed are possible candidates for SQL tuning efforts. Often the introduction of a new index or the creation of a more selective index will reduce the number of blocks that must be examined in order to find the rows required. For example, let's examine the performance of the following query:

```
 SELECT eusr_id,
  us.usec_total_logins,
  eu.eusr_role_cd,
  c.comp_scac_code,
  eu.eusr_login_name,
  ul.usrli_id
FROM el_user eu, company c, user_login ul, user_security us
WHERE ul.USRLI_ACTIVE_STATUS_CD = 'Active'
  AND ul.USRLI_LOGGED_IN_EUSR_ID = eu.EUSR_ID
  AND eu.eusr_comp_id = c.comp_id
  AND eu.eusr_id = us.USEC_EUSR_ID
ORDER BY c.comp_comp_type_cd, c.comp_name, eu.eusr_last_name
```

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.28       0.29          0         51          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1     26.31      40.35      12866    6556373          0          87
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3     26.59      40.64      12866    6556373          0          87

Misses in library cache during parse: 1
```

```
Optimizer goal: CHOOSE
Parsing user id: 33  (MVALLATH)

Rows     Row Source Operation
-------  ---------------------------------------------------
     87  SORT ORDER BY (cr=3176 r=66 w=66 time=346886 us)
     87   TABLE ACCESS BY GLOBAL INDEX ROWID USER_SECURITY PARTITION: 1 1 (cr=3176 r=66 w=66
time=338109 us)
     78    NESTED LOOPS  (cr=3088 r=66 w=66 time=334551 us)
     90     NESTED LOOPS  (cr=2596 r=66 w=66 time=322337 us)
     90      NESTED LOOPS  (cr=1614 r=66 w=66 time=309393 us)
     90       VIEW  (cr=632 r=66 w=66 time=293827 us)
  48390        HASH JOIN  (cr=632 r=66 w=66 time=292465 us)
6556373         TABLE ACCESS FULL USER_LOGIN (cr=190 r=0 w=0 time=138776 us)(object id
24891)
    970         TABLE ACCESS FULL EL_USER (cr=442 r=0 w=0 time=56947 us)(object id 24706)
     90       INDEX UNIQUE SCAN PK_EUSR PARTITION: 1 1 (cr=492 r=0 w=0 time=6055 us)(object
id 24741)
     90      TABLE ACCESS BY LOCAL INDEX ROWID COMPANY PARTITION: 1 1 (cr=982 r=0 w=0
time=10135 us)
     90       INDEX UNIQUE SCAN PK_COMP PARTITION: 1 1 (cr=492 r=0 w=0 time=4905 us)(object
id 24813)
     87     INDEX RANGE SCAN USEC_INDX1 (cr=492 r=0 w=0 time=9115 us)(object id 24694)
```

In the tkprof output from a 10046 event trace, which, it should be noted, retrieves just 87 rows from the database, the SQL is processing a large number (6556373) rows from the USER_LOGIN table, and no index is being used to retrieve the data. Now, if an index is created on the USER_LOGIN table, the query performance improves several fold:

```
SQL> create index USRLI_INDX1 on USER_LOGIN(USRLI_ACTIVE_STATUS_CD);

Index created.

Rows     Row Source Operation
-------  ---------------------------------------------------
    487  SORT ORDER BY (cr=3176 r=66 w=66 time=346886 us)
    487   TABLE ACCESS BY GLOBAL INDEX ROWID USER_SECURITY PARTITION: 1 1 (cr=3176 r=66
w=66 time=338109 us)
    978    NESTED LOOPS  (cr=3088 r=66 w=66 time=334551 us)
    490     NESTED LOOPS  (cr=2596 r=66 w=66 time=322337 us)
    490      NESTED LOOPS  (cr=1614 r=66 w=66 time=309393 us)
    490       VIEW  (cr=632 r=66 w=66 time=293827 us)
    490        HASH JOIN  (cr=632 r=66 w=66 time=292465 us)
56373         INDEX FAST FULL SCAN USRLI_INDX1 (cr=190 r=0 w=0 time=947 us)(object id 28491)
    970         TABLE ACCESS FULL EL_USER (cr=442 r=0 w=0 time=947 us)(object id 24706)
    490       TABLE ACCESS BY LOCAL INDEX ROWID ELOGEX_USER PARTITION: 1 1 (cr=982 r=0 w=0
time=12238 us)
    490       INDEX UNIQUE SCAN PK_EUSR PARTITION: 1 1 (cr=492 r=0 w=0 time=6055 us)(object
id 24741)
```

```
    490      TABLE ACCESS BY LOCAL INDEX ROWID COMPANY PARTITION: 1 1 (cr=982 r=0 w=0
time=10135 us)
    490      INDEX UNIQUE SCAN PK_COMP PARTITION: 1 1 (cr=492 r=0 w=0 time=4905 us)(object
id 24813)
    487     INDEX RANGE SCAN USEC_INDX1 (cr=492 r=0 w=0 time=9115 us)(object id 24694)
```

The optimizer decides to use the new index `USRL_INDX1` and reduces the number of rows retrieved. Now, if another index is added to the `EL_USER` table, further improvement in the query can be obtained.

Indexes that are not selective do not improve query performance but can degrade DML performance. In RAC, unselective index blocks may be subject to interinstance contention, increasing the frequency of cache transfers for indexes belonging to `INSERT`-intensive tables.

9.7.3 SQL Advisory

Oracle's new SQL Advisory feature in EM is a good option for tuning SQL queries. Oracle analysis data gathered from real-time performance statistics uses this data to optimize the query performance. To use the SQL tuning advisory, select the "Advisory Central" option from the performance page from the db console or EM GC, then select the "SQL Tuning Advisory" option. This option provides the "Top Activity" page (Figure 9.8). Highlighting a specific time frame of the top activity will yield the "Top SQL" page ordered by highest activity (Figure 9.9).

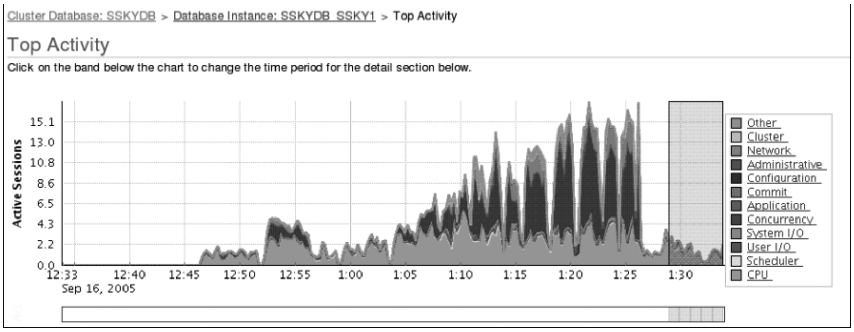


Figure 9.8
EM Top Activity

Poor query performance can occur for several reasons, such as

1. *Stale optimizer statistics.* The Oracle Cost-based Optimizer (CBO) uses the statistics collected to determine the best execution plan. Stale optimizer statistics that do not accurately represent the current status of the data in objects can easily mislead

the optimizer to generate suboptimal plans. Since there is no easy method to determine whether optimizer statistics are up-to-date or stale, this can cause poor execution plans. Starting with Oracle Database 10*g*, optimizer statistics collection has been automated. A new job, `GATHER_STATS_JOB`, runs the `DBMS_STATS.GATHER_DATABASE_STATS_JOB_PROC` procedure. The job is automatically scheduled to run every night.

2. *Missing access structures.* The absence of appropriate access structures like indexes and materialized views is a common source of poor SQL performance.
3. *Suboptimal execution plan.* The CBO can sometimes choose a suboptimal execution plan for a SQL statement. This is primarily because of incorrect estimates of some attributes of a SQL statement, such as its cost, cardinality, or predicate selectivity.
4. *Bad SQL.* Queries using Cartesian joins or `UNION ALL` clauses in SQL queries make the execution plan really expensive and retrieval of the required rows time-consuming.

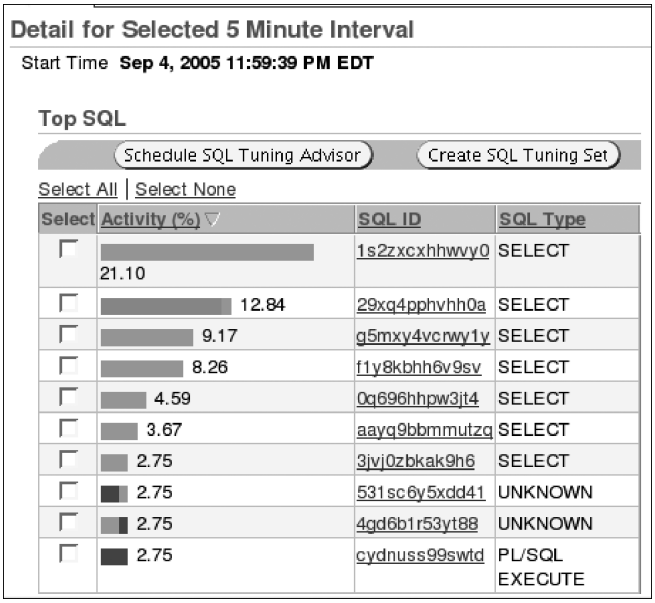


Figure 9.9 *Top SQL*

As illustrated in Figure 9.9, the `SELECT` statement with `SQL_ID` `1s2zxcxhhwvy0` has the highest activity. Once the queries to be tuned have been identified, click "Schedule SQL Tuning Advisor" and allow Oracle to

tune the queries. The tuning process will perform statistics analysis, access path analysis, SQL profiling, and structure analysis before tuning the query. Oracle provides the modified query with a comparison of the execution plan before and after tuning. The changes can then be implemented.

Another option available under the SQL Advisory section is the SQL Access Advisor. This complements the SQL Tuning Advisor functionality, focusing on the identification of indexes, materialized views, and indexes on the materialized views to improve the performance of the entire SQL workload.

9.7.4 Queries with high cluster overhead

Queries not tuned can also be an overhead to performance across the cluster, causing high delays. In Oracle Database 10*g*, four new columns have been introduced to help identify queries that are performing poorly in general and specifically in a RAC environment.

Using the `CLUSTER_WAIT_TIME` column in the `GV$SQLSTATS` view, queries that are experiencing cluster-related waits can be identified and tuned. For example, the following lists the SQL queries, giving the wait times experienced at various stages of the operation:

```
SELECT INST_ID INST,
       SQL_ID,
       APPLICATION_WAIT_TIME AWT,
       CONCURRENCY_WAIT_TIME CONWT,
       CLUSTER_WAIT_TIME CLWT,
       USER_IO_WAIT_TIME UIWT
FROM   GV$SQLSTATS
WHERE  USER_IO_WAIT_TIME > 0
ORDER BY USER_IO_WAIT_TIME;
```

```
      INST SQL_ID               AWT       CONWT       CLWT       UIWT
---------- ------------- ---------- ---------- ---------- ----------
         2 10utkgdw43zmn          0      23007   62106878     263443
         2 1h50ks4ncswfn       8874    1628544   76655404    6194443
         2 8p23kcbgfqnk4          0          0  151877550     349272
         2 gfjvxb25b773h          0       5803  179456437     932921
         2 19v5guvsgcd1v          0      44526  184945511     355544
```

Once the queries with high cluster wait time have been identified, a specific query can be retrieved using the SQL_ID.

```
SQL> SELECT SQL_FULLTXT FROM GV$SQLSTATS WHERE SQL_ID='19v5guvsgcd1v';

SQL_FULLTEXT
--------------------------------------------------------------------------------
SELECT C.TARGET_GUID, C.METRIC_GUID, C.STORE_METRIC, C.SCHEDULE, C.COLL_NAME, M.
. . .
. . .
```

SQL_FULLTEXT is of data type BLOB. To retrieve the complete information from this column, PL/SQL packages such as DBMS_LOB.READ should be used.

9.8 Sequences and index contention

Indexes, with key values generated using sequences tend to be subject to leaf block contention when the insert rate is high. This is because the index leaf block holding the highest key value is changed for every row inserted as the values are monotonically ascending. This may lead to a high transfer rate of current and CR blocks between nodes. Techniques that can reduce such situations include the following:

- Increase sequence cache size. The difference between sequence values generated by different instances increases successive index block splits and tends to create instance affinity to index leaf blocks. This also improves instance affinity to index keys deriving their values from sequences. This technique may result in significant performance gains for multi-instance INSERT-intensive applications.
- Implement the database partitioning option to physically distribute the data.
- Use locally managed tablespaces (LMTs) over dictionary-managed tablespaces.
- Use automatic segment space management (ASSM), which can provide instance affinity to table blocks and is the default in Oracle Database 10*g* Release 2.

Oracle's sequence mechanism is highly efficient and works well in a RAC environment, except when an attempt is made to "fix" it! A common mistake is to add the `ORDER` clause to the `CREATE SEQUENCE` definition. This guarantees that sequence numbers will be issued in ascending order across the cluster, which means that for every sequence number issued, Oracle has to read the last value from the data dictionary to work out the next number (which might have been issued on another instance) and update the data dictionary with the new number. This can have a disastrous impact on applications that need high sequence-generation rates.

9.9 Undo block considerations

Excessive undo block shipment and contention for undo buffers usually happens when index blocks containing active transactions from multiple instances are read frequently. When a `SELECT` statement needs to read a block with active transactions, it has to undo the changes to create a CR version. If the active transaction in the block belongs to more than one instance, the local and remote undo information needs to be combined for the CR operation. Depending on the number of index blocks changed by multiple instances and the duration of transactions, undo block shipment may become a bottleneck.

Usually this happens in applications that read recently inserted data very frequently and commit frequently. Techniques that can reduce such situations include the following:

- Shorter transactions reduce the likelihood that an index block in the cache contains uncommitted data, thereby reducing the need to access undo information for a consistent read.
- As explained earlier, increasing sequence cache sizes can reduce inter-instance concurrent access to index leaf blocks. The CR version of index blocks modified by only one instance can be fabricated without the need for remote undo information.

9.10 Load-balancing

One primary feature of a clustered environment is the distribution of the workload across the various nodes and instances in the cluster. To achieve this distribution, it is advantageous to place new connections on systems that have a higher number of resources in the cluster. As discussed in

Chapter 6, load-balancing can be configured using EM or Oracle-provided PL/SQL packages. Load-balancing is based on the number of sessions or resources available on the various instances.

Figure 9.10, the "Balance" drilldown from Spotlight on RAC, illustrates the current load across the various nodes in the cluster. While the load is balanced from the CPU and physical writes perspectives, it seems out of balance with respect to the number of LIO operations. As indicated earlier, LIO operations consume significant resources and can be reduced by tuning the SQL queries.

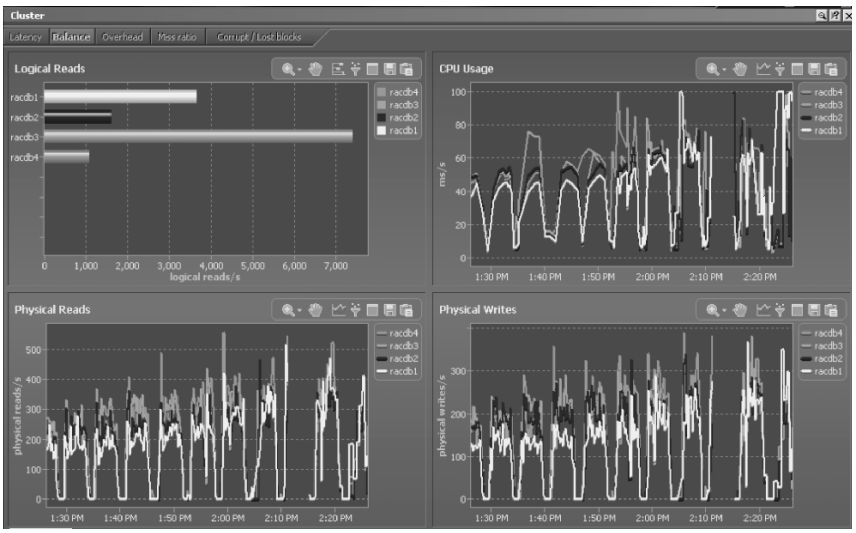


Figure 9.10
Cluster Balance

Besides tuning poorly-performing queries, the next step in obtaining a true load-balanced environment is to use the connection pooling feature on the application side and implement the RCLB feature discussed in Chapter 6. Connection pooling is supported by all application servers, such as Oracle 10*g* AS, Weblogic, and Jboss. How do you verify that the RCLB feature is working?

When `MMON` generates load advices, it is stored in the `GV$SERVICEMETRIC` view and is used to communicate with the client of the current load. For example, the following query displays the load characteristics as updated by `MMON` in the `GV$SERVICEMETRIC` view. The view is joined with `GV$INSTANCE` and `GV$ACTIVE_SERVICES` to obtain additional information.

```
set pagesize 60 space 2 numwidth 8 linesize 132 verify off feedback off
column SERVICE_NAME format a20 truncated heading 'Service'
column INSTANCE_NAME heading 'Instance'  format a10
column SERVICE_TIME heading 'Service Time|mSec/Call' format 999999999
column CPU_TIME heading 'CPU Time |mSec/Call' 99999999
column DB_TIME heading 'DB Time |mSec/Call' 99999999
column THROUGHPUT heading 'Calls/sec' format 99.99 break on SERVICE_NAME skip 1
SELECT SERVICE_NAME,
       INSTANCE_NAME,
       ELAPSEDPERCALL SERVICE_TIME,
       CPUPERCALL     CPU_TIME,
       DBTIMEPERCALL  DB_TIME,
       CALLSPERSEC    THROUGHPUT
FROM  GV$INSTANCE        GVI,
      GV$ACTIVE_SERVICES GVAS,
      GV$SERVICEMETRIC   GVSM
WHERE GVAS.INST_ID   = GVSM.INST_ID
AND   GVAS.NAME_HASH = GVSM.SERVICE_NAME_HASH
AND   GVI.INST_ID    = GVSM.INST_ID
AND   GVSM.GROUP_ID  = 10
ORDER BY
   SERVICE_NAME,
   GVI.INST_ID;
```

```
                          Service Time
Service          Instance       mSec/Call  CPU_TIME   DB_TIME  THROUGHPUT
---------------  ----------  ------------  --------  --------  ----------
SRV1             SSKY1              22981  4525.497  22980.72    202.5948
SRV1             SSKY2             124837   6111.93  124837.4    141.3127
SSKYDB           SSKY1                  0         0         0           0
SSKYDB           SSKY2               1750      1750      1750    1.158301
SYS$BACKGROUND   SSKY1                  0         0         0           0
SYS$BACKGROUND   SSKY2                  0         0         0           0
SYS$USERS        SSKY1               3608      3608      3608    .3992016
SYS$USERS        SSKY2                  0         0         0           0
```

In this output, service `SRV1` on both instances does not seem balanced. The service time on `SSKY2` is high, and the overall throughput is low; however, the DB time and CPU time values seem lower. When a message is received by the application server using the FAN technology regarding the current state of the instance, new sessions will be directed by FAN to instance `SSKY2`.

GV$SERVICEMETRIC

This view contains metric values measured for all the services defined in the database. These values are updated by the MMON process as it captures load and other service-related information from the SGA. Updates to this view happen in five-second and one-minute intervals.

Apart from making updates to the GV$SERVICEMETRIC view, MMON also updates the operating system statistics and uses this information to determine the load characteristics on the various nodes in the cluster. The following query output from GV$OSSTAT provides the operating system statistics.

```
SQL> SELECT * FROM GV$OSSTAT WHERE INST_ID=1;

   INST_ID STAT_NAME                              VALUE  OSSTAT_ID
---------- ------------------------------ ---------- ----------
         1 NUM_CPUS                                2          0
         1 IDLE_TIME                          134867          1
         1 BUSY_TIME                           82151          2
         1 USER_TIME                           66198          3
         1 SYS_TIME                            15953          4
         1 NICE_TIME                               0          6
         1 RSRC_MGR_CPU_WAIT_TIME                  0         14
         1 LOAD                           6.65917969         15
         1 PHYSICAL_MEMORY_BYTES           433270784       1008
```

GV$OSSTAT

This view contains operating system statistics updated by the MMON process and is used to determine the load on the nodes and servers. The values are in hundredths of a second, as a processor has been busy executing code and is averaged over all processors.

9.10.1 Tracing the load metric capture

RCLB can also be verified by enabling tracing at the database level using event 10735 at level 3. Tracing can be enabled using the following statement:

```
SQL> ALTER SYSTEM SET EVENTS '10735 TRACE NAME CONTEXT
FOREVER, LEVEL 3';
```

The output is generated and stored in the user dump destination directory of the instance, which can be determined after connecting to the database and verifying the parameter `USER_DUMP_DEST`.

```
SQL> SHOW PARAMETER USER_DUMP_DEST

NAME              TYPE        VALUE
----------------- ----------- ------------------------------
user_dump_dest    string      /usr/app/oracle/admin/SSKYDB/udump
```

The trace file contains the activities involving the cluster or services that are being monitored. This provides insight into how the client machines using this metric will react. The output of the trace file resembles the following:

```
/usr/app/oracle/admin/SSKYDB/udump/ssky1_ora_23316.trc
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, Real Application Clusters, OLAP and Data Mining options
ORACLE_HOME = /usr/app/oracle/product/10.2.0/db_1
. . .
Instance name: SSKY1
Redo thread mounted by this instance: 1
Oracle process number: 70
Unix process pid: 23316, image: oracleSSKY1@oradb3.sumsky.net

kswsgsnp : length of KSWS_SYSTEM_SVC is 14
kswsgsnp : length of KSWS_DATABASE_SVC is 9
           Both should be less than 64 to avoid overrun
*** SERVICE NAME:(SRV1) 2005-09-24 23:58:31.636
*** SESSION ID:(91.1) 2005-09-24 23:58:31.635
###session count for SRV1=33
###session count for SRV1=34
*** 2005-09-24 23:59:32.626
SKGXPSEGRCV: MESSAGE TRUNCATED user data 40 bytes payload 200 bytes
SKGXPSEGRCV: trucated message buffer data skgxpmsg meta data header 0x0xbfff54e0
len 40 bytes
SKGXPLOSTACK: message truncation expected
SKGXPLOSTACK: data sent to port with no buffers queued from
SSKGXPID 0xbfff559c network 0   Internet address 10.168.2.140   UDP port number
54589
```

```
SSKGXPID 0xbfff559c network 1   Internet address 220.255.187.12         UDP port
number 3260
SKGXPLOSTACK: sent seq 32763 expecting 32764
SKGXPLOSTACK: lost ack detected retransmit ack
*** 2005-09-25 00:02:38.927
###session count for SRV1=31
###session count for SRV1=30
```

This output illustrates that the MMON's statistics collection is working and configured for the default type and that the load-balance on session is counted. In this case, SRV1 is the service, and Oracle updates the MMON process, which in turn builds the advisory and posts the required advice to the advanced queue (AQ), PMON, and the ONS. The trace output also shows that there were several retransmissions, indicating potential issues with the interconnect.

9.11 Resource availability

Resources available on any machine or node or to an Oracle instance are limited, meaning they are not available in abundance and, if a process on the system needs them, they may not be immediately available. There is a physical limit to the number of resources available on any system. For example, are processor resources limited by the number of CPUs available on the system, and the amount of memory or cache area is limited by the amount of physical memory available on the system. For an Oracle process, this is further limited by the actual amount of memory allocated to the SGA. Within the SGA, the shared pool, the buffer cache, and so on, are again preallocated from the total SGA (defined by the SGA_TARGET_SIZE parameter). These are memory allocations used by a regular single-instance configuration.

In a RAC environment, there are no parameters to allocate any global-specific resources (e.g., global cache size or global shared pool area). Oracle allocates a certain portion of the available resources from the SGA for global activity. The availability of global resources can be monitored using the GV$RESOURCE_LIMIT view. For example, the following query displays the current number of resources available for global activity. In the output below, the availability of resources is limited by the column LIMIT_VALUE, and when these resources are low, the method to increase the limit is to increase the SGA_TARGET and SGA_MAX_SIZE parameters.

The following query generates the output containing the current utilization of resources:

```
SELECT
      RESOURCE_NAME,
      CURRENT_UTILIZATION CU,
      MAX_UTILIZATION MU,
      INITIAL_ALLOCATION IA,
      LIMIT_VALUE LV
FROM  GV$RESOURCE_LIMIT
WHERE MAX_UTILIZATION > 0
ORDER BY INST_ID,
         RESOURCE_NAME
```

RESOURCE_NAME	CU	MU	IA	LV
cmtcallbk	0	1	187	UNLIMITED
dml_locks	2	59	748	UNLIMITED
enqueue_locks	19	27	2261	2261
enqueue_resources	22	45	968	UNLIMITED
gcs_resources	**3447**	**3447**	**18245**	**18245**
gcs_shadows	**2259**	**2579**	**18245**	**18245**
ges_big_msgs	**27**	**28**	**964**	**UNLIMITED**
ges_cache_ress	**338**	**1240**	**0**	**UNLIMITED**
ges_procs	**35**	**36**	**320**	**320**
ges_reg_msgs	**44**	**81**	**1050**	**UNLIMITED**
max_rollback_segments	11	11	187	65535
max_shared_servers	1	1	UNLIMITED	UNLIMITED
processes	31	34	150	150
sessions	37	40	170	170
sort_segment_locks	0	1	UNLIMITED	UNLIMITED
transactions	2	4	187	UNLIMITED

When the SGA_TARGET size was increased by 10M, the global resource allocation also changed to the following new values:

gcs_resources	**2553**	**2553**	**19351**	**19351**
gcs_shadows	**1279**	**1279**	**19351**	**19351**

The rule should be that when the `MAX_UTILIZATION` (`MU`) gets close to the `LIMIT_VALUE` (`LV`) and remains constant at this value for a considerable time, consider increasing the SGA.

9.12 Response time

However large the clustered environment might be or however impressive the hardware that supports it, the primary success factor for any application environment is to provide good user response time (i.e., how quickly are the users getting a response for their operations?). Response time is the time used for the query to perform the actual service, plus any additional overhead time spent by the process waiting for resources. In other words, **response time = service time + wait time**.

Service time is the time used to make database-related calls and is improved by tuning the queries and optimizing the various Oracle parameters. The trick is to determine which part of the application is consuming high service times. A number of commercial tools exist to help measure elapsed time in either benchmarking or production environments.

In the absence of such a tool, the alternative is to try to instrument the application itself. This can be done by inserting timing calls into the application using language-specific syntax or using timing routines. Such embedded routines will log the activity into files, which can then be analyzed to determine poorly performing areas of the application code.

Wait time is the overhead experienced by the processes while performing an operation. Wait time can be due to several reasons. The Oracle Wait Interface (OWI) provides a great instrumentation on the various wait states encountered by database operations. In this section, the common waits encountered in a RAC environment are discussed

9.13 Oracle Wait Interface

OWI, or wait event, in Oracle is a situation when a session puts itself to sleep to wait for some external event to happen. Wait events are classified as system, service, or idle conditions. Idle waits occur when a session is waiting for a work request to be issued. Service wait events are waits for the system to perform asynchronous operations such as I/O or process creation.

Before Oracle Database 10*g*, OWI was driven by the three primary views: `GV$SYSTEM_EVENT`, `GV$SESSION_EVENT`, and `GV$SESSION_WAIT`. These views provided wait times at the system level or at the individual session level.

However, there was no method to isolate waits to a specific set of modules within an application. In Oracle Database 10*g*, with the introduction of a new abstraction class to support SOA, a new view is introduced, GV$SERVICE_EVENT, which provides the intermediate level of instrumentation for performance diagnostics and tuning.

To further streamline the wait events, in Oracle Database 10*g*, wait events have been classified into wait classes. This helps in grouping wait events and directing tuning efforts to the various areas of the environment. The following query illustrates the number of wait events grouped under the various wait classes:

```
SELECT WAIT_CLASS,
       COUNT(*)
FROM  V$SYSTEM_EVENT
GROUP BY WAIT_CLASS;

WAIT_CLASS             COUNT(*)
-------------------- ----------
Concurrency                 8
System I/O                  9
User I/O                    7
Configuration               2
Other                      57
Cluster                    11
Application                 3
Idle                       24
Commit                      1
Network                     3
```

While wait events pertaining to all classes have some direct or indirect impact on a RAC environment, for our discussions, only wait events[5] that belong to the Cluster class will be discussed. In Oracle Database 10*g* Release 2, there are 47 wait events in this class.

Note: In a clustered environment, all the V$ views, when prefixed with a "G" (meaning global), will provide statistics from all instances in the cluster.

5. For a thorough understanding of wait events, please refer to Oracle Wait Interface: Practical Guide to Performance Diagnostics & Tuning (2004) by Richmond Shee, Kirtikumar Deshpande, and K. Gopalakrishnan.

Wait categorization has changed and taken a strategic direction in Oracle Database 10*g*. Table 9.1 provides a mapping between the Oracle 9*i* wait events and the Oracle 10*g* wait events concerning RAC.

In Table 9.1, the Oracle Database 10*g* wait events are grouped as current or CR-type wait events. Current and CR waits are based on current and CR blocks in the buffer of various instances. What is the difference between CR and current waits?

9.13.1 Consistent read versus current

The first time a block is read into a buffer of any participating instance, it is termed a current block, no matter what the purpose of the user accessing

Table 9.1 *Oracle Database 9i Event to Oracle Database 10g Event Mapping*

Oracle 9i Wait Event	Oracle 10*g* Wait Event
global cache null to x global cache null to s	gc current block 2-way gc current block 3-way gc current block busy gc current block congested
global cache open x	gc current block 2-way gc current block 3-way gc current block busy gc current block congested gc current grant 2-way gc current multiblock request
global cache s to x	gc current grant 2-way
global cache cr request	gc cr block 2-way gc cr block 3-way gc cr block busy gc cr block congested gc cr grant 2-way gc cr multi block request

the block may be, meaning it can be a `SELECT` operation or a DML operation. The first access is always termed a current operation. Subsequently,

when the block is transferred from one instance to another instance because a session on that instance requested the block, it is termed as a CR block.

As discussed in Chapter 2, when a block is required by a process for read purposes, it accesses the block in shared mode. When the block is to be modified, the processes requires a grant from the GCS to access this block in exclusive mode. The frequency of state/grant changes to the block can be obtained by querying the `STATE` column from the `GV$BH` view. The following are RAC-related state changes [9]:

- `XCUR` exclusive current
- `SCUR` shared current
- `CR` consistent read
- `READ` reading from disk
- `WRITE` write clone mode
- `PI` past image

When blocks are required by more than one process on the same instance, Oracle will clone the block. The number of times a block can be cloned is defined by the parameter `_DB_BLOCK_MAX_CR_DBA`[6] and defaults to six, meaning only six cloned copies of the same block of the same data block address (DBA) can exist in the local buffer of an instance (`SSKY1` in Figure 9.10) at any given time. In Oracle Database 10*g* Release 1, Oracle placed the CR blocks in the cold end of the buffer cache. Now the CR blocks are treated like any other data block, and the Touch Count Algorithm (TCA) is used. Under TCA the block read is placed at midpoint (insertion point) in the buffer cache and will have to gain creditability when session access or touch the block, to climb up the stack to reach the hot buffer area. If the block is not touched by other sessions, it will move down the stack and finally get flushed out when the buffer is needed by new blocks.

Similarly, when blocks are required by more than one other instance, Oracle will ship an image of the CR block (if it has not already done so) to the requesting instance. As discussed in Chapter 2, blocks are shipped from one instance to another (Figure 9.11), and the details about which instance contains the block are maintained in the GRD. The number of

6. Underscore parameters should be modified only after consulting with Oracle support.

instances a block can exist on is determined by the parameter `_FAIRNESS_THRESHOLD`[7] and defaults to four, meaning only four images of the same block of a particular DBA can exist in a RAC cluster (irrespective of the number of instances) at any given time.

Once the holder reaches the threshold defined by the parameter `_FAIRNESS_THRESHOLD`, it stops making more copies, flushes the redo to the disk, and downgrades the locks [9].

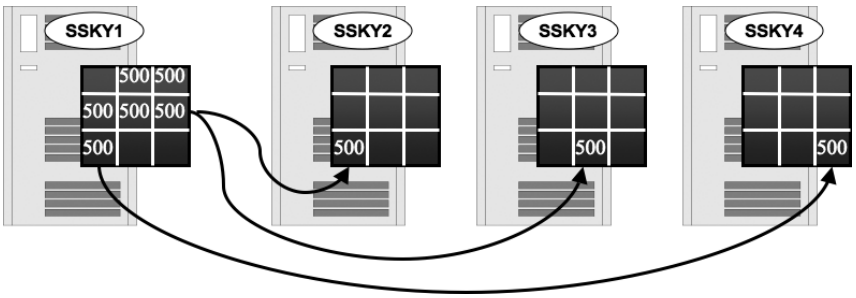


Figure 9.11
Block Cloning and Imaging

Note: While data movement in a RAC environment is at the block level, data modifications are all at the row level (as in a single-instance configuration).

9.13.2 gc cr/current block 2-way/3-way

One primary function of a cluster environment is to share blocks between instances, minimizing access to the physical storage. It is common that blocks will be transferred from one instance to another. The wait events listed in Table 9.1 all relate to the block transfers between the various instances in the cluster. For example, a 2-way event indicates that there was a 2-way shipment to transfer the block in which the requester sends a message to the master, and the master ships the block back to the requester. Similarly, a 3-way event indicates that there were three hops before the block was received by the requestor. A three-hop scenario is illustrated in Figure 9.12.

7. Underscore parameters should be modified only after consulting with Oracle support.

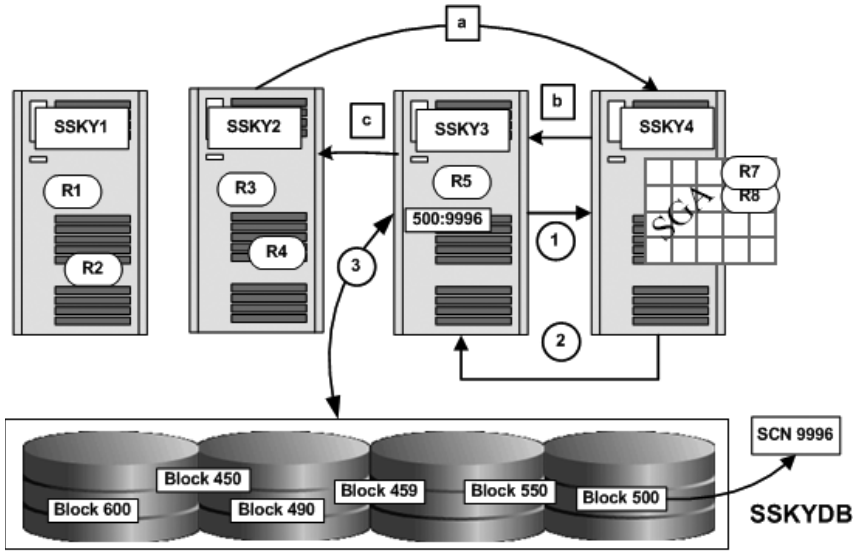


Figure 9.12 *3-Way Block Transfer*

As illustrated in Figure 9.12, in a RAC cluster there are two possibilities when the GCS process has to perform three hops before the requestor receives the block.

1. Read the block from the disk.
   - Instance `SSKY1` requests block 500 from the GCS located on instance `SSKY4`.
   - Instance `SSKY4`, after checking against the GRD, determines that neither instance `SSKY4` nor any other instance in the cluster has a copy of the block requested. Hence, it sends a message to the requesting instance to read the block from disk.
   - Instance `SSKY3` reads the block from disk.
2. Request another instance to transfer the block.
   - Instance `SSKY2` requests block 500 from the GCS located on instance `SSKY4`.
   - Instance `SSKY4` verifies against its GRD and determines that the block is currently held by instance `SSKY3`. It sends a message to instance `SSKY3` requesting that it send a copy of the block to instance `SSKY2`.
   - Instance `SSKY3` accepts the request and sends the block to instance `SSKY2`.

In the RAC architecture, in order to get a block, the GCS may have to perform two hops if the block requested cannot be found on the local instance because, if the master is located on the instance where the demand for the blocks concerning a specific object is the highest, hence, most likely, the block is going to be present where the master is located. In Oracle Database 10*g* Release 2, when the demand for the blocks concerning the object increases on another instance, the master will dynamically move or relocate to this new instance. Only when blocks are not found on the master does the GCS need to direct the transfer from another instance or from disk (three hops). Irrespective of the number of instances in the cluster, the GCS process will have to make a maximum of three hops. That is why the RAC architecture scales irrespective of the number of instances in the cluster: no matter how many instances might be associated with the cluster, the number of hops will never exceed three. The 3-way wait values that are significantly high could indicate that the block was never found on the master or the master did not relocate to another instance where the demand is high.

9.13.3 gc cr/current block congested

This wait indicates that the request was made to the Distributed Lock Manager (DLM) but ended up waiting, and the foreground process have to retry the request. Under these circumstances, the gc cr/current block congested wait counter is incremented.

Normally, this indicates that the GCS process (`LMSn` background) is not able to keep up with the requests. `LMSn` is a single-threaded synchronous process and follows the first in first out (FIFO) algorithm to complete block requests. This means that when multiple requests are received, the GES will place the requests in a queue and send the request to the `LMSn` process when it has completed a previous operation. In such situations, consideration should be given to increase the number of `LMSn` processes using the parameter `GCS_SERVER_PROCESSES`. A good rule of thumb is to set the value of this parameter to one `LMSn` processes for every two CPUs on the node. Setting it to high values will consume resources and can affect the overall performance of the cluster. `LMSn` processing delays can also be the result of scheduling delays and high queue lengths experienced by the node at the operating system level.

RAC has several other types of wait events. For a better understanding, the global cache–related wait events have been grouped in Figure 9.13 by the areas they affect.

9.13.4 gc remaster

This wait indicates the delay encountered when remastering the resource master from one instance to another instance in the cluster. In Oracle Database 10*g* Release 2, RAC architecture allows for dynamic remastering from a less busy instance to an instance where the demand for the object is the highest. This movement is called *resource affinity*.

Apart from remastering based on demand, remastering also happens when an instance leaves (fails) or joins the clustered configuration. During instance failure, remastering may happen more than once, first to place the master dynamically on one of the surviving nodes and, second, once the cluster has reached a stable state to reassess the situation and remaster again based on demand. Monitoring the remastering activity is discussed later in the chapter.

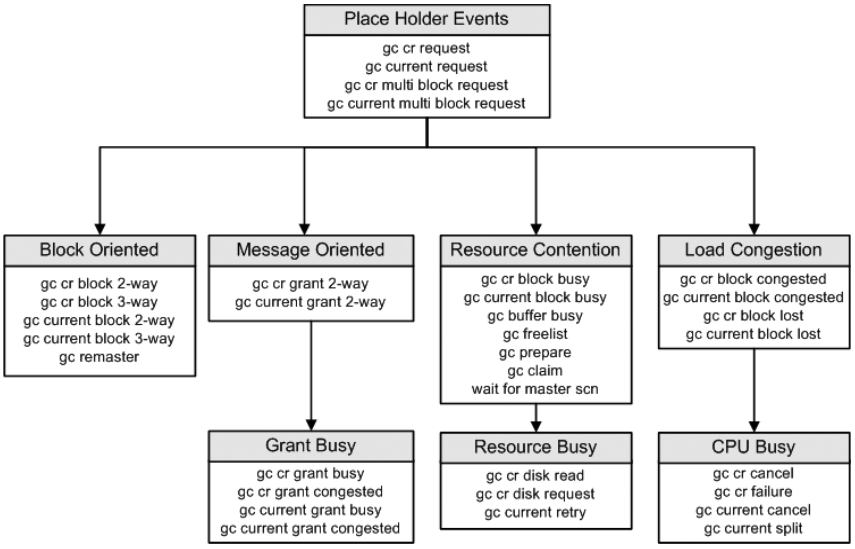


Figure 9.13 *Wait Events Grouped*

9.13.5 wait for master SCN

Each instance in the cluster will generate its own SCN and, subsequently, using the propagation method, will resynchronize to the highest SCN in the cluster. This wait indicates the number of times the foreground processes waited for SCNs to be acknowledged from other instances in the cluster.

Before Oracle Database 10*g* Release 2, the method of SCN propagation was driven by the parameter `MAX_COMMIT_PROPAGATION_DELAY`. Setting this

value to higher than zero uses the Lamport algorithm. In Oracle Database 10*g* Release 2, this parameter is deprecated and is maintained for backward compatibility only and defaults to zero. This functionality is now driven by the underscore (hidden) parameter `_IMMEDIATE_COMMIT_PROPAGATION`[8] and has a Boolean value of `TRUE` or `FALSE`.

When this parameter is set to `TRUE` (default), Oracle uses the Block on Commit (BOC) algorithm for messaging. While the method of propagation remains similar to the Lamport algorithm, in the case of BOC, the global high water mark for the SCNs sent and received is maintained, reducing messaging traffic for global SCN synchronization and, in turn, improving overall performance.

9.13.6 gc cr/current request

This event indicates the time spent waiting when a session is looking for a CR version of a block (indicated by the lock element number or class of the buffer in `P3` (Table 9.2), and the block number in column `P2`; it belongs to the file indicated in column `P1` in the `V$SESSION_WAIT` view cannot find it in its local cache, and so has made a request to a remote instance for the block. However, the transferred block has not yet arrived at the requesting instance. The event ends when the requesting session gets the block or permission to read the block from disk.

This event may not always indicate a problem with the GCS requests. Circumstances under which this wait time value is high can be due to the following:

1. Data blocks are being modified frequently on all instances.
2. Requests for block have resulted in a cache miss.
3. `LMSn` cannot keep up with the high number of CR requests.
4. There are latency issues with the interconnect.
5. Several full table scans are being performed.

High waits on this event can be reduced by looking at the system and scheduling delays at the operating system level (e.g., ensuring that the `LMSn` process gets enough CPU cycles to complete its operation). As explained earlier, increasing the number of `LMSn` processes may be

8. Underscore parameters should be modified only after consulting with Oracle support.

required. Ensuring that a high-speed interconnect is used will also help reduce waits in this category.

Table 9.2 *Common Block Classes [9]*

Block Class	Description
0	System rollback segment
1	Data blocks
2	Sort blocks
3	Deferred rollback segment blocks (save undo)
4	Segment header blocks
5	Deferred rollback segment header blocks
6	Freelist blocks
7	Extent map blocks
8	Bitmapped space management blocks
9	Space management index blocks
10	Unused

9.13.7 gc current/CR block busy

This wait indicates that a current or CR block was requested and received but was not sent immediately by `LMSn` because some special condition that delayed the sending was found.

9.13.8 gc current grant busy

This wait indicates that a current block was requested, and a grant message was received. The busy hint implies that the request was blocked because others were ahead of it, or it could not be handled immediately.

9.14 Server/database statistics

Oracle gathers performance-related statistics from various categories that are subsequently analyzed. These statistics help in diagnosing performance-related issues in the database. Several statistics are collected for the RAC

environment. Monitoring and tuning the related areas will help improve performance.

Server statistics are stored at the system level or at the session level. As with the OWI, system-level statistics stored in `GV$SYSSTAT` are a cumulative value of all sessions since the instance was started. On the other hand, `GV$SESSTAT` contains values for individual sessions that are active at a given point. With the introduction of services in Oracle Database 10*g*, a new view at the service level is also available: `GV$SERVICE_STATS`.

The RAC-related statistics maintained at the system level can be obtained using the following query:

```
SELECT NAME,
       VALUE
FROM V$SYSSTAT
WHERE NAME LIKE 'g%';

NAME                                              VALUE
---------------------------------------- ----------
global enqueue gets sync                      104853
global enqueue gets async                      15529
global enqueue get time                         3203
global enqueue releases                       111512
gcs messages sent                               8286
ges messages sent                               7656
global enqueue CPU used by this session          816
gc cr blocks served                             3461
gc cr block build time                            30
gc cr block flush time                            18
gc cr block send time                             54
gc current blocks served                        4458
gc current block pin time                         13
gc current block flush time                        0
gc current block send time                       112
gc cr blocks received                             76
gc cr block receive time                          67
gc current blocks received                       280
gc current block receive time                    126
gc blocks lost                                    89
gc claim blocks lost                               0
gc blocks corrupt                                  0
```

```
gc CPU used by this session                    807
global undo segment hints helped                 0
global undo segment hints were stale             0
```

9.14.1 Time model statistics

In Oracle Database 10*g*, a new method of monitoring statistics is introduced called the time model. While the wait event shows where the system is spending time waiting, the time model shows where the system is wasting time doing activities such as reloading the SQL cache or performing a parse operation.

Under the time model method, the time spent at various stages of the process is collected, providing a more detailed level of data for problem diagnosis. Like the database statistics discussed earlier, the time model statistics can also be obtained at the system level from the GV$SYS_TIME_MODEL or at the session level from the GV$SES_TIME_MODEL views. The following query lists the time model statistics at the system level:

```
COL STAT_NAME FORMAT A45
SELECT STAT_NAME,
       VALUE
FROM   V$SYS_TIME_MODEL;

STAT_NAME                                           VALUE
--------------------------------------------- ----------
DB time                                        139911756
DB CPU                                          30228434
background elapsed time                        171121413
background cpu time                             38843962
sequence load elapsed time                        527830
parse time elapsed                              47249379
hard parse elapsed time                         45738124
sql execute elapsed time                       137995412
connection management call elapsed time           402551
failed parse elapsed time                              0
failed parse (out of shared memory) elapsed time       0
hard parse (sharing criteria) elapsed time         14348
hard parse (bind mismatch) elapsed time            12671
PL/SQL execution elapsed time                    6104754
inbound PL/SQL rpc elapsed time                        0
PL/SQL compilation elapsed time                  5856205
```

```
Java execution elapsed time                              0
repeated bind elapsed time                          163961
RMAN cpu time (backup/restore)                           0
```

Of all the statistics available under the time model, the two statistics that are important for diagnosing performance-related issues are DB time (database time) and DB CPU. DB time is the key statistic for the time model and represents the total elapsed time spent servicing user requests on database-related calls. The value includes CPU time spent, plus any non-idle wait time to complete the database operation and waiting on the run queue for CPU. In other words,

```
DB time = Sum of time spent processing all user requests = Sum
of time (Running on CPU + Waiting for Resources + Waiting for
CPU) [10]
```

```
DB time/sec = Total DB time / Elapsed time or Wall clock time
```

DB time is used as a mechanism to measure other time statistic values. Typically, the ratio of the time statistic collected from a wait event to DB time will help determine the total system impact of that wait event. The goal of the DBA is to reduce the DB time to improve the overall performance of the database.

9.15 Service-level metrics

With the support for SOA and Oracle's implementation of a layer of abstraction to the application as services, Oracle has also introduced in Oracle Database 10*g* a new dimension for performance tuning. With services, workloads are visible and measurable, and statistics gathered can be attributed to specific applications or modules within applications. Services provide another level of performance optimization by connecting a specific poorly performing SQL operation to a service instead of the traditional approach where a SQL was always related to a session. Apart from wait events at the service level (discussed earlier) available through the `GV$SERVICE_EVENTS` view, the following statistics are also collected at the service level:

User calls	Workarea executions – optimal
DB time – response time	Workarea executions – onepass
DB CPU – CPU/service	Workarea executions – multipass
Parse count (total)	Session cursor cache hits
Parse time elapsed	User rollbacks
Parse time CPU	DB block changes
Execute count	gc cr blocks received
SQL execute elapsed time	gc cr block receive time
Opened cursors cumulative	gc current blocks received
Session logical reads	gc current block receive time
Physical reads	Cluster wait time
Physical writes	Concurrency wait time
Redo size	Application wait time
User commits	User I/O wait time

Optionally, Oracle provides additional levels of data collection by defining modules within applications or actions within modules. This helps in easy identification of performance areas within the application. Module and action-level monitoring can be enabled using the following PL/SQL definition:

```
DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE (<SERVICE_NAME>,
<MODULE NAME>)
```

For example, to enable statistics collection for module `ORDERS` in service `SRV1`, the following should be executed on the database server on any of the available instances:

```
EXEC DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE ('SR1','ORDERS');
```

Once monitoring has been enabled, it remains active until it is disabled using the following procedure:

```
EXEC DBMS_MONITOR.SERV_MOD_ACT_STAT_DISABLE (null,null);
```

These definitions can be verified by querying the `DBA_ENABLED_AGGREGATIONS` table:

```
SELECT AGGREGATION_TYPE,
       QUALIFIER_ID1 MODULE,
       QUALIFIER_ID2 ACTION
FROM DBA_ENABLED_AGGREGATIONS;

AGGREGATION_TYPE      MODULE     ACTION
--------------------- ---------- ---------
SERVICE_MODULE_ACTION ORDERS     Mixed
SERVICE_MODULE_ACTION ORDERS     Multiple
SERVICE_MODULE_ACTION ORDERS     Read
SERVICE_MODULE_ACTION ORDERS     Update
```

Before monitoring the performance statistics, the application connecting to the database should connect to the `SERVICE_NAME` being monitored, and the application should have the module identified in the code. The module name can be set in the application using the following procedure:

```
DBMS_APPLICATION_INFO.SET_MODULE (<MODULE NAME>, <ACTION TYPE>);
```

For example, to let the database know which module is being monitored, the following procedure should be executed from inside the application module:

```
EXEC DBMS_APPLICATION_INFO.SET_MODULE ('ORDERS');
```

Apart from monitoring individual modules, performance-related statistics can also be collected for any specific action. For example, the performance of various users executing update statements can also be monitored with the following procedure:

```
EXEC DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE
('SRV1','ORDERS','UPDATE');
```

Similarly, inside the application, the ORDERS module, the specific action (UPDATE) being modified should also be identified using the following procedure:

```
EXEC DBMS_APPLICATION_INFO.SET_MODULE ('ORDERS','UPDATE');
```

This feature of collecting a performance matrix for an action type within a module was not available until Oracle Database 10*g* and is a great feature that can easily be taken advantage of. In a RAC environment with DWM implementation, this helps track a module's performance across the cluster.

Once the statistics collection has been enabled on the database server and on the client side, the performance metrics can be collected or monitored. For example, the output from the following script against the GV$SERVICE_STATS view provides a high-level indication that DB time for SRV1 on instance 2 is significantly high.

```
COL STAT_NAME FORMAT A35
COL MODULE FORMAT A10
COL SERVICE FORMAT A15
COL INST FORMAT 999
SELECT INST_ID INST,
       SERVICE_NAME SERVICE,
       STAT_NAME,
       VALUE
FROM   GV$SERVICE_STATS
WHERE  VALUE > 0
AND    SERVICE_NAME ='SRV1'
ORDER  BY VALUE;
```

```
 INST SERVICE         STAT_NAME                                VALUE
---- --------------- ---------------------------------- ----------
   2 SRV1            parse count (total)                      114332
   2 SRV1            opened cursors cumulative                114574
   2 SRV1            execute count                            252873
   2 SRV1            session logical reads                   5254843
   2 SRV1            redo size                              21199172
   2 SRV1            cluster wait time                      27815562
   2 SRV1            application wait time                  87809921
   2 SRV1            user I/O wait time                     98546228
```

```
2 SRV1          concurrency wait time          2055384221
2 SRV1          DB CPU                         2156249531
2 SRV1          sql execute elapsed time       6912286900
2 SRV1          parse time elapsed             8681424580
2 SRV1          DB time                        9845032706
```

To identify the module and action type that caused the high DB time values, use the following script against the view GV$SERV_MOD_ACT_STATS:

```
COL STAT_NAME FORMAT A35
COL MODULE FORMAT A10
COL SERVICE FORMAT A10
COL INST FORMAT 999
COL ACTION FORMAT A8
SELECT INST_ID INST,
       AGGREGATION_TYPE,
       SERVICE_NAME SERVICE,
       MODULE,
       ACTION,
       STAT_NAME,
       VALUE
FROM  GV$SERV_MOD_ACT_STATS;
```

The benefits provided for monitoring activity at the service level do not stop here. Tracing user operations is also available at the module and action level. Oracle generates one trace file per session, connecting to the database using the SERVICE_NAME. Users connecting to the database may get attached to any of the available instances supporting the service. The advantage of tracing at this level is that, when multiple trace files are generated from the current instance or across instances in the cluster, data related to a specific action type can be grouped together. For example, the following procedure will enable tracing of a service at the module and action levels:

```
DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE
(<SERVICE_NAME>,<MODULE NAME>,<ACTION TYPE>);

EXEC DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE
('SRV1','ORDERS','MIXED');
```

Apart from the basic SQL-level trace information, additional information such as wait events encountered (collected by default), bind variables, and values used can also be collected. For example:

```
EXEC DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE (
  SERVICE_NAME => 'SRV1',
  MODULE_NAME  => 'ORDERS',
  ACTION_NAME  => DBMS_MONITOR.ALL_ACTIONS,
  WAITS        => TRUE,
  BINDS        => TRUE);
```

Note: The `SERV_MOD_ACT_TRACE_ENABLE` utility generates trace files similar to the trace files generated using event 10406 at level 1. Enabling wait events and binds will be similar to generating tracing using 10406 at level 12.

Once these procedures are executed on the database server, the trace files are generated in the `USER_DUMP_DEST` directory on the respective instances. Oracle generates one trace file for every session, connecting to the database using the service `SRV1`. The trace files can then be consolidated based on different criteria. Based on the example, the trace file will contain information such as SQL statements, wait events encountered, bind variables, and bind values. This trace information can be

1. Analyzed directly using the Transient Kernel Profiler (`tkprof`) utility.

```
tkprof ssky1.ora.*.trc trcSRV1.prf explain=bmf/bmf
table=bmf.temp sys=no
```

2. Scanned through and extracted by action type using the `trcsess` utility. Once extracted into a single file, it can be analyzed using the `tkprof` utility.

```
trcsess output=trcMixed.trc service=SRV1
module='ORDERS' action=Mixed  ssky1_ora_*.trc

trcsess [output=<output file name >]
[session=<session ID>] [clientid=<clientid>]
[service=<service name>] [action=<action name>]
```

```
[module=<module name>] <trace file names>
```

- `<output file name>` is the output destination, default being standard output.
- `<sessionID>` is the session to be traced. `SessionID` is a combination of session index and session serial number.
- `<clientid>` is the client to be traced.
- `<service name>` is the service to be traced.
- `<action name>` is the action to be traced.
- `<module name>` is the module to be traced.
- `<trace file names>` is a space-separated list of trace files with wild card '*' supported.

The following `trcsess` command will extract trace information from the trace file that pertains to service `SRV1` but contains all modules and actions:

```
trcsess output=trcSRV1.trc service=SRV1 ssky1_ora_*.trc
```

Similarly, the following `trcsess` command will extract trace information from the trace files that pertain to service `SRV1` and module `ORDERS`, but will contain all actions:

```
trcsess output=trcRead.trc service=SRV1 action=Mixed
module=ORDERS ssky1_ora_*.trc
```

9.16 Identifying blockers across instances

When two users request access to the same row of data for update purposes, all users except the first will remain in wait mode until the first user has committed or rolled back the change. This is true irrespective of whether it is a single-instance configuration or a clustered RAC configuration, with an additional possibility. Apart from users on the same instance, users from other instances can also request the same row at the same time. The following query helps identify a blocking session in a RAC environment:

```
SELECT DECODE(G.INST_ID,1,'SSKY1',2,'SSKY2') INSTANCE,
       S.SID,
       G.TYPE,
       S.USERNAME,
       S.SERIAL#,
       S.PROCESS,
       DECODE(LMODE,0,'None',1,'Null',2,'Row-S',3,'Row-X',4,'Share',5,'S/ROW',
6,'Exclusive') LMODE,
       DECODE(REQUEST,0,'None',1,'Null',2,'Row-S',3,'Row-X',4,'Share', 5,'S/
ROW',6,'Exclusive')REQUEST,
       DECODE(REQUEST,0,'BLOCKER','WAITER') STATE
FROM   GV$GLOBAL_BLOCKED_LOCKS G,
       GV$SESSION S
WHERE  G.SID = S.SID
AND    G.INST_ID = S.INST_ID
ORDER BY STATE
```

INSTANCE	**SID**	TY	USERN	SERIAL#	PROCESS	LMODE	REQUEST	STATE
SSKY2	**132**	TX	OE	519	2576:820	**Exclusive**	None	**BLOCKER**
SSKY2	**148**	TX	OE	78	980:3388	None	Exclusive	**WAITER**
SSKY1	**139**	TX	OE	114	3192:2972	None	Exclusive	**WAITER**

In this output, the session with SID 132 on instance SSKY2 is the BLOCKER because this session accessed this row first in an exclusive mode and has either committed or rolled back the transaction. Subsequently, the session with SID 148 also on instance SSKY2 requested this same row for update, followed by SID 139 from instance SSKY1. Both of these SIDs will remain as WAITER until the row is available to the session to complete its update operation. Blockers can also be determined from the EM by selecting the "Blocking Sessions" option from the database performance page or by querying BLOCKING_SESSION_STATUS, BLOCKING_INSTANCE, BLOCKING_SESSION columns from V$SESSION view.

9.17 Identifying hot blocks

One primary feature of RAC is to provide scalability by allowing transfer of blocks on user request from one instance to the other, thus avoiding any disk I/O. When the same sets of blocks is requested back and forth between the various instances in the cluster, the sessions requesting the block may have to wait until the holding instance has released it. This back-and-forth

movement of the same blocks may indicate or create contention over the interconnect. Identifying such blocks and then analyzing ways in which they can be minimized will help in the overall performance of the cluster. Hot blocks can be identified by querying the V$SEGSTAT view.

9.18 Monitoring remastering

One of the primary scalability factors for Oracle is balancing resource demands on each instance by satisfying requests on local nodes where demand for the resource is high. For example, if there are 1,000 users on instance SSKY1 and only 100 users on instance SSKY2 for the EMP table object, it would be more efficient to place the master for the EMP object on instance SSKY1. However, when demand for the object changes, and SSKY2 has more users for the object, then the master should move from instance SSKY1 to SSKY2. This is called *resource remastering*. The following query against the V$DYNAMIC_REMASTER_STATS view gives the current remaster activity for an instance:

```
SELECT REMASTER_OPS,
       REMASTER_TIME,
       REMASTERED_OBJECTS,
       CURRENT_OBJECTS,
       SYNC_TIME
FROM   V$DYNAMIC_REMASTER_STATS;
REMASTER_OPS REMASTER_TIME REMASTERED_OBJECTS CURRENT_OBJECTS  SYNC_TIME
------------ ------------- ------------------ --------------- ----------
           2           608                 10              10        360
```

In this output, REMASTER_OPS indicates the number of remaster operations completed this far, REMASTER_TIME indicates the time spent on remaster activities, REMASTERED_OBJECTS indicates the number of objects remastered, the CURRENT_OBJECTS column indicates the number of objects mastered on the current instance that have not been remastered, and SYNC_TIME indicates the amount of time spent in cache synchronization activities during the remaster operation.

Note: All values in this view are cumulative, meaning they reflect the remastering activity since the instance started.

On a clustered configuration with more than two nodes, if the 3-way wait event indicate a significantly high number (numbers greater than 2-way wait events), this would be an indication that remastering was not working or that remaster activity had been disabled. While remastering is based on the number of times the object is touched in a particular instance, the requirement is that it be touched 60 times more than the other instance in a period of approximately 10 minutes. The touch count logic for remastering and the maximum period before remastering occurs are tunable using the underscore parameters `_GC_AFFINITY_LIMIT` and `_GC_AFFINITY_TIME`.[9]

In Oracle Database 10*g* Release 2, this feature is enabled by default at the object level. In prior releases, though mastering was maintained at the datafile level, no dynamic remastering occurred.

9.19 Operating system tuning

Once the database has been tuned to high performance standards, it is also a good idea to look at the overall performance from the operating system level. Areas of the operating system that influence the performance are CPU utilization and memory utilization.

9.19.1 CPU utilization

CPU utilization is the amount of time that the active CPUs on the system are running processes. CPU utilization statistics presented by the `sar -u` command are displayed as a composite of the %`system`, %`user`, and %`idle` times, where the addition of all three parameters will equate to 100%. A lower %`idle` time indicates a higher workload.

```
[root@oradb4 oracle]# sar -u 5 6
Linux 2.4.21-e.41smp (oradb4.sumsky.net)        10/03/2005

11:06:57 PM       CPU     %user     %nice   %system     %idle
11:07:02 PM       all     89.07      0.00      9.72      1.21
11:07:07 PM       all     55.74      0.00     11.06     33.19
11:07:12 PM       all     73.35      0.00      7.23     19.42
11:07:17 PM       all     90.82      0.00      8.16      1.02
11:07:22 PM       all     90.43      0.00      9.16      0.41
11:07:27 PM       all     91.80      0.00      8.20      0.00
Average:          all     70.06      0.00      8.92     21.02
```

9. Underscore parameters should be modified only after consulting with Oracle support.

System and user statistics represent the proportion of time the CPUs are working on system-related activities or user-based programs, respectively. Characterization of a system's performance in terms of CPU utilization is a widely used approach. In a normal workload, %`system` should not consume more than 20% of CPU. CPU utilization information is made more significant when combined with the run queue length and run queue occupancy statistics. For example, if the server is running close to the 95% utilization level for most of the day, does this indicate an immediate CPU deficiency on the server? The answer is, it depends. If this is a single-CPU system, and the run queue is consistently at a value of 1, then the answer is, it is difficult to arrive at any definite conclusion. If the run queue length consistently exceeds the number of CPUs on the system, and the run queue occupancy is consistently high, together with a high CPU utilization rate, then this would indicate a CPU deficiency. CPU information can be verified using `cat /proc/cpuinfo` on Linux-based systems and from the Task Manager in Windows-based environments.

An ideal situation for the server is to run consistently under the 80% utilization level with a stable workload. In this situation, the investment in CPU power is justified since the utilization rate is high, yet not at its maximum. The reality of most servers is that workload fluctuates throughout an entire day and rarely shows as stable. The measurement of CPU utilization can help in identifying shortfalls in CPU processing power, especially during peak periods where the demand on CPU resources can exceed availability and cause systemwide performance degradation. The `sar -u` command will generate the CPU utilization statistics.

9.19.2 Memory utilization

Like the processing power provided by the CPUs on a system, the amount of data stored in Oracle's buffer is based on the memory available at the operating system level, meaning a portion of the total memory available is allocated to the Oracle instance on the server in its SGA.

meminfo

Memory information can be obtained using `cat /proc/meminfo` on a Linux-based system and using the Task Manager in a Windows environment.

```
[root@oradb4 oracle]# cat /proc/meminfo
        total:    used:    free:  shared: buffers:  cached:
Mem:  2636144640 2518822912 117321728 699658240 182558720 1304363008
```

```
Swap: 2146787328        0 2146787328
MemTotal:      2574360 kB
MemFree:        114572 kB
MemShared:      683260 kB
Buffers:        178280 kB
Cached:        1273792 kB
SwapCached:          0 kB
Active:        1164964 kB
Inact_dirty:    970368 kB
Inact_clean:         0 kB
Inact_target:   643492 kB
HighTotal:     1703860 kB
HighFree:         2036 kB
LowTotal:       870500 kB
LowFree:        112536 kB
SwapTotal:     2096472 kB
SwapFree:      2096472 kB
BigPagesFree:        0 kB
Committed_AS:  2153424 kB
HugePages_Total:     0
HugePages_Free:      0
Hugepagesize:     4096 kB
[root@oradb4 oracle]#
```

slabInfo

Another statistic collected by the operating system is `slabinfo`, which provides information about internal kernel caches. Slabs are small allocations of memory, less than a page or not a multiple of page size. The statistics can be useful in determining where the kernel is using too much memory.

```
[root@oradb4 oracle]# cat /proc/slabinfo
slabinfo - version: 1.1 (SMP)
kmem_cache            96     96    244    6    6    1 :  252  126
asm_request          116    118     64    2    2    1 :  252  126
nfs_read_data          0      0    384    0    0    1 :  124   62
nfs_inode_cache        2     17    224    1    1    1 :  252  126
nfs_write_data         0      0    384    0    0    1 :  124   62
nfs_page               0      0     96    0    0    1 :  252  126
ocfs_fileentry        64     64    512    8    8    1 :  124   62
ocfs_lockres         384    384    160   16   16    1 :  252  126
ocfs_ofile           382    444    320   37   37    1 :  124   62
```

```
ocfs_oin              214    340    192   15   17    1 :  252  126

                    active-objects

                    |      allocated-objects

                    |      |         object-size

                    |      |         |   active-slab-allocations

                    |      |         |   |     total-slab-allocations

                    |      |         |   |     |        alloc-size

                    |      |         |   |     |        |

asm_request       116    118       64   2     2        1 :  252  126

                                                             |     |

                                                           limit     |

                                                           batch-count
```

- *active-objects*. After creating a slab cache, you allocate your objects out of that slab cache. This is the count of objects currently allocated out of the cache.
- *allocated-objects*. This is the current total number of objects in the cache.
- *object-size*. This is the size of each allocated object. There is overhead to maintaining the cache, so with a 512-byte object and a 4,096-byte page size, it can fit seven objects in a single page and will waste 512-slab overhead bytes per allocation. Slab overhead varies with object size (smaller objects have more objects per allocation and require more overhead to track used versus unused objects).
- *active-slab-allocs*. This is the number of allocations that have at least one of the allocations objects in use.
- *total-slab-allocs*. This is the total number of allocations in the current slab cache.

- *alloc-size*. This is the size of each allocation in units of memory pages. Page size is architecture specific, but the most common size is 4K.
- The last two items are SMP specific. On SMP machines, the slab cache will keep a per-CPU cache of objects so that an object freed on CPU0 will be reused on CPU0 instead of CPU1 if possible. This improves cache performance on SMP systems greatly.
- *limit*. This is the limit of free objects stored in the per-CPU free list for this slab cache.
- *batch-count*. On SMP systems, when the available object list is refilled, instead of doing them one at a time, a batch of objects are taken at a time.

As illustrated in Figure 9.14, the EM also provides operating system information such as CPU utilization, memory utilization, and disk I/O utilization.

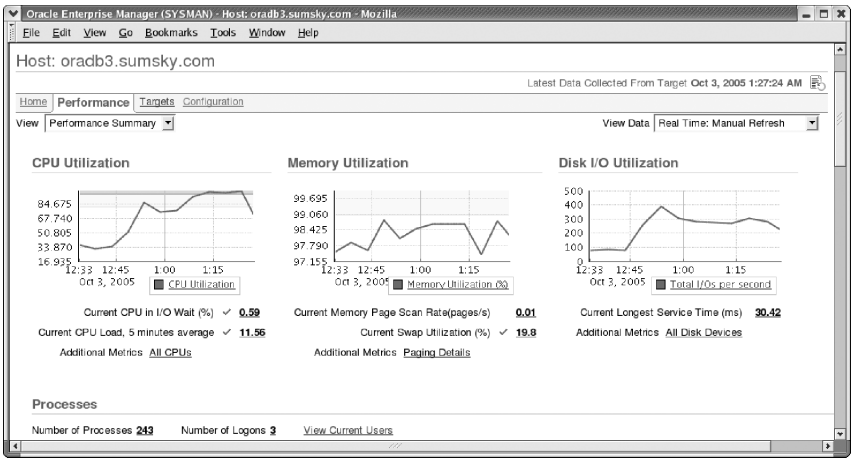


Figure 9.14 *EM operating system Monitoring*

9.20 Automatic workload repository

In Oracle Database 10*g*, Oracle introduced a new process to capture and performance statistics called the Automatic Workload Repository (AWR). For those of us who have used STATSPACK in the past, AWR is an enhanced rewrite of this utility with a more user-friendly interface. Statistics and the entire workload information are collected (snapshot) automatically by the MMON background process every 60 minutes (default) and stored in the wrh$

and wri$ tables in the SYSAUX tablespace. Data collected is retained in the AWR for seven days (default) and then automatically purged. During this period, database performance and workload statistics can be generated into a report by comparing two snapshot periods. Unlike a STATSPACK report, AWR collects data on only two levels: TYPICAL (default) and ALL. These levels are driven by the parameter STATISTICS_LEVEL.

AWR snapshots can also be captured manually using the DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT() procedure and at the ALL level DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT(flush_level=>ALL);

The number of top SQL queries reported by the AWR is also controlled by the STATISTICS_LEVEL parameter. When the value is TYPICAL the top 30 queries are listed, and when the value is ALL the top 100 queries are listed. This can be overridden in Oracle Database 10*g* Release 2 using the following procedure:

```
DBMS_WORKLOAD_REPOSITORY.modify_snapshot_settings(topnsql =>
200);
```

During typical benchmarking cycles, when a set of performance metrics needs to be saved as a baseline for comparisons, the traditional method used by DBAs has been to export performance data. AWR makes this more convenient using the following procedure:

```
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE(<START SNAP ID>,<END
SNAP ID>,<BASLINE NAME><DB ID>);
```

For example, the following procedure will create a baseline QAR1BLINE of the current database instance (default) represented by DB ID in the syntax above.

```
execute DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE
(459,476,'QAR1BLINE');
```

The baseline definition can be validated by querying the DB_HIST_BASELINE table.

```
COL BASELINE_NAME FORMAT A25
SELECT DBID,
       BASELINE_NAME,
```

```
       START_SNAP_ID,
       END_SNAP_ID
FROM   DBA_HIST_BASELINE;

      DBID BASELINE_NAME              START_SNAP_ID END_SNAP_ID
---------- ------------------------- ------------- -----------
4275027223 QAR1BLINE                           459         476
```

Note: Creation of a baseline will override AWR automatic purging for the baseline snapshot range.

While AWR does provide information concerning RAC, statistics are collected from all instances and stored by instance number under a common `SNAP_ID` (`SNAP_ID` is the primary key for the AWR and is generated for every snapshot collection); that is, statistics are collected by instance and not at the global level.

AWR reports or snapshot comparisons can be done either by using a command-line interface by executing an Oracle-provided SQL script (`$ORACLE_HOME/rdbms/admin/awrrpt.sql`) or by using the EM interfaces. An HTML file can be generated and viewed using a browser by either method.

The script provides an option to pick a range of snapshots to compare. Once the range is selected, the report is generated in the default directory. In an RAC environment, a report is generated for one instance at a time.

Tuning using the AWR starts with identifying the top five wait events reported in the first page of the report, illustrated in Table 9.3.

Table 9.3 *AWR Top Five Timed Events*

Top 5 Timed Events

Event	Waits	Time(s)	Percent Total DB Time	Wait Class
db file sequential read	228,989	1,767	57.28	User I/O
CPU time		552	17.89	
log file sync	24,925	208	6.74	Commit
log file parallel write	26,983	201	6.53	System I/O
gc buffer busy	12,071	166	5.37	Cluster

Table 9.3 provides the wait events at the instance level and can contain RAC-related wait events as shown for "`gc buffer busy`." AWR also provides a global-level cache load profile illustrating the global cache activity between the various instances in the cluster.

Table 9.4 *AWR RAC Global Cache Load Profile*

Global Cache Load Profile

	Per Second	Per Transaction
Global Cache blocks received:	99.08	14.50
Global Cache blocks served:	101.06	14.79
GCS/GES messages received:	510.95	74.77
GCS/GES messages sent:	494.37	72.34
DBWR Fusion writes:	8.46	1.24
Estd Interconnect traffic (KB)	22.92	

Like STATSPACK, AWR is also organized by sections. When the AWR report is generated in a RAC environment, the second page of this report relates to RAC. Apart from the performance data illustrated in Tables 9.3 and 9.4, other important data is discussed later in this section.

Table 9.5 *AWR RAC Global Cache Load Profile Formula*

Profile	Formula
Global cache blocks received	(gc current blocks received + gc cr blocks received) / elapsed time
Global cache blocks served	(gc current blocks served + gc cr blocks served) / elapsed time
GCS/GES messages received	(gcs messages received + ges messages received) / elapsed time
GCS/GES messages sent	(gcs messages sent + ges messages sent) / elapsed time
DBWR fusion writes	DBWR fusion writes / elapsed time

Table 9.5 *AWR RAC Global Cache Load Profile Formula*

Estimated interconnect traffic	(((gc cr blocks received + gc current blocks received + gc cr blocks served + gc current blocks served) * db_block_size) + ((gcs messages sent + ges messages sent + gcs msgs received + ges msgs received) * 200) /1024 / elapsed time)

In Table 9.4, the various profiles are calculated using the formula described in Table 9.5.

The formula in Table 9.5 is based on statistic values obtained from `GV$SYSSTAT` or `GV$DLM_MISC` views. Elapsed time is the time between the start and end of the collection period. In the case of an AWR report, the elapsed time is the time between two snapshots being compared. The load profile can be grouped by services using the `GV$SERVICE_STATS` view to obtain a more focused performance metric.

Table 9.6 provides the overall performance of the instance with respect to global cache movement between instances.

Table 9.6 *AWR RAC Global Cache and Enqueue Services*

Global Cache and Enqueue Services - Workload Characteristics

Avg global enqueue get time (ms):	0.3
Avg global cache cr block receive time (ms):	0.9
Avg global cache current block receive time (ms):	1.0
Avg global cache cr block build time (ms):	0.0
Avg global cache cr block send time (ms):	0.1
Global cache log flushes for cr blocks served %:	2.4
Avg global cache cr block flush time (ms):	8.5
Avg global cache current block pin time (ms):	0.1
Avg global cache current block send time (ms):	0.1
Global cache log flushes for current blocks served %:	0.6
Avg global cache current block flush time (ms):	10.8

The average values in Table 9.6 are also based on statistic values from the GV$SYSSTAT and GV$DLM_MISC views. Once the actual values are computed, the average is determined to provide the overall health of the cluster during the snapshot interval. Table 9.7 illustrates the statistic values used in some of the average values included in Table 9.6.

Table 9.7 *AWR RAC Global Cache Load Profile Formula*

Statistic	Formula	Typical Value (ms)	Upper Limit (ms)
Average global cache cr block receive time	10 × gc cr block receive time / gc cr blocks received	4	12
Average global cache current block receive time	10 × gc current block receive time / gc current blocks received	8	30

AWR provides other RAC-related performance statistics in the AWR report, including the following, to name a few:

- RAC report summary
- Global enqueue statistics
- Global CR served stats
- Global Current served statistics
- Global cache transfer stats
- Global cache transfer statistics aggregated per class

Among these, the Global Cache Transfer Stats is an informative section providing details of block transfers between various instances participating in the cluster. As illustrated in Table 9.8, the transfer is broken down between CR and current requests.

Table 9.8, in conjunction with the section on hot blocks, should determine what database-level tuning is required to reduce this movement and reduce block-request contention.

Table 9.9 lists the differences between a STATSPACK and AWR report.

Table 9.8 *Global Cache Transfer Stats*

Global Cache Transfer Stats

- Immediate (Immed) - Block Transfer NOT impacted by Remote Processing Delays
- Busy (Busy) - Block Transfer impacted by Remote Contention
- Congested (Congst) - Block Transfer impacted by Remote System Load
- ordered by CR + Current Blocks Received desc

--

		CR				Current			
Inst No	Block Class	Blocks Received	% Immed	% Busy	% Congst	Blocks Received	% Immed	% Busy	% Congst
2	data block	290	100.00	0.00	0.00	2,135	97.29	0.00	2.71
2	Others	10	100.00	0.00	0.00	3	100.00	0.00	0.00
2	undo header	5	100.00	0.00	0.00	1	100.00	0	0
3	data block	390	100.00	3.07	1.70	4,505	165.29	0.00	13.43
3	Others	19	100.00	0.00	0.00	3	100.00	0.00	0.00
3	undo header	12	100.00	0.00	0.00	1	100.00	0	0

Table 9.9 *AWR versus STATSPACK*

AWR	STATSPACK
It automatically configured by the DBCA.	Manual installation and configuration are required using scripts.
It automatically purges data older than seven days (can be changed).	A manual purge routine is required.
A reporting option is available via both EM and scripts. Output can be in either HTML or ASCII text format.	Reports are generated only via scripts, and the output is always in ASCII text format.
It contains Oracle provided PL/SQL packages to create baselines.	Baselines can be maintained by exporting data or by not purging the existing data.
Statistics are collected by the `MMON` process every 60 minutes (default) and stored in the AWR.	Data is actually collected by the script during execution. The script can be configured to run automatically by scheduling it at regular intervals using `DBMS_JOB` or `DBMS_SCHEDULER`.
Data from the AWR is used by other performance diagnostic tools, such as ADDM and ASH (discussed later in this chapter).	Data collected is only used by the STATSPACK reporter.
It requires an additional Oracle license.	No additional license is required.

9.21 Automatic Database Diagnostic Monitor

The Automatic Database Diagnostic Monitor (ADDM, pronounced "Adam") is a self-diagnostic analysis and reporting tool that is part of Oracle Database 10*g*. Earlier, we discussed the hourly capture (snapshot) of performance statistics by the AWR process. As illustrated in Figure 9.15, ADDM uses these snapshots and provides advice about what and where the problem is, what areas of the system are affected, what has caused performance issues if any, and what can be done to improve the overall performance of the database.

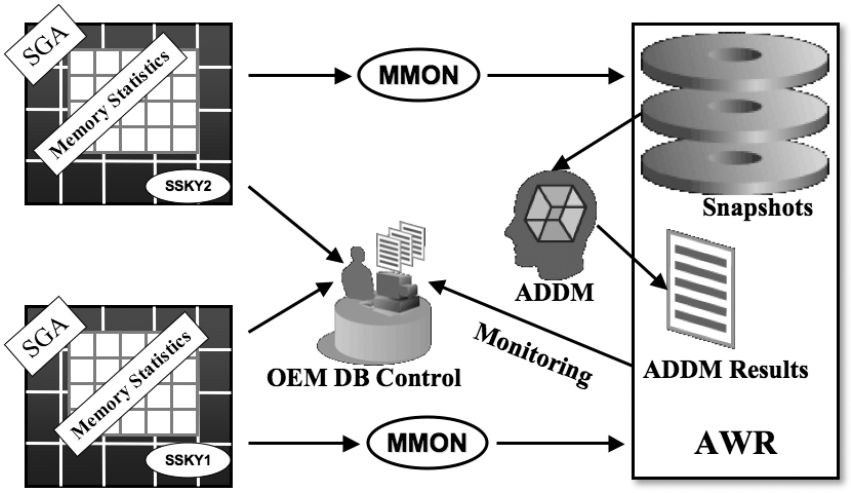


Figure 9.15
AWR and ADDM Process Flow

Like AWR, ADDM can also be generated using the EM or the `addmrpt.sql` script available in the `$ORACLE_HOME/rdbms/admin/` directory. These reports are stored by default for 30 days in the database before being purged. These reports are generated on thresholds predefined (but which can be modified) for a predetermined set of areas. For example, the user I/O is defined by the parameter `DBIO_EXPECTED` and defaults to 1,000 ms. Another parameter that is used to calculate the amount of database time spent is the `DB_ELAPSED_TIME` parameter, which defaults to 0 ms. Both of these parameters can be modified using the `DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER` PL/SQL procedure.

ADDM functionality can be accessed by selecting the "Advisor Central" page from two levels: (1) clustered database performance page and (2) database instance performance page. As illustrated in Figure 9.16, once ADDM is selected, the EM automatically generates an analysis report on the latest

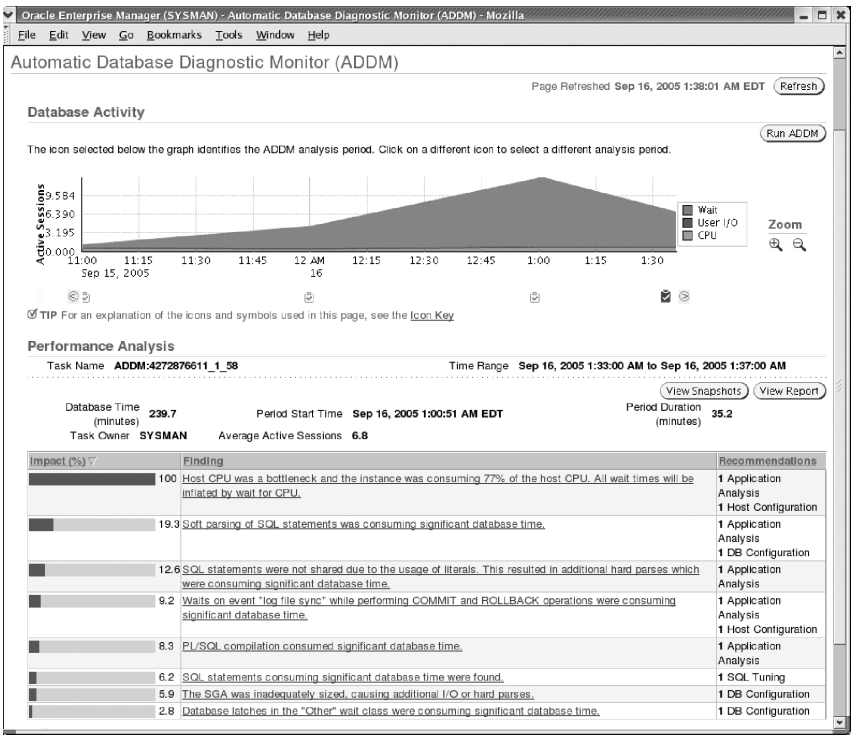


Figure 9.16 *EM ADDM RAC Database Activity*

AWR snapshot available and allows for regeneration of analysis based on another snapshot period.

In Figure 9.16, ADDM provides an overall performance of the database and a list of findings based on the analysis of the snapshots selected. Apart from providing the findings, the analysis also reports on the percentage of impact. In this case, the impact for "Host CPU was a bottleneck" is 100%. To understand the areas of impact and the recommendations for fixing this issue, click on the specific finding. Once this is selected, the EM provides the recommendations (Figure 9.17).

As illustrated in Figure 9.17, the recommendations or actions required are both valid. The system is low on CPU because at the time of testing only one CPU was present. The other recommendation of using ODRM is also valid and helps in implementing a DWM environment.

Note: ODRM and DWM are discussed in Chapter 5.

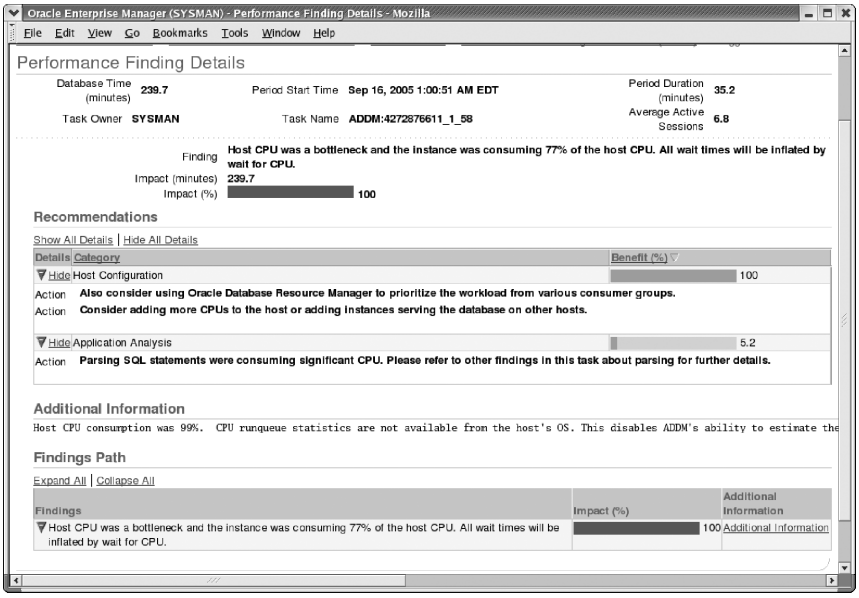


Figure 9.17 *EM ADDM RAC Performance Finding Details*

Similarly, as illustrated in Figure 9.18, ADDM has identified that the `LMSn` background processes are not able to keep up with the workload. Currently, only one `LMSn` background process is present. A good rule of thumb is to configure one `LMSn` process for every two CPUs on the server.

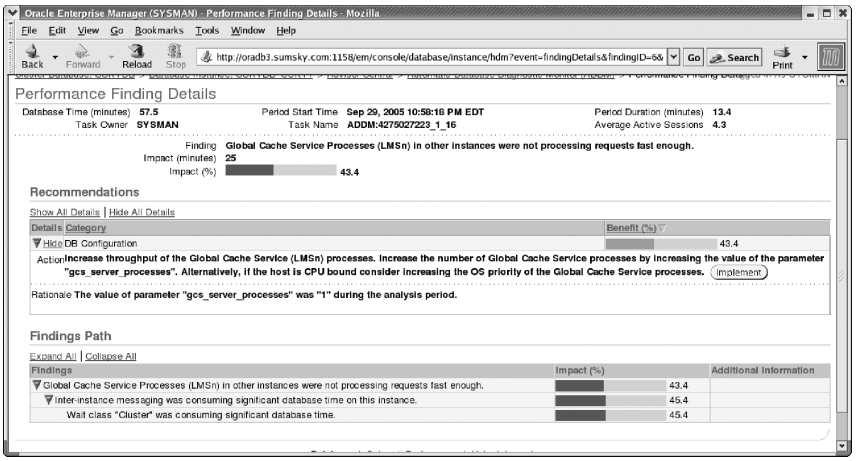


Figure 9.18 *EM ADDM RAC Performance Finding Details*

ADDM is also a good source for identifing high-load SQL statements. Every run of ADDM produces a report containing its analysis and recommendations. These reports are stored by default for 30 days in the database

before being purged. Apart from identifying high-load SQL statements, ADDM will also recommend running SQL advisors on them to obtain optimal performance benefits.

9.22 Active session history

AWR provides snapshots containing performance characteristics of the database; ADDM analyzes this information and provides guidelines and suggestions on how to fix it. However, a one-time occurrence of any issue is not a good indication of any specific performance problem. For instance, when a user reports a bug in his or her application, the developer is interested to find out if this bug is repeatable or reproducible. Only then will the developer spend time investigating it; otherwise, he or she will consider it a minor issue because a one-time occurrence of an issue does not really provide sufficient data to fix the problem. After repeated occurrences data becomes consistent and tunable. Similarly, in a database environment, problems do arise, systems do slow down, and occasional high spikes do occur. However, unless this happens consistently over a length of time (past, present, and future), there is no real concern.

Active session history (ASH, pronounced "ash") tries to bridge this gap. ASH helps perform analysis of transient problems lasting for a few minutes or over various dimensions, such as time, `SQL_ID`, module, action, and so on. As mentioned earlier, unlike the other reactive reporting issues, ASH is based on a sampled history of all events happening in the database. ASH data captured for all active sessions and stored is essentially a fact table (`GV$ACTIVE_SESSION_HISTORY`) in a data warehouse environment, with the columns representing the dimensions of the fact table. In this view, there are about 13 important dimensions of data. However, in the case of ASH, all contents are in memory and are accessing very quickly.

```
SQL> desc gv$active_session_history;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 INST_ID                                            NUMBER
 SAMPLE_ID                                          NUMBER
 SAMPLE_TIME                                        TIMESTAMP(3)
 SESSION_ID                                         NUMBER
 SESSION_SERIAL#                                    NUMBER
 USER_ID                                            NUMBER
 SQL_ID                                             VARCHAR2(13)
```

SQL_CHILD_NUMBER	NUMBER
SQL_PLAN_HASH_VALUE	NUMBER
FORCE_MATCHING_SIGNATURE	NUMBER
SQL_OPCODE	NUMBER
SERVICE_HASH	NUMBER
SESSION_TYPE	VARCHAR2(10)
SESSION_STATE	VARCHAR2(7)
QC_SESSION_ID	NUMBER
QC_INSTANCE_ID	NUMBER
BLOCKING_SESSION	NUMBER
BLOCKING_SESSION_STATUS	VARCHAR2(11)
BLOCKING_SESSION_SERIAL#	NUMBER
EVENT	**VARCHAR2(64)**
EVENT_ID	NUMBER
EVENT#	NUMBER
SEQ#	NUMBER
P1TEXT	VARCHAR2(64)
P1	**NUMBER**
P2TEXT	VARCHAR2(64)
P2	**NUMBER**
P3TEXT	VARCHAR2(64)
P3	**NUMBER**
WAIT_CLASS	**VARCHAR2(64)**
WAIT_CLASS_ID	NUMBER
WAIT_TIME	NUMBER
TIME_WAITED	NUMBER
XID	RAW(8)
CURRENT_OBJ#	NUMBER
CURRENT_FILE#	**NUMBER**
CURRENT_BLOCK#	**NUMBER**
PROGRAM	**VARCHAR2(48)**
MODULE	**VARCHAR2(48)**
ACTION	**VARCHAR2(32)**
CLIENT_ID	VARCHAR2(64)

Like to the AWR and ADDM features, ASH reports can also be generated from the EM console or using the `ashrpt.sql` script located in the `$ORACLE_HOME/rdbms/admin` directory on the database server.

9.23 EM Grid Control

All of the EM-related reports and functionality discussed previously are part of the default dbconsole configuration that is defined by DBCA while creating the database. In a RAC environment, the console is defined on the node where the DBCA was executed.

Apart from the dbconsole, the DBCA provides current database-related information. Another version that captures a performance-related matrix and saves it in a repository is the EM Grid Control (GC). While in this chapter we have been looking at optimization features for the RAC database, the GC supports all tiers of the application:

- Database tier
- Web tier
- Application tier

At the database tier, apart from the traditional functionalities provided by the dbconsole, the GC provides a holistic view of the RAC cluster.

9.23.1 Cluster latency/activity

GC monitors the internode data transfer between the various instances in the cluster. The cluster Interconnects page displays all the interconnects that are configured across the cluster. Both public and private interfaces are identified and shown with their statistics, such as transfer rate and related errors. This screen helps identify configuration problems with the interface setup and performance problems with the interconnect latency.

9.23.2 Topology view

In the Oracle Database 10*g* Release 2 version of GC, Oracle has added a new cluster database home page called the Topology page that displays a graphical view of the entire cluster, including the database instances, listeners, ASM instances, and interfaces. From the Topology page, the administrator can also launch various administrative and configuration functions, like startup and shutdown, monitoring of configurations, and so on.

Figure 9.19
Spotlight on RAC

9.23.3 Spotlight® on RAC

When there are configurations with a large number of nodes, it is convenient to view the cluster as a whole, identify specific issues with any one instance in the cluster, and receive immediate notification by an alarm. In other words, instead of navigating through each instance to find out exactly where the problem is, a console or dashboard kind of tool that provides one view, such as in Figure 9.19, is beneficial. Figure 9.19 is a dashboard screen from Spotlight on RAC. It provides one dashboard view of the cluster and highlights issues with any instance in the cluster. For example, in Figure 9.19, nodes "Venus" and "Earth" are currently high on CPU usage, and an alarm is being raised. Hovering over the alarm will provide details of the alarm, and by drilling down, the specific individual metric relating to the CPU can be identified.

Figure 9.19 also illustrates one view of the three tiers of the database server, the interconnect or global data movement at the top, the specific instances, and the storage or I/O subsystem at the bottom.

9.24 Conclusion

RAC performance depends on how well the application and SQL queries are tuned to access the database. A poorly performing query in a single-instance database will also perform poorly in a RAC environment. Usage of best practices plays an important role.

In this chapter, we looked at the various areas of application and the database, considering the areas that can be improved. In addition, the various ways of identifing specific issues, like blockers, frequency of remastering, and so on, were covered.

The primary goal of the chapter was to identify and help in the diagnostic process of a RAC database. It does not cover all the areas of troubleshooting; several more areas of the database affect the cluster performance directly or indirectly. Please note that an issue discussed in this chapter may not be relevant in other environments and should be tested before implementation.