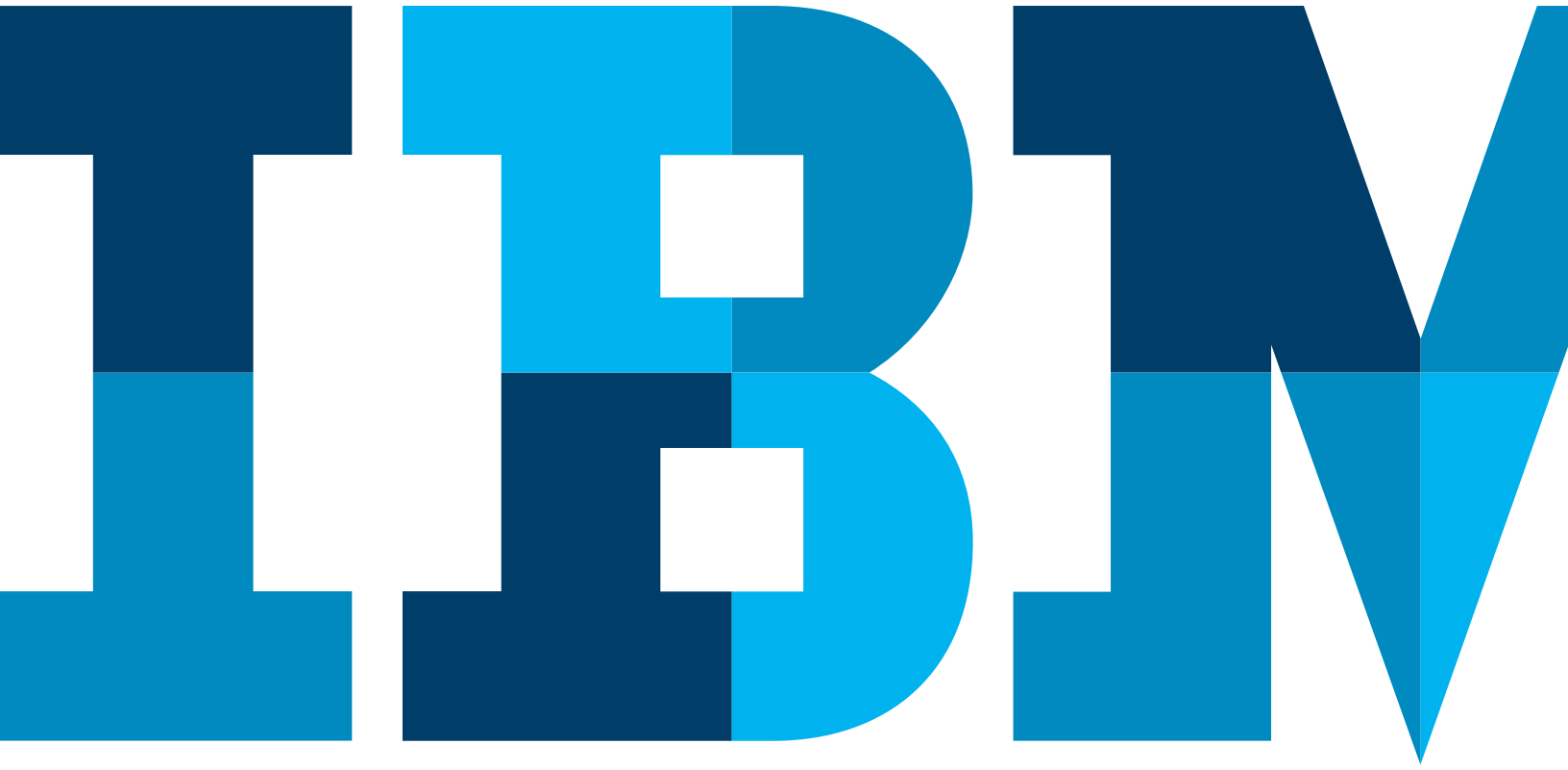
Oracle AWR report in-depth analysis

Determine if your database can benefit from IBM FlashSystem



Contents

Highlights

- Speed up databases to unparalleled rates
- Interpret AWR results to gain valuable, actionable insight
- Utilize flash storage to improve database performance
- Determine optimum conditions for database environments

IBM® FlashSystem™ offers customers the ability to speed up databases to rates that may not be possible even by over provisioning the fastest spinning disk arrays. This paper is intended to provide observations and insights on how to use some native Oracle Database tools to determine the effects that flash storage can have on Oracle Database environments. Oracle utilities—Statspack and now Automatic Workload Repository (AWR) reports—provide database administrators with detailed information concerning a snapshot of database execution time. This snapshot provides statistics on wait events, latches, storage input and output volumes, and timings as well as various views of memory and SQL activities.

The statistics and insights into the memory, input/outputs (I/O), and SQL performance characteristics are invaluable aids in determining if a database is functioning optimally. Unfortunately, there is such an abundance of data in AWR reports that most database administrators (DBAs) feel overwhelmed and may miss important clues to database performance issues.

This paper provides a guide to interpreting AWR results so that even a novice DBA can glean valid, actionable insights from review of an AWR report.

For customers lacking the time to delve into their AWR reports, IBM offers a free service to analyze your AWR reports in order to identify if your database can benefit from IBM FlashSystem.

What is AWR?

The AWR provides a set of tables into which snapshots of system statistics are stored. Generally these snapshots are taken on an hourly basis and include wait interface statistics, top SQL, memory, and I/O information that is cumulative in nature up to the time of the capture. The AWR report process takes the cumulative data from two snapshots and subtracts the earlier snapshot's cumulative data from the later snapshot and then generates a delta report showing the statistics and information relevant for the time period requested. AWR is a more advanced version of the old Statspack reports that has been automated and made integral to Oracle's automated tuning processes for the Oracle Database.

AWR reports are run internally each hour and the findings are reported to the OEM interface. The user can use OEM or manual processes to create a full AWR report, in text or HTML format. These text or HTML reports are what we will be examining in this paper.

What should you know before examining AWR reports?

Before examining AWR reports, DBAs should be familiar with the basic wait events that an Oracle Database may encounter and the typical latches that may cause performance issues, plus be aware of what a typical performance profile looks like for their particular system. Usually by examining several AWR reports from periods of normal or good performance, DBAs can acquaint themselves with the basic performance profile of their database.

Things to notice in the baseline reports include normal levels of specific wait events and latches and the normal I/O profile (normal I/O rates and timings for the database files). Other items to look at are the number and types of sorts, the memory layout, and Process Global Area (PGA) activity levels.

In order to be aware of what waits, latches, and statistics are significant, it is suggested that DBAs become familiar with the Oracle Database Tuning guide and concepts manual. Tuning books by outside authors can also provide more detailed insight into Oracle Database tuning techniques and concepts.

Take a top-down approach

Unless you are looking for specific problems such as a known SQL issue, it is usually best to start with the top data in the AWR report and drill down into the later sections as indicated by the wait events and latch data. The first section of an AWR report shows general data about the instance; look at Figure 1 for an example header section.

```
WORKLOAD REPOSITORY report for

DB Name         DB Id    Instance     Inst Num Startup Time    Release     RAC
------------ ----------- ------------ -------- --------------- ----------- ---
AULTDB        4030696936 aultdb1             1 04-Aug-08 10:16 11.1.0.6.0  YES

Host Name        Platform                         CPUs Cores Sockets Memory(GB)
---------------- -------------------------------- ---- ----- ------- ----------
aultlinux3       Linux IA (32-bit)                   2     1       1       2.97

              Snap Id      Snap Time      Sessions Curs/Sess
            --------- ------------------- -------- ---------
Begin Snap:        91 04-Aug-08 12:00:15        41       1.2
  End Snap:        92 04-Aug-08 13:00:28        47       1.1
   Elapsed:               60.22 (mins)
   DB Time:              139.52 (mins)

Cache Sizes                       Begin        End
~~~~~~~~~~~                  ---------- ----------
               Buffer Cache:     1,312M     1,312M  Std Block Size:         8K
           Shared Pool Size:       224M       224M      Log Buffer:    10,604K
```

Figure 1: AWR report header

The report header provides information about the instance upon which this report was run. The AWR header was expanded to include data about the platform, CPUs, cores, and sockets, as well as memory available in the server. The instance number, whether this is an Oracle RAC (Real Application Clusters) instance or not, and the release level of the database is shown here. The header also includes the startup time as well as the times for the two snapshots used to generate the report. The delta-T between the snapshots is also calculated and displayed.

Following the instance and snapshot data, the cache sizes section gives basic data about the buffer pool, shared pool, standard block, and log buffer sizes.

All of this information in the first part of the AWR header forms the foundation that we draw from to gauge whether a particular statistic is reasonable or not. For example, if we see that the system is an Oracle RAC-based system, then we know to expect Oracle RAC-related statistics to be included in the report. If we see a large number of CPUs, then we might expect to see parallel query related numbers. The knowledge of whether a system is Linux®, Unix®, AIX®, Windows®, or some other supported platform is also valuable in tracking down platform specific issues.

You should pay attention to the duration of a snapshot window against which the report was run. If the window is too long then too much averaging of values could distort the true problem. On the other hand, if the period is too short important events may have been missed. All in all you need to understand why a report was generated: Was it for a specific SQL statement? If so it might be a short duration report. Was it for a non-specific problem we are trying to isolate? Then it could be from 15 minutes to a couple of hours in duration.

Also, pay attention to the number of sessions at the beginning and end of the report; this can tell you if the load was constant, increasing, or decreasing during the report period.

The second section of the report header contains load information and is shown in Figure 2.

Load Profile	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	2.3	7.1	0.63	1.05
DB CPU(s):	0.3	0.9	0.07	0.13
Redo size:	800.5	2,461.8		
Logical reads:	6,307.6	19,396.7		
Block changes:	3.6	10.9		
Physical reads:	2,704.9	8,317.8		
Physical writes:	86.9	267.3		
User calls:	2.2	6.8		
Parses:	2.0	6.1		
Hard parses:	0.0	0.1		
W/A MB processed:	932,965.4	2,868,990.9		
Logons:	0.1	0.2		
Executes:	3.7	11.3		
Rollbacks:	0.1	0.3		
Transactions:	0.3			

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	100.00	Redo NoWait %:	99.97
Buffer Hit %:	96.09	In-memory Sort %:	100.00
Library Hit %:	98.17	Soft Parse %:	97.88
Execute to Parse %:	45.80	Latch Hit %:	99.95
Parse CPU to Parse Elapsd %:	0.00	% Non-Parse CPU:	99.77

Shared Pool Statistics	Begin	End
Memory Usage %:	81.53	85.39
% SQL with executions>1:	79.29	79.48
% Memory for SQL w/exec>1:	76.73	78.19

Figure 2: Load related statistics in header

The critical things to watch for in the load section depend on the type of application issue you are trying to resolve. For example, in the header section in Figure 2 we see a large number of physical reads and logical reads with few block changes; this is a typical profile for a reporting environment such as a data warehouse or decision support system (DSS). Seeing this large number of logical and physical reads should key us to look at I/O-related issues for any database performance problems. An additional sign that this is probably a warehouse or DSS environment is the large amount of Work Area processing (W/A) occurring; this means sorting. A further indicator is that the user calls and parses are low, indicating that the transactions contain few statements and are long-running.

In a predominately online transaction processing system (OLTP) we would expect to see more logical reads, few physical reads, and many user calls, parses, and executes, as well as rollbacks and transactions. Generally speaking, report environments have fewer, longer transactions that utilize the Work Area, while OLTP environments tend to have numerous small transactions with many commits and rollbacks.

The next section of the header shows us the Instance Efficiency percentages. Generally you want these as close to one hundred percent as possible. As you can see, all of our efficiencies are near to 100 percent, with the exception of Execute to Parse and Parse CPU to Parse Elapsed. Because we are dealing with a reporting system, it will probably have a great deal of ad-hoc reports. Because these by their nature are not reusable, we will have low values for parse-related efficiencies in this type of system. In an OLTP-type system where transactions are usually repeated over and over again we would expect the parse-related efficiencies to be high as long as cursor sharing and bind variables were being properly utilized.

If we were to see that the Buffer NOWAIT and Buffer Hit percentages were low (less than 95 percent) we would investigate if the data block buffers were being properly used and if we might need to increase data block buffer sizes.

If the library hit percentage was low we would consider increasing the shared pool allocation, or at least looking into why its percentage was low (it could be improper bind variable usage).

The redo NOWAIT percentage tells us how efficiently our redo buffers are being utilized; if the percentage is low, we would need to look at tuning the redo log buffers and redo logs. If processes are waiting on redo then either the buffers are too small or something is blocking the redo logs from being reused in a proper manner. For example, in an archive log situation if there are insufficient logs then the system may have to wait on the archive log process while it copies a log to the archive location, decreasing the NOWAIT percentage.

The memory sort percentage tells us if our PGA_AGGREGATE_TARGET or, if manual settings are used, SORT_AREA_SIZE, HASH_AREA_SIZE and bitmap settings need to be examined. Numbers less than 100 for the sort percentage indicate that sorts are going to disk. Sorts going to disk are slow and can cause significant performance issues.

The soft parse percentage tells us how often the SQL statement we submit is being found in the cursor caches. This is directly related to proper bind variable usage and how much ad-hoc SQL generation is taking place in our system. Hard parses cause recursive SQL and are quite costly in processing time. In a system where SQL is being reused efficiently this should be near one hundred percent.

Latch hit percent tells us how often we are not waiting on latches. If we are frequently spinning on latches, this value will decrease. If this percentage is low, look for CPU-bound processes and issues with latching.

The Non-Parse CPU percentage tells us how much of the time the CPU is spending on processing our requests verses how much time it is spending doing things like recursive SQL. If this percentage is low, look at the parse-related percentages because they too will be low. When this percentage is low it indicates the system is spending too much time processing SQL statements and not enough time doing real work.

The next section of the header shows us the shared pool statistics. One of the main purposes of the shared pool is to provide a pre-parsed pool of SQL statements that can quickly be reused. This header section shows the amount of memory being utilized for reusable statements. Generally, if 70 to 80 percent (or higher) of memory is being utilized, or higher then good reuse is occurring in the database. If the percentages are less than 70 percent, then the application should be reviewed for proper SQL reuse techniques such as PL/SQL encapsulation and bind variable utilization.

The next few sections of the header really help point where DBAs or tuning users should look for the problems causing performance issues in the database. This next section is shown in Figure 3.

```
Top 5 Timed Foreground Events
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                                           Avg
                                                          wait   % DB
Event                                 Waits     Time(s)   (ms)   time Wait Class
------------------------------ ------------ ----------- ------ ------ ----------
db file sequential read             465,020       3,969      9   47.4 User I/O
DB CPU                                              995          11.9
db file parallel read                 2,251         322    143    3.8 User I/O
db file scattered read               15,268         153     10    1.8 User I/O
gc current block 2-way              108,739         116      1    1.4 Cluster

Host CPU (CPUs:    2 Cores:    1 Sockets:    1)
~~~~~~~~          Load Average
              Begin       End      %User   %System      %WIO     %Idle
            --------- --------- --------- --------- --------- ---------
                0.37      3.05      10.6       6.7      45.3      82.6

Instance CPU
~~~~~~~~~~~~
             % of total CPU for Instance:  14.8
             % of busy  CPU for Instance:  85.0
  %DB time waiting for CPU - Resource Mgr:   0.0

Memory Statistics
~~~~~~~~~~~~~~~~~                            Begin          End
                    Host Mem (MB):       3,041.4        3,041.4
                     SGA use (MB):       1,584.0        1,584.0
                     PGA use (MB):         169.0          301.7
    % Host Mem used for SGA+PGA:          57.64          57.64
```

Figure 3: Wait, CPU, and memory statistics

Of the statistics in this next section of the header, the top five wait statistics are probably the most important. The wait interface in the Oracle Database stores counters for waits and timings for several hundred wait events that instrument the internals of the database. By examining the wait statistics you can quickly find the pinch points for performance in your database. In our example in Figure 4 we can see that the following statistics are the dominant waits:

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
db file sequential read	465,020	3,969	9	47.4	User I/O
db file parallel read	2,251	322	143	3.8	User I/O
db file scattered read	15,268	153	10	1.8	User I/O

Figure 4: Dominant waits after analysis

Obviously with 54 percent of the wait activity (and probably more) being I/O related, our I/O subsystem on this Oracle RAC setup is being stressed.

Note that the CPU is showing 11.9 percent usage during this period. In a normal system the CPU should show the dominant time percentage.

Other wait events that might dominate an Oracle RAC environment could be redo log related, interconnect related, or undo tablespace related. Note that the fifth largest wait was the "gc current block 2-way." This indicates that the same block was being shared back and forth across the interconnect.

Of course because this is a parallel query environment with the parallel query being not only cross-table and cross-index but cross-node, some amount of interconnect related waits are expected. However, if "gc" related waits dominated the top five wait events section, this would indicate there was definite stress on the interconnect and it was a significant source of wait issues.

In this case the predominant wait is the "db file sequential read" which indicates that single block reads (i.e. index reads) are causing issues. Normally this would be resolved by adding more db block cache memory (server memory); however, our system is memory constrained so if we can't remove the waits we would look to reduce the wait time per incident. By increasing the number of disks in the array and increasing the spread of the files causing the reads we could possibly reduce this wait to as small as five milliseconds (maybe lower if we move to a more expensive cached SAN setup), but this would be the limit in a disk-based system. The only way to further reduce the value would be to increase the amount of server memory through a server upgrade or decrease the read latency by moving to IBM FlashSystem. The other read-based waits would also benefit from either more memory or faster I/O subsystems.

The other major wait that is usually seen when an I/O subsystem is stressed is the "db file scattered read" which indicates full table scans are occurring. Full table scans can usually be corrected by proper indexing. However, in DSS or warehouse situations this may not always be possible. In the case of DSS or data warehouses, use of partitioning can reduce the amount of data scanned. However, each disk read is going to require at least five milliseconds and the only way to beat that is through large-scale caching or the use of IBM FlashSystem to reduce latency. Where a disk based system can have latency greater than 5 milliseconds, IBM FlashSystem provides latency as low as 100 microseconds (a 25x improvement).

When the "db file sequential read" or "db file scattered read" are the significant wait sources, then DBAs need to look in the SQL sections of the report to review the top SQL that is generating excessive logical or physical reads. Usually, if a SQL statement shows up in two or more of the SQL subsections, it is a top candidate for tuning actions.

A key indicator for log file stress (redo logs) would be the *log file sync*, *log file parallel write*, *log file sequential write*, *log file single write*, or *log file switch completion* wait events dominating the top five wait event listing; however, you must make sure that the wait is truly from I/O-related issues and not issues such as archive logging before taking proper action. Usually, log file stress occurs when the log files are placed on the same physical disks as the data and index files and can usually be relieved by moving the logs to their own disk array section. However, if high wait times for log-related events occur, then moving the logs to an IBM FlashSystem is indicated. While the AWR report does not show latency for redo log activities, redo log writes can be very latency sensitive in environments with heavy write activity and especially those with single threaded synchronous I/O. In heavy write environments, IBM FlashSystem reduces the latency for redo log writes, thus improving a transactional system's ability to support higher concurrency.

The next section of the report shows the breakdown of the CPU timing and run queue status (Load Average) for the time interval.

```
Host CPU (CPUs:    2 Cores:    1 Sockets:    1)
~~~~~~~~              Load Average
                  Begin       End      %User   %System      %WIO     %Idle
              --------- --------- --------- --------- --------- ---------
                   0.37      3.05      10.6       6.7      45.3      82.6
```

The run queue tells you how many processes are waiting to execute. If the run queue exceeds the number of available CPUs and the CPUs are not idle, then increasing the number of CPUs or upgrading the speed of CPUs is indicated, assuming all other tuning actions, such as reducing recursion, have been accomplished. As you can see from the report section above, our CPU was 83 percent idle during the period while I/O waits were 45 percent, thus CPU stress was not causing the run queue of 3. It was most likely I/O-related wait activity. The other statistics in the section show the amount of time utilized in user and system modes of the CPU, as well as the percentage of time the CPU was idle and the average I/O wait. If the I/O wait percentage is high then increasing the number of disks (after proper tuning has occurred) may help. If you have already tuned SQL and I/O wait is still a large percentage of the total waits, then your best choice is moving to a lower latency I/O subsystem such as IBM FlashSystem.

Following the CPU timing section, the Instance CPU section shows how efficiently this instance was using the CPU resources it was given.

```
 Instance CPU
 ~~~~~~~~~~~~
              % of total CPU for Instance:  14.8
              % of busy  CPU for Instance:  85.0
  %DB time waiting for CPU - Resource Mgr:   0.0
```

This instance utilized the total CPU time available for only 14.8 percent of the time. Of that 14.8 percent, 85 percent of the CPU time was utilized for processing. Because no resource groups are in use in the database, zero percent of the CPU was used for resource management with the resource manager. This again points out that the system was I/O bound, leaving the system basically idle while it waited on disks to serve data.

The last section of the header deals with memory usage.

```
Memory Statistics
~~~~~~~~~~~~~~~~~                          Begin          End
                        Host Mem (MB):     3,041.4      3,041.4
                         SGA use (MB):     1,584.0      1,584.0
                         PGA use (MB):       169.0        301.7
      % Host Mem used for SGA+PGA:          57.64        57.64
```

According to Oracle, you should only use about 60 percent of the memory in your system for Oracle Database; however, as memory sizes increase this old saw is showing its age. Nonetheless, in memory-constrained 32-bit systems such as the one this report came from, 60 percent is probably a good point to shoot for with Oracle Database memory usage. As you can see, this instance is using 57.64 percent so it is pretty close to the 60 percent limit. The rest of the memory is reserved for process and operating system requirements. We can see that our System Global Area (SGA) remained fairly stable at 1,584 megabytes while our PGA usage grew from 169 to 302 megabytes. This again points to the system being a report DSS or data warehouse system utilizing a lot of sort area.

Oracle RAC-specific pages

Once we get out of the header and system profiles area, if you are using Oracle RAC you get to an Oracle RAC-specific section that will not be present if Oracle RAC is not being used. The first part of the Oracle RAC-specific statistics deals with profiling the global cache load. This section of the report is shown in Figure 5.

```
RAC Statistics  DB/Inst: AULTDB/aultdb1  Snaps: 91-92

                              Begin   End
                              ----- -----
              Number of Instances:     2     2


Global Cache Load Profile
~~~~~~~~~~~~~~~~~~~~~~~~~                  Per Second       Per Transaction
                                           ---------------  ---------------
  Global Cache blocks received:                  26.51            81.54
    Global Cache blocks served:                  26.02            80.01
     GCS/GES messages received:                 156.31           480.68
         GCS/GES messages sent:                 157.74           485.06
            DBWR Fusion writes:                   0.01             0.04
 Estd Interconnect traffic (KB)                 481.59


Global Cache Efficiency Percentages (Target local+remote 100%)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Buffer access -  local cache %:   95.44
Buffer access - remote cache %:    0.65
Buffer access -         disk %:    3.91
```

Figure 5: Oracle RAC load profiles

The first part of the listing shows how many instances you started with in the Oracle RAC environment and how many you ended up with. This is important because cross-instance parallel operations would be directly affected with loss of or addition of any instances to the Oracle RAC environment.

The next part of the Oracle RAC report shows the Global Cache Load Profile. The Global Cache Load Profile shows how stressed the global cache may have been during the period monitored. In the report shown above we only transferred a total of 481 kilobytes per second across an interconnect capable of handling 100 megabytes per second, so we were hardly using the interconnect, in spite of our cross-instance parallel operations. This is further shown by the low Buffer Access-Remote cache statistic of 0.65 percent which is telling us that only 0.65 percent of our blocks came from the other node's Oracle RAC instance.

Things to watch out for in this section include severe instance unbalancing where the Global Cache blocks received verses the Global Cache blocks served is way out of alignment (they should be roughly equal). Another possible indication of problems is excessive amounts of Database Writer (DBWR) fusion writes. Fusion writes should only be done for cache replacement, which should be an infrequent operation. If fusion writes are excessive it could indicate inadequately sized db block cache areas, excessive checkpointing, commits, or a combination of all of the above.

The next Oracle RAC-specific section deals with the actual timing statistics associated with the global cache. You need to pay close attention to the various block service timings. If the time it takes to serve a block across the interconnect exceeds the time it would take to read it from disk then the interconnect is becoming a bottleneck instead of a benefit. The section is shown in Figure 6.

```
Global Cache and Enqueue Services - Workload Characteristics
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                    Avg global enqueue get time (ms):      0.2

          Avg global cache cr block receive time (ms):      1.8
     Avg global cache current block receive time (ms):      1.8

            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 %:      0.8
            Avg global cache cr block flush time (ms):     17.5

         Avg global cache current block pin time (ms):      0.0
        Avg global cache current block send time (ms):      0.1
 Global cache log flushes for current blocks served %:      0.0
       Avg global cache current block flush time (ms):     20.0

Global Cache and Enqueue Services - Messaging Statistics
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                    Avg message sent queue time (ms):      1.1
             Avg message sent queue time on ksxp (ms):      1.3
                 Avg message received queue time (ms):      0.1
                   Avg GCS message process time (ms):      0.0
                   Avg GES message process time (ms):      0.0

                            % of direct sent messages:    35.13
                          % of indirect sent messages:    64.34
                       % of flow controlled messages:     0.54
          -------------------------------------------------------------
```

Figure 6: Global cache and enqueue workload section

The most important statistics in this entire section are:

```
     Avg global cache cr block receive time (ms):      1.8
Avg global cache current block receive time (ms):      1.8
```

These should be compared to an AWR report run on the other instance:

```
     Avg global cache cr block receive time (ms):      2.1
Avg global cache current block receive time (ms):      1.7
```

If the numbers on both or all RAC instances aren't similar then this could indicate a problem with the interconnect, either at the OS buffer level or with the NIC or interface cards themselves.

Notice the high flush values in Figure 6; these are not the correct values and probably point to an issue with the way AWR is collecting the data because they are a direct input to the product that results in the receive times.

The Global Enqueue timing numbers should be less than 2-3 milliseconds in most cases. If they get anywhere near 20 milliseconds, as stated in current Oracle documentation, you have a serious issue with the global enqueue services (GES) part of the global dictionary.

The last part of this Oracle RAC-specific section deals with the interconnect:

```
Cluster Interconnect
~~~~~~~~~~~~~~~~~~~~
                                  Begin                                  End
                 ------------------------------------------------   -----------
Interface    IP Address      Pub Source                           IP  Pub Src
----------   --------------- --- -------------------------------  --- --- ---
eth0         11.1.1.1        N   Oracle Cluster Repository
```

You should verify that the interconnect is the proper private interconnect and that it is not a public Ethernet. If you see excessive global cache services (GCS) values in previous sections, be sure that the proper interconnect is being used.

Time breakdown statistics

Another nice addition to the statistics in AWR over Statspack involves the time breakdown statistics that show the components of CPU, OS, and other time fields. The first shown is the CPU statistics breakdown; this is shown in Figure 7.

```
Time Model Statistics                    DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Total time in database user-calls (DB Time): 8371.3s
-> Statistics including the word "background" measure background process
   time, and so do not contribute to the DB time statistic
-> Ordered by % or DB time desc, Statistic name

Statistic Name                                       Time (s) % of DB Time
------------------------------------------ -------------------- ------------
sql execute elapsed time                           8,145.5         97.3
DB CPU                                               995.1         11.9
parse time elapsed                                     7.4           .1
hard parse elapsed time                                5.2           .1
PL/SQL execution elapsed time                          4.8           .1
Java execution elapsed time                            0.7           .0
hard parse (sharing criteria) elapsed time             0.2           .0
sequence load elapsed time                             0.1           .0
repeated bind elapsed time                             0.1           .0
PL/SQL compilation elapsed time                        0.0           .0
failed parse elapsed time                              0.0           .0
hard parse (bind mismatch) elapsed time                0.0           .0
DB time                                            8,371.3
background elapsed time                              214.7
background cpu time                                   75.8
          -------------------------------------------------------------
```

Figure 7: CPU time breakdown

Much of how the CPU seconds are determined is a black box. For example, it is a real mystery how in one section the CPU was only utilized 14.8 percent for this instance and yet shows *sql execute elapsed time* as 8,145.5 seconds when with two CPUs there are only 7,200 seconds of CPU time in an hour. It may be including all calls that completed during the interval, which of course may then include calls whose majority of time was actually outside of the measurement interval. However, that is a topic for another discussion.

In the report excerpt in Figure 6 we see that a majority of the CPU time allotted to us for this measurement interval (97.3 percent) was spent in *sql execute elapsed time* and this really is precisely where we want the CPU to be spending its time. If we were to see that *parse time elapsed* or *hard parse elapsed time* were consuming a large portion of time, it would indicate that we either had an ad-hoc environment with a majority of unique SQL statements or we have an application that is not using bind variables properly. Of course if the CPU was spending its time in any of the other areas for a majority of the reported time, that segment of processing should be investigated.

Operating system statistics

The next section of the AWR report shows operating system-related settings and statistics. Figure 8 shows an example report section for OS statistics.

```
Operating System Statistics                DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> *TIME statistic values are diffed.
   All others display actual values.  End Value is displayed if different
-> ordered by statistic type (CPU Use, Virtual Memory, Hardware Config), Name

Statistic                                     Value         End Value
------------------------- ---------------------- ----------------
BUSY_TIME                                   126,029
IDLE_TIME                                   597,505
IOWAIT_TIME                                 327,861
NICE_TIME                                       766
SYS_TIME                                     48,452
USER_TIME                                    76,784
LOAD                                              0                3
PHYSICAL_MEMORY_BYTES                 3,189,190,656
NUM_CPUS                                          2
NUM_CPU_CORES                                     1
NUM_CPU_SOCKETS                                   1
GLOBAL_RECEIVE_SIZE_MAX                   4,194,304
GLOBAL_SEND_SIZE_MAX                        262,144
TCP_RECEIVE_SIZE_DEFAULT                     87,380
TCP_RECEIVE_SIZE_MAX                      1,048,576
TCP_RECEIVE_SIZE_MIN                          4,096
TCP_SEND_SIZE_DEFAULT                        65,536
TCP_SEND_SIZE_MAX                         1,048,576
TCP_SEND_SIZE_MIN                             4,096
          ------------------------------------------------------------

Operating System Statistics - Detail     DB/Inst: AULTDB/aultdb1  Snaps: 91-92

Snap Time           Load    %busy    %user     %sys    %idle  %iowait
--------------- -------- -------- -------- -------- -------- --------
04-Aug 12:00:15      0.4      N/A      N/A      N/A      N/A      N/A
04-Aug 13:00:28      3.0     17.4     10.6      6.7     45.3     82.6
          ------------------------------------------------------------
```

Figure 8: OS statistics

The OS section of the report gives us the time breakdown in CPU ticks to support the percentages claimed in the other sections of the report. The correlation of reported ticks to actual ticks is still a bit foggy. However, examination of this section still shows that the system being examined is not CPU bound but is suffering from I/O contention because both the idle time and I/O wait time statistics are larger than the busy time value. This part of the report also shows us the TCP/UDP buffer settings which are useful when determining issues in Oracle RAC. One problem often noted is that the I/O wait time reported in this section may not be accurate and should not be used for analysis.

Foreground wait events

The next section of the AWR report shows the foreground wait events, which are wait events that occur in foreground processes. Foreground processes are the user or application-level processes. Figure 9 shows the excerpt from the report we are analyzing.

```
Foreground Wait Class                    DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> s  - second, ms - millisecond -    1000th of a second
-> ordered by wait time desc, waits desc
-> %Timeouts: value of 0 indicates value was < .5%.  Value of null is truly 0
-> Captured Time accounts for        68.9%  of Total DB time       8,371.33 (s)
-> Total FG Wait Time:            4,770.85 (s)  DB CPU time:         995.13 (s)

                                                                    Avg
                                     %Time       Total Wait        wait
Wait Class                     Waits -outs         Time (s)        (ms)  %DB time
-------------------- ---------------- ----- ---------------- -------- ---------
User I/O                     518,267     0            4,449        9      53.1
DB CPU                                                   995               11.9
Cluster                      188,753     9              173        1       2.1
Other                      3,806,446   100              146        0       1.7
Concurrency                    1,854     2                2        1       0.0
Commit                            15     0                1       39       0.0
Application                      740     0                0        0       0.0
System I/O                        40     0                0        3       0.0
Network                      6,970     0                0        0       0.0
Configuration                     0                      0                0.0
          ------------------------------------------------------------

Foreground Wait Events                   DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> s  - second, ms - millisecond -    1000th of a second
-> Only events with Total Wait Time (s) >= .001 are shown
-> ordered by wait time desc, waits desc (idle events last)
-> %Timeouts: value of 0 indicates value was < .5%.  Value of null is truly 0

                                                          Avg
                                     %Time Total Wait     wait    Waits   % DB
Event                          Waits -outs   Time (s)     (ms)     /txn   time
-------------------------- ---------- ----- ---------- ------- -------- ------
db file sequential read       465,020     0      3,969       9    395.8   47.4
db file parallel read           2,251     0        322     143      1.9    3.8
db file scattered read         15,268     0        153      10     13.0    1.8
gc current block 2-way        108,739    11        116       1     92.5    1.4
PX Deq: reap credit         3,247,703   100        107       0 2,764.0    1.3
gc cr grant 2-way              57,265     7         28       0     48.7     .3
gc cr multi block request      22,451     6         23       1     19.1     .3
enq: BF - allocation conte         14    93         14     983      0.0     .2
PX qref latch                 555,843   100          9       0    473.1     .1
IPC send completion sync        1,070    52          8       8      0.9     .1
gc remaster                        22     0          5     221      0.0     .1
```

Figure 9: Foreground wait events

The wait events that accounted for less than 0.1 percent of DB time have been omitted for brevity's sake (the actual listing is two pages long). The thing to note about this section of the report is that we have already looked at the main top five events, which are where we should focus our tuning efforts. However, if you are attempting to tune some specific operation or database section, the other waits in this section may apply to that effort. One thing to mention in an Oracle RAC environment, if you see a large number of waits for the *read by other session* event, this usually indicates your block size or latency is too large, resulting in contention. If you see that *read by other session* is one of the predominant wait events, look to the *Segments by x* sections of the AWR for guidance on which tables and indexes are being heavily utilized and consider moving these segments to a tablespace with a smaller than default block size such as 4K or even 2K or to lower latency storage such as IBM FlashSystem to reduce this contention.

Background wait events

Background wait events, as their name implies, are waits generated by the numerous background processes in the Oracle Database process stack. DBWR, log writer process (LGWR), system monitor process (SMON) and process monitor (PMON) all contribute to the background wait events. The report excerpt, limited to the events with at least 0.1 percent of DB time, is shown in Figure 10.

Background Wait Events DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> ordered by wait time desc, waits desc (idle events last)
-> Only events with Total Wait Time (s) >= .001 are shown
-> %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn	% bg time
control file sequential re	8,336	0	72	9	7.1	33.5
control file parallel writ	1,287	0	31	24	1.1	14.5
db file parallel write	792	0	11	14	0.7	5.3
log file parallel write	701	0	11	15	0.6	4.9
events in waitclass Other	44,191	98	5	0	37.6	2.5
library cache pin	449	0	2	4	0.4	.8
db file sequential read	221	0	2	7	0.2	.8
gc cr multi block request	1,915	0	2	1	1.6	.7
os thread startup	19	0	1	56	0.0	.5
gc cr block 2-way	246	0	0	1	0.2	.2
db file scattered read	18	0	0	12	0.0	.1
db file parallel read	3	0	0	59	0.0	.1
gc current grant 2-way	98	0	0	1	0.1	.1

Figure 10: Background wait events

As we can see, the events that dominate the background waits are also I/O related. If the events that are top in the foreground are similar (such as both being control file related) then that is the I/O area in which we should concentrate the tuning efforts. As we can see from the report excerpt, while the I/O-related waits are similar, the predominant ones in each section are different, indicating a general I/O issue rather than an issue with a specific set of files.

Wait event histograms

In the next section of the AWR, Oracle provides a time-based histogram report for the wait events. If the histogram report was ordered by the predominant wait events by time, it would be more useful; instead it is ordered by event name, making us have to search for the important events. The liberty has been taken to remove the unimportant events from the listing example in Figure 11 for brevity's sake.

Wait Event Histogram DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> Units for Total Waits column: K is 1000, M is 1000000, G is 1000000000
-> % of Waits: value of .0 indicates value was <.05%. Value of null is truly 0
-> % of Waits: column heading of <=1s is truly <1024ms, >1s is truly >=1024ms
-> Ordered by Event (idle events last)

Event	Total Waits	% of Waits <1ms	<2ms	<4ms	<8ms	<16ms	<32ms	<=1s	>1s
control file parallel writ	**1287**					**59.0**	**24.1**	**16.9**	
control file sequential re	**9147**		**23.4**	**21.3**	**23.3**	**22.3**	**6.8**	**2.9**	**.0**
db file parallel read	**2256**			**.3**	**1.0**	**7.4**	**32.6**	**56.8**	**1.9**
db file parallel write	792	.5	.8	4.2	28.7	50.0	8.8	7.1	
db file scattered read	**15K**		**.4**	**2.7**	**31.5**	**59.2**	**5.8**	**.5**	
db file sequential read	**465K**	**.0**	**.6**	**2.2**	**49.5**	**45.0**	**2.3**	**.4**	
gc cr grant 2-way	50K	87.2	11.1	1.3	.3	.2		.0	
gc cr multi block request	24K	59.0	36.8	3.0	.5	.6	.0		
gc current block 2-way	84K	6.5	87.7	5.2	.3	.2	.0		
library cache lock	488	82.8	10.9	4.9	1.0	.2	.2		
library cache pin	4371	77.6	11.1	7.4	3.1	.6	.0		
gcs remote message	274K	28.5	15.4	9.9	11.6	7.5	5.8	21.4	
ges remote message	53K	11.4	3.3	2.7	1.9	1.8	2.1	76.8	

Figure 11: Event time histograms

The most important events have been bolded in the above excerpt. Notice that the read events are more important than the write events. With an Oracle Database, unless we are talking direct writes, undo or redo log writes, or control file writes, Oracle Database uses the concept of delayed block cleanout, only writing blocks to disk when absolutely needed. This delayed block cleanout mechanism means that for most data-related writes we aren't too concerned with write times unless our application does frequent commits and the data is needed nearly immediately after the commit.

Because this application is a read-dominated application, we aren't seeing a lot of redo log and undo tablespace related events. In an OLTP type environment, we would expect to see log writes and log syncs rise to the top in an I/O bound system as dominant events. In systems that generate a lot of transactions, we would also expect to see undo tablespace related events be more prevalent.

By looking at the histograms we can see that our read events are taking anywhere from four milliseconds (ms) to one second to complete. This is a typical disk-based histogram. However, in our histogram the largest number of reads by percent are taking more than 8ms to complete. This indicates disk stress is happening. We could possibly reduce this nearer to 5ms by increasing the number of disks in our disk array. However, you cannot expect to get to less than 5ms read or write times in a disk-based system unless you place a large amount of cache in front of the disks. Another option, which can be more cost effective and enable greater inputs/outputs per second (IOPS), is to use IBM FlashSystem technology which should provide less than 0.5 ms latency.

Service related statistics

Since Oracle Database version 10g, Oracle is increasingly using the concept of a database "service." A service is a grouping of processes that are used to accomplish a common function. For example, all of the parallel query slaves and processes used to provide the results for a series of SQL statements for the same user could be grouped into a service. Figure 12 shows the service related section from our example report.

```
Service Statistics                        DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> ordered by DB Time

                                                                    Physical      Logical
Service Name                    DB Time (s)   DB CPU (s)    Reads (K)    Reads (K)
------------------------------ ------------ ------------ ------------ ------------
aultdb                                 8,344          981        9,769       22,715
SYS$USERS                                 23           12            1           56
SYS$BACKGROUND                             1            0            1           17
aultdbXDB                                  0            0            0            0
          -------------------------------------------------------------

Service Wait Class Stats                   DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Wait Class info for services in the Service Statistics section.
-> Total Waits and Time Waited displayed for the following wait
   classes:  User I/O, Concurrency, Administrative, Network
-> Time Waited (Wt Time) in seconds

Service Name
----------------------------------------------------------------
 User I/O  User I/O  Concurcy  Concurcy     Admin     Admin   Network   Network
Total Wts   Wt Time Total Wts   Wt Time Total Wts   Wt Time Total Wts   Wt Time
--------- --------- --------- --------- --------- --------- --------- ---------
aultdb
   517710      4446       234         1         0         0      5828         0
SYS$USERS
      555         3      1615         1         0         0      1140         0
SYS$BACKGROUND
      350         3      3486         4         0         0         0         0
          ---------------------------------------------------------------
```

Figure 12: Service related statistics

The service related statistics allow you to see which users are consuming the most resources. By knowing this you can concentrate your tuning activities on the hard hitters. In the report excerpt in Figure 12, we see the generic service "aultdb" which holds all the user processes that are non-background and non-sys owned. Because there was only one set of processes (we know this because we are good DBAs that keep tabs on what is happening in our system) we can track the usage back to a user called tpch. From looking at the second half of the report we can see that our user experienced over 517,710 I/O-related waits for a total effective wait time of 4,446 seconds or 8.59 milliseconds per wait. Because we know that the best wait time we can get with a disk-based, non-cached system is 5 milliseconds, a wait time of 8.59 milliseconds shows the disks are experiencing some stress. The higher this type of wait is, the more stress experienced by the I/O subsystem. This section of the report can show where the timing issues are occurring.

The SQL sections

The next sections of the report slice and dice the SQL in the shared pool by several different statistics. By using the waits and other statistics we have discussed so far you can usually figure out which SQL area to examine. A general rule of thumb is that if a SQL statement appears in the top five statements in two or more areas, it is a prime candidate for tuning.

The sections are:

- Total elapsed time
- Total CPU time
- Total buffer gets
- Total disk reads
- Total executions
- Total parse calls
- Total sharable memory
- Total version count
- Total cluster wait time

Let's look at each section and discuss the indicators that would lead you to consider investigating the SQL in each.

Total elapsed time

If a SQL statement appears in the total elapsed time area of the report, this means its CPU time plus any other wait times made it pop to the top of the pile. If for some reason it is at the top of the total elapsed time but not at the top of total CPU time, this indicates that there is an issue with recursion associated with this statement. Generally, you will see the same SQL in both the total elapsed and total CPU time sections. If you see high recursion indicators such as parse ratios that are sub-optimal or in the Instance Activity Statistics (the section following the SQL areas) the recursive calls or recursive CPU usage statistics are high.

Total CPU time

When a SQL statement appears in the total CPU time area this indicates it used excessive CPU cycles during its processing. Excessive CPU processing time can be caused by sorting, excessive functions, or long parse times. Indicators that you should be looking at this section for SQL tuning candidates include high CPU percentages in the services section for the service associated with this SQL (hint—if the SQL is uppercase it probably comes from a user or application; if it is lowercase it usually comes from internal or background processes). To reduce total CPU time, reduce sorting by using multi-column indexes that can act as sort eliminators and use bind variables to reduce parse times.

Total buffer gets

"Total buffer gets" means a SQL statement is reading a lot of information from the db block buffers. Generally speaking, buffer gets (or logical reads in Statspack) are desirable, except when they become excessive. Like excessive disk reads, excessive buffer gets can cause performance issues and they are reduced in the same way. To reduce excessive total buffer gets use partitioning, use indexes, and look at optimizing SQL to avoid excessive full table scans. Total buffer gets are typified by high logical reads, high buffer cache hit ratio (when they are driven by a poor selectivity index), and high CPU usage.

Total disk reads

"High total disk reads" mean a SQL statement is reading a lot of information from disks rather than from the db block buffers. Generally speaking, disk reads (or physical reads in Statspack) are undesirable, especially when they become excessive. Excessive disk reads cause performance issues. To reduce excessive disk reads, use partitioning, use indexes, and look at optimizing SQL to avoid excessive full table scans. You can also increase the db buffer cache if memory is not an issue. Total disk reads are typified by high physical reads, low buffer cache hit ratio, and low CPU usage with high I/O wait times. If disk reads are a part of your database (such as DSS or data warehouses where full table scans are a natural result of their structure), then moving to IBM FlashSystem will improve your performance, sometimes dramatically.

Total executions

High total executions can be an indicator that you are doing something correct in the SQL in the database. Statements with high numbers of executions usually are being properly reused. However, be sure that statements with high numbers of executions are supposed to be executed multiple times, an example would be a SQL statement executed over and over again in PL/SQL or Java, or C routine in a loop when it should only execute once. Statements with high executions and high logical and/or physical reads are candidates for review to be sure they are not being executed multiple times when a single execution would serve. If the database is seeing excessive physical and logical reads or excessive I/O wait times, then look at the SQL statements that show excessive executions and show high physical and logical reads.

Parse calls

Whenever a statement is issued by a user or process, regardless of whether it is in the SQL pool, it undergoes a parse. The parse can be a hard parse or a soft parse. If it cannot find an identical hash signature in the SQL pool it does a hard parse with loads of recursive SQL and all the rest of the parse baggage. If it finds the SQL in the pool then it simply does a soft parse with minimal recursion to verify user permissions on the underlying objects. Excessive parse calls usually go with excessive executions. If the statement is using what are known as unsafe bind variables then the statement will be reparsed each time. If the header parse ratios are low, look here and in the version count areas.

Shareable memory

The shareable memory area provides information on SQL statements that are reused and the amount of memory in the shared pool that they consume. Only statements with 1,048,576 bytes of shared memory usage are shown in the report. Usually, high memory consumption is a result of poor coding or overly large SQL statements that join many tables. In a DSS or data warehouse (DWH) environment, large complex statements may be normal. In an OLTP database, large or complex statements are usually the result of over-normalization of the database design, attempts to use an OLTP system as a DWH or DSS, or poor coding techniques. Usually large statements will result in excessive parsing, recursion, and large CPU usage.

Version count

High version counts are usually due to multiple identical-schema databases, unsafe bind variables, or software bugs. In Oracle Database 9i there are bugs that result in unsafe bind variables driving multiple versions. Multiple versions eat up SQL memory space in the shared pool. High version counts can also result in excessive parsing. Setting the undocumented parameter "_sqlexec_progression_cost" to higher than the default of 1,000 decreases versioning in susceptible versions. High values for sharable memory in the SQL pool can indicate issues if you aren't seeing good performance along with high sharable memory for statements with executions greater than 1.

Cluster wait time

As the name implies, the cluster wait time will only be present if you are using an Oracle RAC system. SQL that transfers a high number of statements across the interconnect will be listed in this section. High levels of block transfer occur if the block size is too large, the db caches on each server are too small, or the SQL is using too much of the table data. Large update statements may appear here because updates require block transfers in many cases for current blocks. High levels of GC-type wait events indicate you need to check this section for causative SQL statements.

Instance activity statistics

The next section deals with instance activity statistics. The biggest problem with the instance activity statistics is that there are so many of them and many are not useful except in specific tuning situations that may never occur in a normal database. The example excerpt from the report we have been examining is in Figure 13. The excerpt has had the statistics that aren't normally a concern removed.

Instance Activity Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92

Statistic	Total	per Second	per Trans
CPU used by this session	77,997	21.6	66.4
CPU used when call started	288,270	79.8	245.3
DB time	2,547,336	705.0	2,168.0
Effective IO time	909,644	251.8	774.2
Number of read IOs issued	27,685	7.7	23.6
SQL*Net roundtrips to/from clien	6,970	1.9	5.9
bytes received via SQL*Net from	2,385,638	660.2	2,030.3
bytes sent via SQL*Net to client	2,595,626	718.4	2,209.0
consistent gets	22,777,682	6,303.9	19,385.3
consistent gets - examination	6,073,207	1,680.8	5,168.7
consistent gets direct	3,277,142	907.0	2,789.1
consistent gets from cache	14,648,585	4,054.1	12,466.9
consistent gets from cache (fast	193,221	53.5	164.4
db block changes	12,812	3.6	10.9
db block gets	13,389	3.7	11.4
db block gets from cache	13,364	3.7	11.4
db block gets from cache (fastpa	3,512	1.0	3.0
dirty buffers inspected	825	0.2	0.7
enqueue timeouts	40	0.0	0.0
execute count	13,287	3.7	11.3
free buffer inspected	556,747	154.1	473.8
free buffer requested	731,667	202.5	622.7
gc CPU used by this session	11,859	3.3	10.1
gc blocks lost	0	0.0	0.0
gc cr block build time	1	0.0	0.0
gc cr block flush time	7	0.0	0.0
gc cr block receive time	66	0.0	0.1
gc cr block send time	3	0.0	0.0
gc cr blocks received	361	0.1	0.3
gc cr blocks served	522	0.1	0.4
gc current block flush time	2	0.0	0.0
gc current block pin time	205	0.1	0.2
gc current block receive time	16,726	4.6	14.2
gc current block send time	577	0.2	0.5
gc current blocks received	95,445	26.4	81.2
gc current blocks served	93,484	25.9	79.6
index fast full scans (direct re	90	0.0	0.1
index fast full scans (full)	4	0.0	0.0
index fast full scans (rowid ran	90	0.0	0.1
index fetch by key	3,086,965	854.3	2,627.2
index scans kdiixs1	29,551	8.2	25.2
leaf node 90-10 splits	19	0.0	0.0
leaf node splits	26	0.0	0.0
opened cursors cumulative	13,077	3.6	11.1
parse count (failures)	2	0.0	0.0
parse count (hard)	153	0.0	0.1
parse count (total)	7,202	2.0	6.1
parse time cpu	227	0.1	0.2
parse time elapsed	399	0.1	0.3
physical read IO requests	550,974	152.5	468.9
physical read bytes	32,562,569,216	9,011,916.7	27,712,824.9
physical read total IO requests	605,019	167.4	514.9
physical read total bytes	32,711,421,952	9,053,112.7	27,839,508.0
physical read total multi block	30,330	8.4	25.8
physical reads	9,773,380	2,704.9	8,317.8
physical reads cache	572,745	158.5	487.4
physical reads cache prefetch	153,965	42.6	131.0
physical reads direct	3,402,178	941.6	2,895.5
physical reads direct temporary	124,434	34.4	105.9
physical reads prefetch warmup	58,580	16.2	49.9
physical write IO requests	4,983	1.4	4.2
physical write bytes	1,037,123,584	287,031.1	882,658.4
physical write total IO requests	15,031	4.2	12.8
physical write total bytes	1,085,801,472	300,503.1	924,086.4
physical write total multi block	4,062	1.1	3.5
physical writes	314,090	86.9	267.3
physical writes direct	124,459	34.4	105.9
physical writes direct (lob)	0	0.0	0.0
physical writes direct temporary	124,434	34.4	105.9
physical writes from cache	2,143	0.6	1.8
physical writes non checkpoint	124,952	34.6	106.3
recursive calls	78,415	21.7	66.7
recursive cpu usage	77,189	21.4	65.7
redo entries	7,832	2.2	6.7
redo log space requests	2	0.0	0.0
redo log space wait time	28	0.0	0.0
redo size	2,892,568	800.5	2,461.8
redo synch time	66	0.0	0.1
redo synch writes	72	0.0	0.1
redo wastage	196,192	54.3	167.0
redo write time	1,110	0.3	0.9
redo writes	701	0.2	0.6
rollback changes - undo records	0	0.0	0.0
session cursor cache hits	12,415	3.4	10.6
session logical reads	22,791,070	6,307.6	19,396.7
sorts (memory)	3,875	1.1	3.3
sorts (rows)	1,460,468	404.2	1,243.0
summed dirty queue length	3,284	0.9	2.8
table fetch by rowid	1,322,667	366.1	1,125.7
table fetch continued row	13	0.0	0.0
table scan blocks gotten	2,780,775	769.6	2,366.6
table scan rows gotten	158,164,979	43,773.3	134,608.5
table scans (direct read)	776	0.2	0.7
table scans (long tables)	776	0.2	0.7
table scans (rowid ranges)	776	0.2	0.7
table scans (short tables)	2,255	0.6	1.9
transaction rollbacks	0	0.0	0.0
undo change vector size	1,870,904	517.8	1,592.3
user I/O wait time	445,246	123.2	378.9
user calls	7,943	2.2	6.8
user commits	794	0.2	0.7
user rollbacks	381	0.1	0.3
workarea executions - onepass	6	0.0	0.0
workarea executions - optimal	2,323	0.6	2.0

Instance Activity Stats - Absolute Values
DB/Inst: AULTDB/aultdb1 Snaps: 91-9
-> Statistics with absolute values (should not be diffed)

Statistic	Begin Value	End Value
session pga memory max	544,192,924	4,940,081,136
session cursor cache count	2,266	8,279
session uga memory	73,033,165,084	3.393545E+11
opened cursors current	48	54
workarea memory allocated	0	16,041
logons current	41	47
session uga memory max	4,427,536,236	5,963,059,828
session pga memory	390,773,148	826,689,340

Instance Activity Stats - Thread Activity
DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> Statistics identified by '(derived)' come from sources other than SYSSTAT

Statistic	Total	per Hour
log switches (derived)	1	1.00

Figure 13: Instance activity statistics

It is best to focus on the larger summary type statistics, at least at first, when dealing with this section of the reports. Even the pared down list in Figure 12 still has many entries that may not really help novices find problems with their databases. One of the biggest hurdles to understanding the statistics in the Instance Activity section is knowing what the time units are for the time based statistics. Generally they will be reported in milliseconds; so, for example, the DB time value of 2,547,336 corresponds to 2,547.336 seconds out of a possible 7200 (there are two equivalent CPUs) in an hour, yielding a percentage of total time of 35 percent. Of that 2,547 seconds, 910 (effective I/O time) were spent doing I/O related items, so only 1,637 seconds of processing time or 22 percent of available time. Looking at other CPU-related timings, parsing took another 399 milliseconds and recursive SQL took 77,189 for a total non-processing time of 77,588 rounded up to 78 seconds. That means only 1559 seconds or 21 percent of total CPU time was used to do actual work on our queries. Of course there are other, non-database activities that also eat a bit of CPU time, dropping the total to the reported 18 or so percent.

So what else is contained in the mine of data? We have an *effective I/O time* and the number of I/Os issued. From this we can see that each I/O cost an effective time of 32 milliseconds. No wonder we spent about 45 percent of the time waiting on I/O!

By looking at *SQLNet roundtrips* we can tell if our application is making effective use of array processing. If it is taking hundreds or thousands of roundtrips per transaction then we really need to examine how our application is handling arrays. By default, languages like C and Java only process 10-20 records at a time, SQL*Plus defaults to 10. By increasing array processing via precompiler flags or by the "SET ARRAYSIZE" command in SQL*Plus we can greatly reduce round-trips and improve performance.

Bytes sent and received to and from the clients via SQLNet can be used with roundtrips to see how large a chunk is being shipped between the client and the server, allowing insight into possible network tuning. In addition, this information can be used to see if the network is being strained (generally speaking 1 gigabit Ethernet can handle about 100 megabytes per second of transfers).

Consistent get statistics

Consistent gets deal with logical reads and can be heavy weight (using two latches as in a normal consistent get) or light weight (using one latch as in consistent get—examination). Large numbers of consistent gets can be good or, if they are excessive because of poor index or database design, bad because they can consume CPU resources best used for other things. These statistics are used in conjunction with others, such as those involving your heavy hitter SQLs, to diagnose database issues.

DB block get statistics

DB block gets are current mode gets. A current mode get is for the data block in its current incarnation, with incarnation defined as all permanent changes applied (for example, if the database shut down now and restarted, this is the block you would get). Now, this can either be from cache, from another instance cache, from the file system cache, or from disk. Sometimes it will result in a disk read or a block transfer from another instance cache because there can only be one version of the current block in the cache of an instance at a time.

Dirty block statistics

The dirty buffers inspected statistic tells you how many times a dirty buffer (one that has changes) was looked at when the processes were trying to find a clean buffer. If this statistic is large then you probably don't have enough buffers assigned, because the processes are continually looking at dirty buffers to find clean ones.

Enqueue statistics

The enqueue statistics dealing with deadlocks, timeouts, and waits tell you how often processes were waiting on enqueues and if they were successful. High numbers of enqueue deadlocks indicate there may be application locking issues; high numbers of waits and failures indicate high levels of contention. You need to look in the enqueue section of the report to see the specific enqueues that are causing the problems.

Execution count

The *execute count* statistics are used with other statistics to develop ratios to show how much of a specific resource or statistic applies to a single execution on the average. This can be misleading, however, if there are several long-running transactions and many short supporting transactions. For example, a large DSS query that requires a number of recursive SQL operations to parse it will drive the executions up, but you are really only interested in the large DSS query and not the underlying recursive transactions, except as they contribute to the DSS transaction itself.

Free buffer statistics

The *free buffers requested* verses the *free buffers inspected* statistics show how many buffers, while not actually dirty, were being used by other processes and had to be skipped when searching for a free buffer. If the free buffers inspected is overly large and the statistic *dirty buffers inspected* is also large, then look at commit frequency as well as possibly increasing the total db block buffers because the cache is probably congested.

GC statistics (global cache)

The GC statistics show the components that make up the send times for the consistent read (CR) and current blocks. The statistics for build, flush, and send for the respective type of block (CR or current) are added together and divided by the blocks of that type sent to determine the latency for that operation. The receive times can be divided by the number of blocks received to determine that latency (and should be compared with the send latency as calculated from the other instance's AWR report). By seeing the components of the latency for send operations you can determine if the issue is internal (build or flush times are large) or external (the send time is large). The GC and GES statistics will only be present if Oracle RAC is being utilized. Remember that if send or receive times are greater than the average disk latency, then the interconnect has become a source of performance bottlenecks and needs to be tuned or replaced with a higher-speed and larger-bandwidth interconnect such as Infiniband. If only one node is showing issues (send times excessive point to this node, receive times excessive point to the other nodes) then look to excessive load, TCP buffer settings, or NIC card issues on that node.

The global enqueue statistics haven't been shown because they haven't been a large source of performance issues. If the messaging shows large latencies, it will also be shown in the global cache services because global cache activity depends on the global enqueue service.

Index scan statistics

There are two main index scan statistics:

Index fetch by key—This statistic is incremented for each "INDEX (UNIQUE SCAN)" operation that is part of a SELECT or DML statement execution plan.

Index scans kdiixs1—This statistic is incremented for each index range scan operation that is not one of the types: index fast full scans, index full scan, and index unique scan.

By comparing the two values you get an idea of the ratio of single index lookups verses range scan. In most systems, single index lookups should predominate because they are usually more efficient. However, in DSS or DWH, systems scans or fast scans may become the dominant type of index activity.

Leaf node statistics

The leaf node statistics refer to index leaf nodes and tell you how much insert activity is happening in your database. The 10-90 splits show activity for monotonically increasing indexes (those that use sequences or dates, generally speaking) and the 50-50 splits show other types of index activity such as text or random value indexes. If you see heavy 10-90 split operations then you might want to look at index management operations to be sure your indexes aren't getting too broad due to excessive unused space in your sequence or date based indexes. Usually index rebuilds are only required in databases that have monotonically increasing indexes that also undergo large amounts of random deletions resulting in numerous partially filled blocks.

Open cursors

The *open cursors cumulative* statistic is used with other statistics to calculate ratios for resources used per cursor or cursors open per login, for example.

Parse statistics

The parse statistics are used to show how efficiently you are using parses. If you have large numbers of *parse count (failures)* or large numbers of *parse count (hard)* it could indicate a large number of ad-hoc queries. A large number of hard parses (greater than 10 percent of parses) indicates that the system probably isn't using bind variables efficiently. If there is a large discrepancy between *parse CPU* and *parse Elapsed* times it indicates that the system is overloaded and may be CPU bound.

Physical read and write statistics

For the physical reads and writes statistics we will look at their definitions from the Oracle Database 10g Reference manual:

Physical reads—Total number of data blocks read from disk. This value can be greater than the value of "physical reads direct" plus "physical reads cache" because reads into process private buffers are also included in this statistic.

Physical read bytes—Total size in bytes of all disk reads by application activity (and not other instance activity) only.

Physical read I/O requests—Number of read requests for application activity (mainly buffer cache and direct load operation) which read one or more database blocks per request. This is a subset of the "physical read total I/O requests" statistic.

Physical read total bytes—Total size in bytes of disk reads by all database instance activity, including application reads, backup and recovery, and other utilities. The difference between this value and "physical read bytes" gives the total read size in bytes by non-application workload.

Physical read total I/O requests—Number of read requests which read one or more database blocks for all instance activity, including application, backup and recovery, and other utilities. The difference between this value and "physical read total multi block requests" gives the total number of single block read requests.

Physical read total multi block requests—Total number of Oracle Database instance read requests which read in two or more database blocks per request for all instance activity, including application, backup and recovery, and other utilities.

Physical reads cache—Total number of data blocks read from disk into the buffer cache. This is a subset of the "physical reads" statistic.

Physical reads direct—Number of reads directly from disk, bypassing the buffer cache. For example, in high bandwidth, data-intensive operations such as parallel query, reads of disk blocks bypass the buffer cache to maximize transfer rates and to prevent the premature aging of shared data blocks resident in the buffer cache.

Physical reads prefetch warmup—Number of data blocks that were read from the disk during the automatic prewarming of the buffer cache.

Physical write bytes—Total size in bytes of all disk writes from the database application activity (and not other kinds of instance activity).

Physical write I/O requests—Number of write requests for application activity (mainly buffer cache and direct load operation) which wrote one or more database blocks per request.

Physical write total bytes—Total size in bytes of all disk writes for the database instance, including application activity, backup and recovery, and other utilities. The difference between this value and "physical write bytes" gives the total write size in bytes by non-application workload.

Physical write total I/O requests—Number of write requests which wrote one or more database blocks from all instance activity, including application activity, backup and recovery, and other utilities. The difference between this stat and "physical write total multi block requests" gives the number of single block write requests.

Physical write total multi block requests—Total number of Oracle Database instance write requests which wrote two or more blocks per request to the disk for all instance activity, including application activity, recovery and backup, and other utilities.

Physical writes—Total number of data blocks written to disk. This statistic's value equals the sum of the "physical writes direct" and "physical writes from cache" values.

Physical writes direct—Number of writes directly to disk, bypassing the buffer cache (as in a direct load operation).

Physical writes from cache—Total number of data blocks written to disk from the buffer cache. This is a subset of the "physical writes" statistic.

Physical writes non checkpoint—Number of times a buffer is written for reasons other than advancement of the checkpoint. Used as a metric for determining the I/O overhead imposed by setting the FAST_START_IO_TARGET parameter to limit recovery I/Os (Note that FAST_START_IO_TARGET is a depreciated parameter). Essentially this statistic measures the number of writes that would have occurred had there been no checkpointing. Subtracting this value from "physical writes" gives the extra I/O for checkpointing.

Recursive statistics

The *recursive calls* statistics can be used in ratio with the *user calls* statistic to get the number of *recursive calls* per user call. If the number of recursive calls is high for each user call then this indicates you are not reusing SQL very efficiently. In our example printout the ratio is about 10 to 1, which is fine. If the ratio was 50 to 1 or greater it would bear investigation. Essentially, you need to determine what is a good ratio of *recursive calls* to *user calls* for your system; it will depend on the number of tables on average in your queries, the number of indexes on those tables, and whether or not the Oracle Database has to reparse the entire statement or if it can instead use a soft parse. This ratio is actually reported in the header information. We have already shown how the *recursive CPU* statistic is used with the CPU usage and other CPU related timings.

Redo related statistics

The redo-related statistics can be used to determine the health of the redo log activity and the LGWR processes. By using *redo log space wait time* divided by *redo log space requests* you can determine the wait time per space request. If this time is excessive it shows that the redo logs are under I/O stress and should be moved to IBM FlashSystem. In a similar calculation the *redo synch time* can be divided by the *redo synch writes* to determine the time taken during each redo sync operation. This too is an indicator of I/O stress if it is excessive. A final indicator of I/O stress is a ratio of redo *write time* to *redo writes*, giving the time for each redo write. The *redo wastage* statistic shows the amount of unused space in the redo logs when they were written; excessive values of *redo wastage* per *redo write* indicates that the LQWR process is being stressed. The *rollback changes—undo records* statistics are actually *rollback changes—undo records applied*. According to Jonathan Lewis, if a session's "*user rollbacks*" is large, but its "*rollback changes—undo records applied*" is small (and those numbers are relative to your system) then most of the rollbacks are doing nothing. So by comparing these two metrics you can determine, relative to your system, if you have an undo issue. Undo issues deal with rollback commands either explicit or implicit. Explicit are generated by issuing the rollback command, while implicit can be from DDL, DCL or improper session terminations.

Session cursor statistic

The *session cursor cache hits* statistic shows how often a statement issued by a session was actually found in the session cursor cache. The session cursor cache is controlled by the *session_cached_cursors* setting and defaults (usually) to 50. If you see that the ratio of *session cursor cache hits/user calls+recursive calls* is low then increase the setting of *session_cached_cursors*. A majority of the time, settings from 100 to 150 or higher are recommended.

Sort statistics

The sorts statistics: sorts(rows), sorts(memory), and sorts(disk) show how the system is doing sorts. In later versions you may see sorts(disk) replaced by the *workarea executions—one pass* and *workarea executions multipass* statistics. Ideally you want no *sorts(disk)* or *workarea executions—one pass* or *workarea executions—mulitpass*; however, in reality this may be impossible to achieve, so seek to set *sort_area_size*, *hash_area_size*, *merge_bitmap_area_size*, *create_bitmap_area_size*, or *pga_aggregate_target* to large enough values to reduce sorts to disk as much as possible. Note that no statistic really tracks bitmap, hash, or global temporary table operations that overflow to disk, so it is possible to get temporary tablespace IOPS while having zero values for all disk-related sort and workarea statistics. The sort segment histogram section of the report will help you determine settings for the sort individual or the PGA aggregate parameter settings.

Summed dirty queue length

The *summed dirty queue length* statistic can be used in concert with the *physical writes from cache* statistic to determine if the DBWR process is being stressed. If the ratio of summed *dirty queue length* to *physical writes from cache* is greater than 100 then more DBWR processes are needed.

Table fetch statistics

The table fetch statistics:

table fetch by rowid	1,322,667	366.1	1,125.7
table fetch continued row	13	0.0	0.0
table scan blocks gotten	2,780,775	769.6	2,366.6
table scan rows gotten	158,164,979	43,773.3	134,608.5
table scans (direct read)	776	0.2	0.7
table scans (long tables)	776	0.2	0.7
table scans (rowid ranges)	776	0.2	0.7
table scans (short tables)	2,255	0.6	1.9

… provide details on how table data has been accessed. The three most important statistics are:

Table fetch by rowid — This is the cumulative number of rows fetched by index lookup of rowid.

Table scan rows gotten — This shows the number of rows retrieved via table scan operations (full table scans).

Table fetch continued row — This shows the number of row fetches that resulted in a continued row read, essentially doubling (or more) the I/Os. This could be an indication of chained rows, chained blocks, or BLOB activity.

Depending on your database type, you could have more index-based reads (usually OLTP) or more table scan-based reads (DWH, DSS). It is important to have a feeling for the ratio of these two statistics (index verses scan rows) so any change in the profile will alert you to possible index issues. Also monitoring the ratio of continued row reads to the sum of scan and index row reads will inform you if you are getting excessive chained row activity.

Table scans (direct reads) are usually indicative of parallel query activity. Table scans (rowed ranges) are usually also caused by parallel query operations.

The two table type scans, long and short, are based on whether a table is less than a certain percentage of the size of your db cache size. For example, some releases set the boundary between short and long table scans at 2 percent of the db cache size. Generally, short table scans should be much greater than long table scans in a properly indexed environment.

Transaction rollback

The *transaction rollbacks* statistic is for rollbacks that actually do work. This statistic will also track the rollbacks done automatically, for example, when an update occurs for multiple rows but has to be backed out and restarted by the Oracle Database because of blocking locks. Because this attempt to update-block-rollback may occur several times for a single transaction, you will see several transaction rollback increments even though a single transaction actually occurred. If this statistic is high, then check for conflicting transactions that lock many rows.

Undo change vector statistic

The *undo change vector size* statistic is a cumulative count in bytes of the size of undo records. By calculating a ratio of *undo change vector size* to *user calls* you can get an idea of the amount of undo being generated per user call. If the undo being generated is large then look for excessive write activity to the undo tablespace because this could be slowing down your transactions and causing stress on the I/O subsystem. If your application generates large amounts of undo by nature, consider moving the undo tablespace to IBM FlashSystem.

User statistics

The *user I/O wait time* statistic is the cumulative I/O wait time. Use this statistic with *user calls* to determine the average I/O wait time per user call. This can also be used with the *physical reads* plus the *physical writes* to determine the average I/O wait time per I/O. If the I/O wait time becomes an issue, your best solution is to move hot tables and indexes onto IBM FlashSystem.

The user commits and user rollbacks are used with other statistics to determine weighting. If the number of *user rollbacks* is high compared to *user commits*, look to ad-hoc SQL or improper session termination as the possible cause.

Work area statistics

The work area statistics have already been discussed in the section on sort statistics. Essentially the most desired workarea statistic is *workarea executions—optimal* because these were done within the PGA_AGGREGATE_TARGET settings in memory. Any of the other workarea statistics indicate a sort to disk. If there seem to be excessive *workarea executions-optimal* then look to eliminate unneeded sorts such as distincts from improper joins, sorts that could be eliminated using a multi-column index, or unneeded order by and group by operations.

Instance activity stats — absolute values

The absolute values show the high water marks for various memory, login, and other cumulative statistics that cannot be diffed. Using ratios of logins to the various PGA and UGA memory allocations can show possible settings for PGA_AGGREGATE_TARGET values or sort area values. Remember that these values are incremented each time a process is created and adds to its PGA or UGA areas, and that contribution is not removed once the session logs out, so the actual totals may bear little resemblance to the real values. Use these statistics as information only.

Instance activity stats — thread activity

The thread activity statistic shows the number of (projected) redo logs switched per hour. The old saw was to have redo logs switch every 15 minutes. This rule of thumb to switch every 15 minutes may or may not apply to your database. Tune redo log size according to the needs of your system. However, excessive redo log switching (like once a minute with a 5 meg log) should be avoided because this generates lots of I/O overhead.

Tablespace I/O statistics

The next section of the AWR report deals with the tablespace I/O statistics. There are two parts to the tablespace I/O statistics: part one rolls up the I/O statistics by tablespace and part two lists the statistics by data file because each tablespace may have multiple data files associated with it.

Figure 14 shows an example of this section of the report.

```
Tablespace IO Stats                          DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> ordered by IOs (Reads + Writes) desc
Tablespace
------------------------------
                 Av      Av     Av                       Av     Buffer Av Buf
         Reads Reads/s Rd(ms) Blks/Rd       Writes Writes/s      Waits Wt(ms)
-------------- ------- ------ ------- ------------ -------- ---------- ------
DATA
       512,639     142   11.8     6.4            0        0          6  151.7
INDEXES
        32,625       9   11.3    16.7            0        0         37   83.5
TEMP
         4,024       1   17.6    30.9        4,014        1          0    0.0
SYSAUX
           571       0   29.3     1.4          698        0          0    0.0
SYSTEM
           471       0    5.3     1.8           56        0          0    0.0
UNDOTBS1
             9       0   10.0     1.0          215        0          1   10.0
USERS
             1       0   10.0     1.0            0        0          0    0.0
          -------------------------------------------------------------
File IO Stats                                DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> ordered by Tablespace, File
Tablespace               Filename
------------------------ ----------------------------------------------------
                 Av      Av     Av                       Av     Buffer Av Buf
         Reads Reads/s Rd(ms) Blks/Rd       Writes Writes/s      Waits Wt(ms)
-------------- ------- ------ ------- ------------ -------- ---------- ------
DATA                     +DATA/aultdb/datafile/data.257.660765277
       501,566     139   10.8     5.2            0        0          6  151.7
DATA                     +DATA/aultdb/datafile/data.259.660843403
        11,073       3   56.9    64.5            0        0          0    0.0
INDEXES                  +DATA/aultdb/datafile/indexes.258.660765437
        32,625       9   11.3    16.7            0        0         37   83.5
SYSAUX                   +DATA2/aultdb/datafile/sysaux.257.660755929
           571       0   29.3     1.4          698        0          0    0.0
SYSTEM                   +DATA2/aultdb/datafile/system.256.660755927
           471       0    5.3     1.8           56        0          0    0.0
TEMP                     +DATA2/aultdb/tempfile/temp.266.660756117
         1,340       0   17.5    31.0        1,338        0          0    N/A
TEMP                     +DATA2/aultdb/tempfile/temp.273.660930207
             2       0   10.0     1.0            0        0          0    N/A
TEMP                     +DATA2/aultdb/tempfile/temp.274.660989059
         1,340       0   17.7    31.0        1,338        0          0    N/A
TEMP                     +DATA2/aultdb/tempfile/temp.275.661003761
         1,340       0   17.6    31.0        1,338        0          0    N/A
UNDOTBS1                 +DATA2/aultdb/datafile/undotbs1.258.660755929
             9       0   10.0     1.0          215        0          1   10.0
USERS                    +DATA2/aultdb/datafile/users.259.660755929
             1       0   10.0     1.0            0        0          0    0.0
          -------------------------------------------------------------
```

Figure 14: Tablespace I/O statistics

The tablespace I/O section of the AWR report can run to several pages if you have many tablespaces each with multiple data files. In situations where partitions are being used and each is given their own tablespace or datafile, this is often the case. Notice that the timing for writes is not a part of the report. This is because Oracle doesn't stress write tuning because for most writes it uses delayed block cleanout and only writes the blocks back to the tablespaces when needed. However, redo writes, undo writes, and temporary tablespace writes fall outside the normal writes in that they are done immediately.

When reviewing this section of the AWR report watch for the tablespaces that are showing high numbers of reads and writes, high average read milliseconds, and high numbers of buffer waits. High values for read millisecond and buffer waits indicates I/O stress and possible memory starvation for the instance. If buffer waits are not indicated but there is still a high value for read milliseconds, then the I/O subsystem is being stressed.

For tablepaces or data files that are exhibiting I/O stress, make sure there is adequate memory. After that, consider the use of IBM FlashSystem technology.

Review the I/O reports when the top five events are I/O related. Correlate the objects accessed with the top SQL statements for physical reads to the tablespace and data file level statistics. By examining the average blocks per read you can determine if the access to the tablespace is predominately index based (the value is closer to one block per read), if the activity is predominately full table or index scans (the value is close to the db file multi block read count), or if the access is direct (the value is higher than the db file multiblock read count).

If the temporary tablespace shows high levels of I/O in spite of sorts (disk) or work area executions single pass and multipass being zero, then look at the use of hash joins in the v$sql_plan table and also look for global temporary table and bitmap usage because these may be causing the temporary tablespace activity.

Buffer pool statistics

The next section of the report deals with how the buffer pools are being used. In our example (in Figure 15) there is only one buffer pool, the default one, being shown; however, in your database there could be a keep, recycle, 2K, 4K, 8K, 16K, or 32K (or 64 bit) in addition to your default block size pool (note that you cannot use the special blocksize designation parameter for the blocksize that is the same as your default pool).

```
Buffer Pool Statistics                  DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Standard block size Pools  D: default,  K: keep,  R: recycle
-> Default Pools for other block sizes: 2k, 4k, 8k, 16k, 32k

                                                             Free Writ      Buffer
     Number of Pool         Buffer     Physical    Physical Buff Comp        Busy
P      Buffers Hit%           Gets        Reads      Writes Wait Wait       Waits
--- ---------- ---- -------------- ------------ ----------- ---- ---- ----------
D      159,244   91      6,287,434      572,581       2,143    0    0          44
          -------------------------------------------------------------

Instance Recovery Stats                 DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> B: Begin snapshot,  E: End snapshot

  Targt  Estd                                     Log File Log Ckpt     Log Ckpt
  MTTR   MTTR    Recovery  Actual     Target        Size    Timeout     Interval
   (s)    (s)    Estd IOs Redo Blks Redo Blks  Redo Blks Redo Blks   Redo Blks
- ----- ----- ---------- --------- --------- ---------- --------- ------------
B     0     0        250       974      1015      92160      1015          N/A
E     0     0        292      1192      2751      92160      2751          N/A
          -------------------------------------------------------------

Buffer Pool Advisory                    DB/Inst: AULTDB/aultdb1  Snap: 92
-> Only rows with estimated physical reads >0 are displayed
-> ordered by Block Size, Buffers For Estimate

                                    Est
                                   Phys
    Size for   Size      Buffers for   Read          Estimated
P    Est (M) Factor        Estimate Factor     Physical Reads
--- -------- ------ --------------- ------ ------------------
D        128     .1          15,536    3.4          1,988,649
D        256     .2          31,072    2.5          1,497,186
D        384     .3          46,608    2.0          1,196,087
D        512     .4          62,144    1.6            959,386
D        640     .5          77,680    1.3            787,130
D        768     .6          93,216    1.1            674,908
D        896     .7         108,752    1.0            620,121
D      1,024     .8         124,288    1.0            599,692
D      1,152     .9         139,824    1.0            593,191
D      1,280    1.0         155,360    1.0            592,402
D      1,312    1.0         159,244    1.0            592,402
D      1,408    1.1         170,896    1.0            592,356
D      1,536    1.2         186,432    1.0            591,798
D      1,664    1.3         201,968    1.0            591,798
D      1,792    1.4         217,504    1.0            591,798
D      1,920    1.5         233,040    1.0            591,798
D      2,048    1.6         248,576    1.0            591,798
D      2,176    1.7         264,112    1.0            591,798
D      2,304    1.8         279,648    1.0            591,798
D      2,432    1.9         295,184    1.0            591,798
D      2,560    2.0         310,720    1.0            591,798
          -------------------------------------------------------------
```

Figure 15: Buffer pool statistics

This report section has three parts: buffer pool statistics, instance recovery stats, and the buffer pool advisory. We will start with the buffer pool statistics.

Buffer pool statistics

The buffer pool statistics give the gross performance indicators for the buffer pool. The number of buffers, cache hit ratio, buffer gets, physical reads, physical writes, free buffer waits, write completion waits, and buffer busy waits are shown here.

Of the statistics shown, the most important are those dealing with waits. The free buffer waits statistics are a good indicator if you need more buffers in the pool where it shows a greater than zero value. The write complete waits occur if DBWR cannot keep up with the writing of dirty blocks. A block that has been put on the write list cannot be reused until it has been written. When the buffer activity is such that DBWR can't write the blocks fast enough after they are on the write list, then write complete waits are generated. If you see write complete waits try boosting the priority of DBWR and LGWR processes and adding DBWR processes.

Buffer busy waits are indicative of an overloaded buffer cache where processes aren't releasing buffers fast enough. This can happen because of interested transaction-list (ITL) waits, locks, and other issues that prevent a session from taking ownership of a block in use by another session. Sometimes increasing the number of buffers can help with buffer busy waits, but it can also be a signal of application locking issues or too large a block size. With too large a block size in later versions of the Oracle Database you may also see the wait for other processes, which is actually more descriptive of what is going on. Placing tables and indexes that are experiencing these types of waits into a tablespace with a smaller blocksize can help. Look in the sections on ITL waits later in the report to help pin down which objects are causing the problems.

Instance recovery statistics

The instance recovery statistics show how many blocks would need to be recovered if an instance crash were to occur. Essentially, any block that is not current at the time of the crash would need to be evaluated for roll forward then roll back operations. Use this section to tune the various fast start and recovery initialization parameters.

Buffer pool advisory section

The buffer pool advisory attempts to show you with numbers what would happen if you increase or decrease the number of buffers in your buffer pool. It is a good practice to graph the size factor or the buffers for evaluating against the estimated physical reads saved. An example graph from the data in Figure 16 is shown. Start with the 0.7 read factor value or higher or you may see odd results.

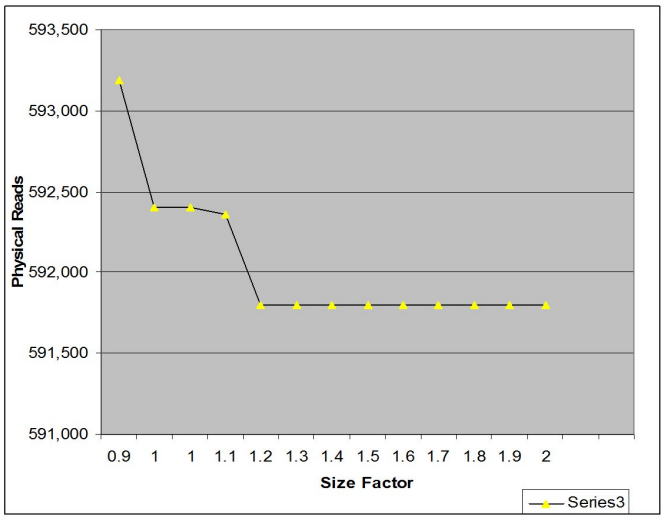


Figure 16: Example advisory plot

As you can see, we started our plot at 0.9. At the 1.2 size factor we see a drop in expected physical reads and then the physical reads are expected to plateau out to past twice our current setting. Therefore, based on the advisory, increasing our db cache size to 1.2 times the current setting should provide a benefit. There is another small decrease at 1.4 but the gain probably wouldn't be noticeable between 1.2 and 1.4.

PGA statistics

The next section of the report deals with the PGA and its settings (PGA_AGGREGATE_TARGET). You would look at tuning the PGA if you were seeing sorts to disk, workarea executions in the single or multipass statistics, or excessive I/O to the temporary tablespace. Figure 17 shows this report section.

```
PGA Aggr Summary                          DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> PGA cache hit % - percentage of W/A (WorkArea) data processed only in-memory

PGA Cache Hit %   W/A MB Processed  Extra W/A MB Read/Written
--------------- ------------------ --------------------------
           54.8              2,843                      2,345
-------------------------------------------------------------

PGA Aggr Target Stats                     DB/Inst: AULTDB/aultdb1  Snaps: 91-92

                No data exists for this section of the report.
-------------------------------------------------------------

PGA Aggr Target Histogram                 DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Optimal Executions are purely in-memory operations

  Low     High
Optimal Optimal    Total Execs  Optimal Execs 1-Pass Execs M-Pass Execs
------- ------- -------------- -------------- ------------ ------------
     2K      4K          1,833          1,833            0            0
    64K    128K              5              5            0            0
   128K    256K              1              1            0            0
   256K    512K              6              6            0            0
   512K   1024K            439            439            0            0
     1M      2M              6              6            0            0
     2M      4M              6              6            0            0
     4M      8M             14             14            0            0
     8M     16M              6              6            0            0
    16M     32M              4              4            0            0
    64M    128M              3              3            0            0
   256M    512M              6              0            6            0
-------------------------------------------------------------

PGA Memory Advisory                       DB/Inst: AULTDB/aultdb1  Snap: 92
-> When using Auto Memory Mgmt, minimally choose a pga_aggregate_target value
   where Estd PGA Overalloc Count is 0

                                          Estd Extra    Estd P Estd PGA
PGA Target    Size           W/A MB   W/A MB Read/    Cache Overallo    Estd
  Est (MB)   Factr        Processed Written to Disk   Hit %    Count    Time
---------- ------- ---------------- ---------------- ------ -------- -------
        64     0.1          3,388.1          6,390.6   35.0       22 1.6E+05
       128     0.3          3,388.1          5,795.7   37.0        2 1.5E+05
       256     0.5          3,388.1          4,885.0   41.0        1 1.3E+05
       384     0.8          3,388.1          1,172.5   74.0        0 74,015
       512     1.0          3,388.1          1,172.5   74.0        0 74,015
       614     1.2          3,388.1          1,172.5   74.0        0 74,015
       717     1.4          3,388.1          1,172.5   74.0        0 74,015
       819     1.6          3,388.1          1,172.5   74.0        0 74,015
       922     1.8          3,388.1          1,172.5   74.0        0 74,015
     1,024     2.0          3,388.1          1,172.5   74.0        0 74,015
     1,536     3.0          3,388.1          1,172.5   74.0        0 74,015
     2,048     4.0          3,388.1          1,172.5   74.0        0 74,015
     3,072     6.0          3,388.1          1,172.5   74.0        0 74,015
     4,096     8.0          3,388.1          1,172.5   74.0        0 74,015
-------------------------------------------------------------
```

Figure 17: PGA statistics

There are four parts to the PGA statistics section of the AWR reports: PGA Aggregate Summary, PGA Aggregate Target Status, PGA Aggregate Target Histogram, and PGA Memory Advisor. We will begin with the PGA Aggregate Summary.

PGA Aggregate Summary

The PGA Aggregate Summary section of the AWR report shows the rolled up usage data for the PGA area. It shows what it calls the PGA Cache Hit percent and then the components that make up that percent: the work area megabytes processed and the Extra Work Area Megabyte Read/Written. The PGA Cache Hit Percent is the total number of bytes processed in the PGA versus the total number of bytes processed plus extra bytes read/written in extra passes. In our example report we see that we only get 54 percent because we processed 2 gigabytes to disk in six single passes. If this was a typical transaction profile captured in the AWR report, it would indicate the need to add memory to our PGA_AGGREGATE_TARGET to eliminate disk-based sort. The next section we will look at is the PGA Aggregate Target statistics.

PGA Aggregate Target statistics

The example report didn't include a section on this so a section from another report has been used.

	PGA Aggr Target(M)	Auto PGA Target(M)	PGA Mem Alloc(M)	W/A PGA Used(M)	PGA W/A % Mem	%Auto W/A Mem	%Man W/A Mem	Global Mem Bound(K)
B	1,628	1,434	425.37	284.23	66.82	100.00	0.00	166,700
E	1,628	1,424	315.79	177.43	56.19	100.00	0.00	166,700

In this section of the PGA reports the differences between the start and end AWR statistics collections are shown. This section is used to determine if the PGA aggregate settings are adequate or if they should be increased. If the ending statistics show increases, then the system had to adjust the parameters on the fly (or they were manually adjusted). You need to review this section to be sure that someone didn't reduce or otherwise change the PGA profile during the test.

PGA Aggregate Target histogram

This is probably the most useful part of the PGA section of the AWR report. In this section the histogram shows you the various memory sizes that were used during the time period measured. By examining where single or multipass executions occurred you can decide what your PGA_AGGREGATE_TARGET setpoint needs to be. By remembering that a single process gets a maximum of 5percent of the PGA_AGGREGATE_TARGET setting up to the value of the undocumented parameter "_pga_max_size" and knowing that the largest assigned sort segment would be a hash segment and that by default this is two times the normal sort segment, you can simply multiply the high boundary for the sorts you want to eliminate by 40 to get the needed set point. In our case, the upper range is 512MB so 40*512MB would be 20GB. Unfortunately, in Oracle Database 11g the setting for "_pga_max_size" is 500MB so we can't totally guarantee we could eliminate all the sorts.

In Oracle Database 11g the maximum setting is 32GB for PGA_AGGREGATE_TARGET. However, that being said, you can set this parameter to more than the physical memory that is on the machine; just be sure you don't have enough processes wanting to do sorts at the same time that would cause swapping!

The final section, and the least useful, is the PGA Advisor.

PGA Memory Advisor

In theory, a tool that estimates the effect of increasing or decreasing the PGA_AGGREGATE_TARGET sounds like a good idea, but in practice this part of the PGA section of the AWR report has never worked properly. For example, looking at what it recommends: it says that we could reduce the size of the PGA_AGGREGATE_TARGET to 80 percent of what it is right now with no ill effects. Supposedly if the Estimated PGA Over Allocation count column shows values, then that setting is too low. If it doesn't and the estimated time column shows no decrease, then that is the good setting. Yet when we do the calculations to eliminate sorts to disk, we get a number much larger than what we currently have set. However, it is showing that even at the largest setting it thinks is available we would only see a 74 percent PGA cache hit percent. Trust your own calculations based on the histogram.

Shared pool statistics

The shared pool section of the AWR report is another section that is often useless. There seems to be numerous problems with the algorithms that populate the advisories. Figure 18 shows the example excerpt from our AWR report.

```
Shared Pool Advisory                          DB/Inst: AULTDB/aultdb1  Snap: 92
-> SP: Shared Pool     Est LC: Estimated Library Cache    Factr: Factor
-> Note there is often a 1:Many correlation between a single logical object
   in the Library Cache, and the physical number of memory objects associated
   with it.  Therefore comparing the number of Lib Cache objects (e.g. in
   v$librarycache), with the number of Lib Cache Memory Objects is invalid.

                                                 Est LC Est LC  Est LC Est LC
  Shared    SP   Est LC                          Time   Time    Load   Load       Est LC
    Pool  Size      Size       Est LC     Saved  Saved    Time   Time    Mem Obj
Size(M) Factr       (M)     Mem Obj       (s)  Factr     (s)  Factr   Hits (K)
-------- ----- -------- ------------ -------- ------ ------- ------ ----------
     192    .9        3          495    4,555    1.0      41    1.0         12
     224   1.0       33        4,350    4,555    1.0      41    1.0         96
     256   1.1       45        6,645    4,557    1.0      39    1.0         96
     288   1.3       45        6,645    4,558    1.0      38     .9         96
     320   1.4       45        6,645    4,558    1.0      38     .9         96
     352   1.6       45        6,645    4,558    1.0      38     .9         96
     384   1.7       45        6,645    4,558    1.0      38     .9         96
     416   1.9       45        6,645    4,558    1.0      38     .9         96
     448   2.0       45        6,645    4,558    1.0      38     .9         96
          -------------------------------------------------------------
```

Figure 18: Shared pool statistics

Other than recommending that we increase the shared pool size by 30 percent, not much else of value is being shown. However, if the header data about shared SQL shows problems, this section might help you reset some sizes. Using SGA_MAX_SIZE and SGA_TARGET in Oracle Database 10g and higher can help the Oracle Database adjust shared pool size. In Oracle Database 11g use either the previously mentioned parameters or MEMORY_MAX_SIZE and MEMORY_TARGET parameters to allow automatic shared pool size adjustment.

Other advisories

There are three other advisories contained in the AWR report. An example excerpt from the AWR report for the advisories is shown in Figure 19.

SGA Target Advisory DB/Inst: AULTDB/aultdb1 Snap: 92

SGA Target Size (M)	SGA Size Factor	Est DB Time (s)	Est Physical Reads
396	0.3	8,538	592,206
792	0.5	8,536	592,206
1,188	0.8	8,536	592,206
1,584	1.0	8,536	592,206
1,980	1.3	8,536	592,206
2,376	1.5	8,542	592,206
2,772	1.8	8,542	592,206
3,168	2.0	8,542	592,206

Streams Pool Advisory DB/Inst: AULTDB/aultdb1 Snap: 92

Size for Est (MB)	Size Factor	Est Spill Count	Est Spill Time (s)	Est Unspill Count	Est Unspill Time (s)
32	0.13	0	0	0	0
64	0.25	0	0	0	0
96	0.38	0	0	0	0
128	0.50	0	0	0	0
160	0.63	0	0	0	0
192	0.75	0	0	0	0
224	0.88	0	0	0	0
256	1.00	0	0	0	0
288	1.13	0	0	0	0
320	1.25	0	0	0	0
352	1.38	0	0	0	0
384	1.50	0	0	0	0
416	1.63	0	0	0	0
448	1.75	0	0	0	0
480	1.88	0	0	0	0
512	2.00	0	0	0	0
544	2.13	0	0	0	0
576	2.25	0	0	0	0
608	2.38	0	0	0	0
640	2.50	0	0	0	0

Java Pool Advisory DB/Inst: AULTDB/aultdb1 Snap: 92

Java Pool Size(M)	JP Size Factr	Est LC Size (M)	Est LC Mem Obj	Est LC Time Saved (s)	Est LC Time Saved Factr	Est LC Load Time (s)	Est LC Load Time Factr	Est LC Mem Obj Hits
16	1.0	2	79	7	1.0	41	1.0	88
32	2.0	4	163	7	1.0	41	1.0	182

Figure 19: AWR advisories

SGA target advisory

If you have the SGA_TARGET parameter set, the AWR report shows the SGA_TARGET advisory. The advisory shows the affects of changing the size of the SGA_TARGET parameter on your system. In the case of the example report, it is indicating that if we increased our SGA_TARGET by 50 percent we would see a marginal increase in the efficiency of the memory management; however, the increase is so small it probably isn't worth the effort unless other factors confirm the recommendation. In the case of the other sections we have looked at, there is no real reason to increase the setting.

Streams Pool Advisory

When the STREAMS_POOL_SIZE parameter is set, the Streams Pool Advisory is populated in the AWR report. Streams uses the Streams Pool to buffer the messages it sends and receives to and from other systems. If the Streams Pool size is insufficient, these messages are queued to disk ("spilled"). Excessive disk spills by the streams processes result in poor performance of the streams processes. The advisory shows the effects of increasing or decreasing the current stream pool by showing the increase or decrease in spillage and the effect on performance with projected time in seconds for either performance losses (Est Spill Time) or gains (Est Unspill Time).

Java pool advisory

There was a time when the Java pool was rarely if ever used. It was set to 16MB and ignored. Now Oracle does more and more of its work, especially with export and import and other utilities, using Java in the kernel, making the Java pool setting an important one to periodically review. The Java pool advisory shows the effects of increasing the pool size. In the example report, even if we doubled our pool size there would be no net gain (other than in objects able to be stored). If we were seeing Java pool related errors or our Java was running slow, then this report might help us determine if it was a Java pool issue.

Buffer Waits statistics

If we see that the Buffer Busy Waits event is causing issues with performance, the next section of the AWR report would be where we'd look to see the class of objects causing the problem. Figure 20 shows the Buffer Waits section of our example report.

```
Buffer Wait Statistics                    DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> ordered by wait time desc, waits desc

Class                    Waits Total Wait Time (s)  Avg Time (ms)
------------------ ----------- ------------------- --------------
data block                  43                   4             93
undo header                  1                   0             10
          -------------------------------------------------------------
```

Figure 20: Buffer Waits statistics

In our example we only see *data block* and *undo header* waits; however, there are several other types of buffer waits possible:

- File header block
- 1st level bitmap block (bmb)
- Segment header
- 2nd level bmb

Each of these buffer waits points us to a different type of issue.

The data block type of wait usually indicates an issue with block sharing between processes. Essentially the block may be too big for its own britches, meaning that it has too many rows and too many users want it at the same time. Usually this means someone was reading in the block when someone else requested it. Try reducing rows per block.

The undo header Buffer Waits may indicate we have an insufficient number of undo segments. If you are using automatic undo management, try reducing the *transactions_per_rollback_segment* parameter to bring more undo segments online.

File header block waits usually mean freelist or freelist group problems. You can find the segments causing the issues in the *Segments with ITL waits* section of the AWR report, which is after the latch sections. Try using automatic segment space management (ASSM) to relieve this type of contention.

The first and second level bmb type waits indicate issues with the bitmap blocks used in ASSM to manage freespace in tables (the bmbs take the place of traditional freelists and freelist groups). This could be caused by too large a blocksize or extreme internal "whitespace" inside tables caused by deletions.

The segment header buffer wait is usually caused by an insufficient number of freelists or freelist groups, which causes serialization of access and buffer waits as a result. If you are using ASSM and get these, switching back to manual management may help.

Enqueue statistics

Enqueues are serialization mechanisms within the Oracle Database. Enqueues are how the database prevents multiple updates from happening against the same record at the same time; they work with locking and latching to achieve this. When the *enqueue deadlocks*, *enqueue waits*, or *enqueue timeouts* point to an enqueue issue, you need to check the enqueue statistics section of the AWR report. The enqueue statistics section of our example report is in Figure 21.

```
Enqueue Activity                          DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> only enqueues with waits are shown
-> Enqueue stats gathered prior to 10g should not be compared with 10g data
-> ordered by Wait Time desc, Waits desc

Enqueue Type (Request Reason)
------------------------------------------------------------------------------
    Requests    Succ Gets Failed Gets       Waits  Wt Time (s) Av Wt Time(ms)
------------ ------------ ----------- ----------- ------------ --------------
BF-BLOOM FILTER (allocation contention)
       2,618        2,611           7          14           14         982.14
TD-KTF map table enqueue (KTF dump entries)
           9            9           0           9            0          37.78
PS-PX Process Reservation
         648          616          32         208            0            .96
CF-Controlfile Transaction
       7,661        7,660           1         118            0           1.27
TM-DML
       5,559        5,559           0          16            0           3.75
XL-ASM Extent Fault Lock (fault extent map)
          14           14           0           1            0          60.00
TT-Tablespace
       1,125        1,125           0          30            0           1.67
HW-Segment High Water Mark
          76           76           0          12            0           2.50
WF-AWR Flush
          19           19           0          11            0           1.82
TA-Instance Undo
          12           12           0           8            0           2.50
KO-Multiple Object Checkpoint (fast object checkpoint)
          81           81           0          27            0            .37
FB-Format Block
           8            8           0           8            0           1.25
JQ-Job Queue
          60           60           0           2            0           5.00
PG-Global Parameter
           4            4           0           2            0           5.00
AF-Advisor Framework (task serialization)
           7            7           0           1            0          10.00
PI-Remote PX Process Spawn Status
          18           18           0           8            0            .00
RS-Reclaimable Space (prevent aging list update)
           4            4           0           4            0            .00
DR-Distributed Recovery
           2            2           0           2            0            .00
PE-Parameter
           8            8           0           2            0            .00
JS-Job Scheduler (job run lock - synchronize)
           2            2           0           1            0            .00
UL-User-defined
           2            2           0           1            0            .00
US-Undo Segment
           1            1           0           1            0            .00
          -------------------------------------------------------------
```

Figure 21: Enqueue statistics

The enqueue statistics tell you more about what is happening under the covers in the user processes and SQL processing. In the excerpt in Figure 20, we see that the BF-BLOOM FILTER enqueue is our predominant enqueue for this time period. This BF enqueue is only present when parallel query is used and signifies that the database used a bloom filter mechanism to filter the results from one or more of the parallel query slaves during the parallel query operations. The BF type enqueue has only been available since Oracle Database 10g when the bloom filter was added to the Oracle Database's repertoire.

The biggest issue with determining what enqueues are telling us is that they aren't always well documented and may involve a web search or search of www.oracle.com or metalink.oracle.com to resolve. The V$SESSION_WAIT and V$LOCK dynamic performance views will provide more details about enqueue issues by looking at the P1 and P2 values listed and knowing the type of enqueue. To see what the various P1 and P2 values mean for a specific enqueue, the following query should be run in your instance:

```
column parameter1 format a15
column parameter2 format a15
column parameter3 format a15
column lock format a8

Select
      substr(name,1,7) as
"lock",parameter1,parameter2,parameter3 from v$event_name
where name like 'enq%'
```

The list for Oracle Database 11g has 247 entries. For the BF enqueue, the additional information we could get would be the node#, parallelizer#, and bloom#. You should concentrate on the enqueues that show a wait time, because if there is no time penalty (at least one that can be measured in whole milliseconds) associated with an enqueue, you don't really care how many times a process or group of processes waited on it.

Undo segment statistics

In the beginning there was redo and rollback; then in late Oracle Database 8i the terminology was switched to redo and undo. If you hear someone talking about rollback segments, they mean undo segments and visa versa. Most DBAs use automated undo management. Automated undo management essentially uses the process count and the *TRANSACTIONS_PER_ROLLBACK_SEGMENT* parameter to determine when segments should be brought online. Initially, Oracle brings 10 segments online then waits for the ratio of processes/*transactions_per_rollback_segment* to exceed 10 to bring on a new one, and each time the ratio increments after that a new segment is brought online. Unfortunately, most of these segments just sit there taking up space; for as we all know, a process doesn't translate directly into a new transaction.

Systems have been tracked that have a hundred undo segments, of which only five are actually being used and the rest are sitting there idle and offline just taking up space. An example of the undo statistics section from our AWR report is shown in Figure 22.

```
Undo Segment Summary                    DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Min/Max TR (mins) - Min and Max Tuned Retention (minutes)
-> STO - Snapshot Too Old count,  OOS - Out of Space count
-> Undo segment block stats:
-> uS - unexpired Stolen,   uR - unexpired Released,   uU - unexpired reUsed
-> eS - expired   Stolen,   eR - expired   Released,   eU - expired   reUsed

Undo   Num Undo       Number of  Max Qry   Max Tx Min/Max   STO/     uS/uR/uU/
 TS# Blocks (K)    Transactions  Len (s) Concurcy TR (mins) OOS      eS/eR/eU
---- ---------- --------------- -------- -------- --------- ----- --------------
   2         .1           1,090    1,725        3 18.8/42.8 0/0   0/0/0/0/0/0
          -------------------------------------------------------------

Undo Segment Stats                      DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Most recent 35 Undostat rows, ordered by Time desc

                Num Undo    Number of Max Qry  Max Tx Tun Ret STO/    uS/uR/uU/
End Time          Blocks Transactions Len (s)   Concy  (mins) OOS     eS/eR/eU
------------ ----------- ------------ ------- ------- ------- ----- ------------
04-Aug 12:56          14          167     890       3      29 0/0   0/0/0/0/0/0
04-Aug 12:46          10          141     289       3      19 0/0   0/0/0/0/0/0
04-Aug 12:36          10          163   1,725       2      43 0/0   0/0/0/0/0/0
04-Aug 12:26          18          240   1,124       3      33 0/0   0/0/0/0/0/0
04-Aug 12:16           9          133     901       2      29 0/0   0/0/0/0/0/0
04-Aug 12:06          88          246     300       3      19 0/0   0/0/0/0/0/0
          ---------------------------------------------------------------
```

Figure 22: Undo statistics

Undo statistics tell us how efficiently the undo segments are being handled. Unfortunately there is not much that can be done to tune them if we are using automatic undo management. You control three aspects of the undo segments when you use automatic management: size of the undo tablespace, placement of the undo tablespace datafiles, and the value of the parameter *transactions_per_rollback_segment*. You can use the undo advisor in Oracle Enterprise Manager, Oracle Grid Control, or Oracle Database Control to determine the suggested size of the undo tablespace to be based on historical AWR data (it defaults to seven days worth, the maximum retained by default). By tweaking the *transactions_per_rollback_segment* you can also reduce the STO numbers (if you get them). STO stands for snapshot too old, or ORA-01555. The OOS column means out of space, which is rare and usually means the database ran out of space in your tablespace or filesystem. By using the undo advisor and playing with the undo retention numbers you can derive a tablespace size to prevent both STO and OOS errors. The dba_rollback_segments view provides detailed information on the undo segments, if you want to see more information.

Latch statistics

The next area of the AWR report is the latch statistics area. Like the Instance Activity statistics, there is a lot of information contained in this section; unfortunately, most of it is not useful for you in your tuning efforts and should be filtered out of the result set. However, the Oracle Database must use some of it for internal tuning efforts, so we are stuck with using our own filters to remove the non-essential data.

A reduced version of the full latch section (filtered to show the latches of concern in this environment) is shown in Figure 23.

```
Latch Activity                          DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> "Get Requests", "Pct Get Miss" and "Avg Slps/Miss" are statistics for
   willing-to-wait latch get requests
-> "NoWait Requests", "Pct NoWait Miss" are for no-wait latch get requests
-> "Pct Misses" for both should be very close to 0.0

                                    Pct    Avg   Wait                 Pct
                               Get  Get    Slps  Time       NoWait NoWait
Latch Name                Requests  Miss  /Miss   (s)     Requests   Miss
------------------------ ---------- ------ ------ ------ ---------- ------
KJC message pool free li       14,582    0.3    0.0      0     15,463    0.1
gc element                2,414,376    0.0    0.3      2      7,880    0.0
gcs resource hash         1,861,484    0.0    0.5      1          6    0.0
virtual circuit queues            1    0.0             0          0    N/A
          -------------------------------------------------------------

Latch Sleep Breakdown                   DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> ordered by misses desc

                                 Get                                    Spin
Latch Name                    Requests       Misses      Sleeps         Gets
------------------------ ------------ ------------ ----------- -----------
cache buffers chains        8,492,860       21,037           3      21,034
simulator lru latch         1,823,879       12,065         311      11,774
cache buffers lru chain     1,190,948        6,096         352       5,799
gc element                  2,414,376          767         213         582
KJCT flow control latch       443,643          735          11         725
          -------------------------------------------------------------

Latch Miss Sources                      DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> only latches with sleeps are shown
-> ordered by name, sleeps desc

                                                 NoWait              Waiter
Latch Name               Where                   Misses     Sleeps   Sleeps
------------------------ ----------------------- ------- ---------- --------
cache buffers lru chain  kcbzgws_1                     0        248      272
gc element               kclnfndnewm                   0        112       18
gcs resource hash        kjbassume                     0         88        0
          -------------------------------------------------------------
Mutex Sleep Summary                     DB/Inst: AULTDB/aultdb1  Snaps: 91-92

          No data exists for this section of the report.
          -------------------------------------------------------------
Parent Latch Statistics                 DB/Inst: AULTDB/aultdb1  Snaps: 91-92

          No data exists for this section of the report.
          -------------------------------------------------------------

Child Latch Statistics                  DB/Inst: AULTDB/aultdb1  Snaps: 91-92

          No data exists for this section of the report.
          -------------------------------------------------------------
```

Figure 23: Latch section of AWR report

The proper use of latches within the Oracle Database environment can determine whether or not your application can scale. That being said, if the latch related events such as *latch free* is not a top five wait event then latching is not an issue for your database at this time and your efforts should be spent elsewhere. Analysis of the latch section of the report is critical if scalability has become a concern in your database. The latch section of the AWR report has six sections:

- *Latch Activity*—Overall latch statistics
- *Latch Sleep Breakdown*—List of latches with sleeps
- *Latch Miss Sources*—List of latches that were the source of most misses
- *Mutex Sleep Summary*—In Oracle Database 10.2.0.1 some latches were switched to lighter weight mutexes; this section shows the waits related to these objects.
- *Parent Latch Statistics*—This section shows parent latches
- *Child Latch Statistics*—This section shows the latches that are children latches to parent latches in the previous section

Some of these sections are useful; it should be noted however that they contain lots of extraneous information. As was stated above, this example figure was paired down to just those that showed statistics of concern; the actual section was two to three pages long!

Latch activity

In the latch activity section there are two percentage statistics: *Pct Get Misses* and *Pct NoWait Misses*. The two percent statistics show the latches you should be concerned with right now. If either or both show a value other than N/A or 0.0 then that latch is showing some stress. Now, should you be concerned with fractional percentages? Probably not. As you can see in the section of the report shown above, the highest percentage in the example report is 0.3 percent for *Pct Get Misses* for the *KJC message pool free li* latch (actually the full name is *KJC message pool free list*) which is an Oracle RAC related latch dealing with the Kernel Job Control message pool (Global Enqueue Services). Because the percentage is less than 1 percent it is of no real concern. Probably the best source of information about these latches is Oracle Metalink, but it may take some digging.

It should be noted that many experts believe that the use of miss percentages in tuning latches may lead you to tune latches that are really not important. Instead, you should be looking at the next section of the report on latch sleeps.

Latch sleep breakdown

When a process tries to get a latch, if there is no latch available the process spins on the CPU, waiting for the latch to become available. This is called a sleep and latches with high sleeps may be having contention issues. Generally speaking, look at the latch with the highest percentage of sleeps as related to total sleeps (sleeps/sum(sleeps)*100 from v$latch) if you have a high *latch free* situation because this will be the latch most affecting performance.

The best way to tune latching issues is to ensure the database has a proper level of resources (generally memory in the cache and shared pool areas); be sure to use bind variables, and eliminate hot block issues.

Many experts point to the value of the undocumented parameter "*_spin_count*" as a possible source for latch spin issues. The spin count tells the Oracle Database how many CPU cycles to wait for a latch before incrementing spin count. This may be set as low as 2000 for many systems. With higher speed CPUs this value is too low and can result in high latch sleeps when there really isn't a problem. The argument for adjusting this undocumented parameter is that the value is really dependent upon the CPU speed, and the value of 2000 for the default was based on CPU speeds available when *Oracle Database version 7.0* was released! Obviously we have seen a huge increase in CPU speeds since then with no increase in the setting of "*_spin_count*". However, spin should only be adjusted if the value of your runqueue is: a.) due to CPU and not I/O, b.) less than the number of CPUs (assuming you are on Linux, Unix, AIX, or HP-UX). Runqueue tells how many processes are waiting to execute and can be incremented by processes waiting on either I/O or CPU. A clue to if the runqueue is I/O or CPU based is if *runqueue>#CPUs* and CPUs show idle time while I/O wait is high, then the runqueue is due to I/O wait. If *runqueue>#CPUs* and CPU utilization is near 100 percent with low or no I/O wait, then it is due to CPU.

In situations where there is a high runqueue, you can sometimes correct this if it is CPU or I/O related with an Oracle Database system by renicing the LGWR, DBWR, and if using Oracle RAC, the LMON processes. Renicing means to increase those processes' priority.

So, by increasing the "*_spin_count*" parameter, sometimes to as high as 10,000 or more, improvements in throughput and reductions in overall wait times have been seen; however, this will need to be tested for a proper setting in your environment.

Latch miss sources

The latch miss sources section shows which latches are missed during an attempted get operation. In this section the important consideration is once again the latch with the highest level of sleeps. See the above section on Latch Sleeps for tuning suggestions.

Mutex sleep summary

As with latches, mutexes, which replace some latches in Oracle Database 10.2.0.1 and above releases, will sleep and spin. The mutex with the highest level of sleeps should be investigated for tuning.

Parent and child latches

The sections on parent and child latches are used to help isolate which latches and their children are causing issues. By determining which latch or child latch is sleeping or waiting the most, you can determine whether the problem is related to memory, bind variables, or hot blocks.

Segment access areas

The next several sections are similar to the SQL sections, except they deal with segments. They are similar to the SQL sections because they slice and dice the various segment access issues and show which segments are exhibiting specific forms of contention. Figure 24 shows these sections of the report.

```
Segments by Logical Reads                DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Total Logical Reads:      22,791,070
-> Captured Segments account for    41.8% of Total

           Tablespace                      Subobject  Obj.       Logical
Owner         Name    Object Name            Name     Type         Reads  %Total
---------- ---------- -------------------- ---------- ----- ------------ -------
TPCH       INDEXES    H_ORDERS_IDX1                   INDEX    4,294,720   18.84
TPCH       DATA       H_LINEITEM                      TABLE    2,117,568    9.29
TPCH       DATA       H_ORDER                         TABLE    1,017,136    4.46
TPCH       INDEXES    SUPPLIER_IDX1                   INDEX      626,848    2.75
TPCH       DATA       H_SUPPLIER                      TABLE      620,432    2.72
          -------------------------------------------------------------

Segments by Physical Reads               DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Total Physical Reads:       9,773,380
-> Captured Segments account for    39.3% of Total

           Tablespace                      Subobject  Obj.      Physical
Owner         Name    Object Name            Name     Type         Reads  %Total
---------- ---------- -------------------- ---------- ----- ------------ -------
TPCH       DATA       H_LINEITEM                      TABLE    2,107,980   21.57
TPCH       DATA       H_ORDER                         TABLE      894,131    9.15
** UNAVAIL ** UNAVAIL ** UNAVAILABLE **    AILABLE ** UNDEF       511,994    5.24
TPCH       DATA       H_PART                          TABLE      123,676    1.27
TPCH       DATA       H_PARTSUPP                      TABLE      117,400    1.20
          -------------------------------------------------------------

Segments by Row Lock Waits               DB/Inst: AULTDB/aultdb1  Snaps: 91-92

              No data exists for this section of the report.
          -------------------------------------------------------------

Segments by ITL Waits                    DB/Inst: AULTDB/aultdb1  Snaps: 91-92

              No data exists for this section of the report.
          -------------------------------------------------------------

Segments by Buffer Busy Waits            DB/Inst: AULTDB/aultdb1  Snaps: 91-92

              No data exists for this section of the report.
          -------------------------------------------------------------
```

```
Segments by Global Cache Buffer Busy     DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> % of Capture shows % of GC Buffer Busy for each top segment compared
-> with GC Buffer Busy for all segments captured by the Snapshot

                                                                      GC
           Tablespace                      Subobject  Obj.       Buffer    % of
Owner         Name    Object Name            Name     Type         Busy Capture
---------- ---------- -------------------- ---------- ----- ------------ -------
** UNAVAIL ** UNAVAIL ** UNAVAILABLE **    AILABLE ** UNDEF            9   81.82
TPCH       INDEXES    H_ORDERS_IDX1                   INDEX            1    9.09
TPCH       INDEXES    PARTSUPP_IDX1                   INDEX            1    9.09
          -------------------------------------------------------------

Segments by CR Blocks Received           DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Total CR Blocks Received:             361
-> Captured Segments account for        35.5% of Total

                                                                     CR
           Tablespace                      Subobject  Obj.       Blocks
Owner         Name    Object Name            Name     Type     Received  %Total
---------- ---------- -------------------- ---------- ----- ------------ -------
SYS        SYSTEM     JOB$                            TABLE          22    6.09
SYS        SYSAUX     SMON_SCN_TIME                   TABLE          21    5.82
SYSMAN     SYSAUX     MGMT_SYSTEM_PERFORMA            TABLE          12    3.32
SYSMAN     SYSAUX     MGMT_SYSTEM_PERF_LOG            INDEX          12    3.32
SYSMAN     SYSAUX     MGMT_TASK_QTABLE                TABLE          12    3.32
          -------------------------------------------------------------

Segments by Current Blocks Received      DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Total Current Blocks Received:          95,445
-> Captured Segments account for            99.9% of Total

                                                                   Current
           Tablespace                      Subobject  Obj.       Blocks
Owner         Name    Object Name            Name     Type     Received  %Total
---------- ---------- -------------------- ---------- ----- ------------ -------
TPCH       INDEXES    H_ORDERS_IDX1                   INDEX       65,524   68.65
TPCH       DATA       H_ORDER                         TABLE       24,149   25.30
TPCH       DATA       H_SUPPLIER                      TABLE        2,232    2.34
SYS        SYSTEM     TAB$                            TABLE          996    1.04
** UNAVAIL ** UNAVAIL ** UNAVAILABLE **    AILABLE ** UNDEF          776     .81
          -------------------------------------------------------------
```

Figure 24: Segments sections

The segments sections of the AWR reports allow you to pinpoint which segments and tablespaces are responsible for the various types or reads, waits, and other database related statistics that are related to segments. The segment sections are:

- *Segments by logical reads*—If you have issues with high logical reads, review these objects for possible partitioning, SQL issues, or index issues.
- *Segments by physical reads*—If you have physical read issues, review these objects for index issues or possible partitioning.
- *Segments by row lock waits*—If you have high transaction-type enqueues, look here for locking issues with objects. These will also show up in the Oracle RAC sections if the problem is present in a cluster database.
- *Segments by ITL waits*—If you have large numbers of block waits, look here for the causes.
- *Segments by buffer busy waits*—These segments probably have too large a block size if in Oracle RAC. Otherwise look at insufficient memory allocations. These segments are experiencing hot blocks.
- *Segments by global cache buffer busy*—Usually due to too large a block size in Oracle RAC. May be helped if the segments are indexes by using reverse key (however this inhibits index scans). Look to these segments for hot block issues in Oracle RAC.
- *Segments by CR blocks received*—For consistent read issues, look at hot blocking and block size, reduce rows per block.
- *Segments by current blocks received*—Current blocks being transferred means high levels of transactions. Small blocksizes can help, as can reverse key indexers.

Library cache activity sections

In Oracle Database version 7 the tuning of the various dictionary caches was automated. However, the statistics in the next sections dealing with library cache activity show you, for example, if you need to use more caching for sequences or if you should look at using automated segment management and other internal issues that are indicated through the library cache statistics. Figure 25 shows an excerpt from our AWR report.

Dictionary Cache Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> "Pct Misses" should be very low (< 2% in most cases)
-> "Final Usage" is the number of cache entries being used

Cache	Get Requests	Pct Miss	Scan Reqs	Pct Miss	Mod Reqs	Final Usage
dc_awr_control	63	3.2	0	N/A	0	1
dc_database_links	2	0.0	0	N/A	0	1
dc_files	8	0.0	0	N/A	0	8
dc_global_oids	2,826	0.2	0	N/A	0	133
dc_histogram_data	1,151	11.6	0	N/A	0	750
dc_histogram_defs	3,213	5.6	0	N/A	0	3,460
dc_object_grants	484	0.0	0	N/A	0	17
dc_objects	7,172	1.0	0	N/A	17	2,203
dc_profiles	64	0.0	0	N/A	0	1
dc_rollback_segments	850	0.0	0	N/A	0	22
dc_segments	1,020	5.9	0	N/A	4	728
dc_sequences	13	30.8	0	N/A	13	0
dc_tablespaces	9,757	0.0	0	N/A	0	9
dc_users	13,294	0.0	0	N/A	0	142
global database name	4,485	0.0	0	N/A	0	1
outstanding_alerts	52	69.2	0	N/A	2	1

Dictionary Cache Stats (RAC) DB/Inst: AULTDB/aultdb1 Snaps: 91-92

Cache	GES Requests	GES Conflicts	GES Releases
dc_awr_control	2	2	0
dc_global_oids	5	0	0
dc_histogram_defs	181	0	0
dc_objects	71	0	0
dc_segments	68	5	0
dc_sequences	26	5	0
dc_tablespaces	1	0	0
dc_users	5	0	0
outstanding_alerts	100	36	0

Library Cache Activity DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> "Pct Misses" should be very low

Namespace	Get Requests	Pct Miss	Pin Requests	Pct Miss	Reloads	Invali-dations
BODY	1,514	0.0	1,858	0.2	4	0
CLUSTER	44	0.0	16	0.0	0	0
INDEX	2	0.0	2	0.0	0	0
JAVA DATA	2	0.0	0	N/A	0	0
SQL AREA	2,246	1.5	17,091	2.5	121	6
TABLE/PROCEDURE	12,745	0.1	16,155	1.4	166	0
TRIGGER	376	0.0	423	0.0	0	0

Library Cache Activity (RAC) DB/Inst: AULTDB/aultdb1 Snaps: 91-92

Namespace	GES Lock Requests	GES Pin Requests	GES Pin Releases	GES Inval Requests	GES Invali-dations
CLUSTER	16	16	0	0	0
INDEX	2	2	0	0	0
TABLE/PROCEDURE	4,553	15,492	0	0	0

Figure 25: Library cache statistics

One thing to remember when dealing with any statistics, you need to have a statistically relevant number of measurements before the statistics are valid. If you only have two occurrences then it is rather hard to draw valid conclusions. On the other hand, if you have 1,000 or 10,000 occurrences then the statistics are more valid. If we eliminate the "invalid" statistics from the above sections, we are left with Figure 26.

```
Dictionary Cache Stats                  DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> "Pct Misses"  should be very low (< 2% in most cases)
-> "Final Usage" is the number of cache entries being used

                                    Get    Pct    Scan   Pct      Mod      Final
Cache                          Requests   Miss    Reqs  Miss     Reqs      Usage
------------------------- ------------ ------ ------- ----- -------- ----------
dc_global_oids                    2,826    0.2       0   N/A        0        133
dc_histogram_data                 1,151   11.6       0   N/A        0        750
dc_histogram_defs                 3,213    5.6       0   N/A        0      3,460
dc_objects                        7,172    1.0       0   N/A       17      2,203
dc_segments                       1,020    5.9       0   N/A        4        728
dc_tablespaces                    9,757    0.0       0   N/A        0          9
dc_users                         13,294    0.0       0   N/A        0        142
global database name              4,485    0.0       0   N/A        0          1
          -------------------------------------------------------------

Dictionary Cache Stats (RAC)            DB/Inst: AULTDB/aultdb1  Snaps: 91-92

                                   GES          GES          GES
Cache                          Requests    Conflicts     Releases
------------------------- ------------ ------------ ------------
          -------------------------------------------------------------

Library Cache Activity                  DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> "Pct Misses"  should be very low

                         Get    Pct            Pin    Pct             Invali-
Namespace           Requests   Miss       Requests   Miss    Reloads  dations
--------------- ------------ ------ -------------- ------ ---------- --------
BODY                   1,514    0.0          1,858    0.2          4        0
SQL AREA               2,246    1.5         17,091    2.5        121        6
TABLE/PROCEDURE       12,745    0.1         16,155    1.4        166        0
          -------------------------------------------------------------

Library Cache Activity (RAC)            DB/Inst: AULTDB/aultdb1  Snaps: 91-92

                    GES Lock      GES Pin      GES Pin   GES Inval GES Invali-
Namespace           Requests     Requests     Releases    Requests     dations
--------------- ------------ ------------ ------------ ---------- -----------
TABLE/PROCEDURE        4,553       15,492            0          0           0
          -------------------------------------------------------------
```

Figure 26: Valid library cache statistics

So, what should we be looking for in the valid library cache statistics? Generally speaking, miss percentages should be low; however, that being said, if the database is just starting up then the miss percentages will be high until the cache fills. In cases where many temporary segments, undo segments, and other objects are accessed and released, we may see high miss percentages but they really aren't of major concern. If we see the miss percentages for a majority of areas, then we probably have a shared pool sizing issue and we may need to (if we are using automated memory management) manually establish a "floor" value by setting the *shared_pool_size* parameter. If we can't set the shared pool parameter, we would need to increase *sga_target* to closer to *sga_max_size* in Oracle Database 10*g* or *memory_target* closer to *memory_max_target*. If you find that these pairs of parameters are set to the same value, you will need to raise the max size parameters first. In most cases you should have a several percentage point difference in the values for the target and max size parameters, usually anywhere from 200MB to 1GB depending on the amount of memory your system can give to the Oracle Database.

Based on the value of the target parameters, Oracle Database will automatically allocate space to the various automatically controlled areas such as the shared pool, default db block cache, large pool, java pool, and others. Within the shared pool the library and dictionary caches will be set. If the dictionary cache areas are too small, then misses will result. If the SQL or PL/SQL areas are too small then you will see reloads and invalidations. This can also drive *latch free* events.

Dynamic memory components sections

If you are using the automatic memory management in Oracle Database 10g or 11g then the AWR report will contain sections showing what components were resized or modified during the AWR report period. An excerpt of this section is shown in Figure 27.

```
Memory Dynamic Components               DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Min/Max sizes since instance startup
-> Oper Types/Modes: INItializing,GROw,SHRink,STAtic/IMMediate,DEFerred
-> ordered by Component

                Begin Snap     Current         Min         Max    Oper Last Op
Component         Size (Mb)   Size (Mb)   Size (Mb)   Size (Mb)   Count Typ/Mod
--------------- ----------- ----------- ----------- ----------- ------ -------
ASM Buffer Cach          .00         .00         .00         .00      0 STA/
DEFAULT 16K buf          .00         .00         .00         .00      0 STA/
DEFAULT 2K buff          .00         .00         .00         .00      0 STA/
DEFAULT 32K buf          .00         .00         .00         .00      0 STA/
DEFAULT 4K buff          .00         .00         .00         .00      0 STA/
DEFAULT 8K buff          .00         .00         .00         .00      0 STA/
DEFAULT buffer     1,312.00    1,312.00    1,296.00    1,328.00      2 GRO/DEF
KEEP buffer cac          .00         .00         .00         .00      0 STA/
PGA Target            512.00      512.00      512.00      512.00      0 STA/
RECYCLE buffer           .00         .00         .00         .00      0 STA/
SGA Target         1,584.00    1,584.00    1,584.00    1,584.00      0 STA/
Shared IO Pool           .00         .00         .00         .00      0 STA/
java pool              16.00       16.00       16.00       16.00      0 STA/
large pool             16.00       16.00       16.00       16.00      0 STA/
shared pool           224.00      224.00      208.00      240.00      2 SHR/DEF
streams pool             .00         .00         .00         .00      0 STA/
          -------------------------------------------------------------

Memory Resize Operations Summary        DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Resizes, Grows, Shrinks - Operations captured by AWR
   if there are operations on the same component for the same
   operation_type, target_size, and with the same start_time
   only one operation is captured
-> ordered by Component

                        Min         Max         Avg    Re-
Component         Size (Mb)   Size (Mb)   Size (Mb)  Sizes  Grows Shrink
--------------- ----------- ----------- ----------- ------ ------ ------
DEFAULT buffer     1,296.00    1,312.00    1,304.00      2      1      1
shared pool          224.00      240.00      232.00      2      1      1
          -------------------------------------------------------------

Memory Resize Ops                       DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Oper Types/Modes: INItializing,GROw,SHRink,STAtic/IMMediate,DEFerred
   Delta       : change in size of the component

   Target Delta: displayed only if final size <> target_size
-> Status: COMplete/CANcelled/INActive/PENding/ERRor
-> ordered by start_time desc,component

                Ela                Oper       Init        Target  Final
Start           (s) Component   Typ/Mod Size (M  Delta  Delta     (M) Sta
--------------- --- ----------- ------- ------- ------ ------ ------ ---
08/04 12:39:06    0 bufcache    GRO/DEF   1,296     16    N/A  1,312 COM
08/04 12:39:06    0 shared      SHR/DEF     240    -16    N/A    224 COM
08/04 12:02:59    2 bufcache    SHR/DEF   1,312    -16    N/A  1,296 COM
08/04 12:02:59    2 shared      GRO/DEF     224     16    N/A    240 COM
          -------------------------------------------------------------
```

Figure 27: Dynamic memory sections

The dynamic memory sections are used as a guide to setting the dynamic memory parameters *sga_target* and *sga_max_size* in Oracle Database 10*g* and *memory_target* and *memory_max_targetI* in Oracle Database 11*g*.

If you see large numbers of resize operations in your statistics in this section, pay attention to which component is doing which action. You can use this information to determine if the manual parameters (shared_pool_size, db_cache_size, java_pool_size, large_pool_size, streams_pool_size) should be set to a floor value. For example, if we were to see in the Memory Resize Operations Summary section that the shared pool was seeing a number of grow operations while the default cache was shrinking, we might want to set the parameters for their sizes to the values indicated in the Memory Resize Ops section. If, when we look at these sections we see that all of the actions are deferred and not completed, that usually indicates that the target and max size parameters are set to the same value. Many defers or shrinks followed by grows can also indicate that we need to increase the max size parameters because it shows that we are robbing Peter to pay Paul within our memory structures.

If we have the proper resources, and the target and max size parameters are set correctly, we should see marginal grows in the various pools and few shrinks. However, this assumes that we have all the memory we need to satisfy all memory needs; in systems that are memory poor there may be no way to prevent this give and take of memory and in fact that is by design.

If you have issues with cache buffer or library related latches and mutexes, this section of the report will show you possible reasons.

Process memory sections

The process global area is usually controlled by the *pga_aggregate_target* parameter since Oracle Database 9i. The individual process is usually constrained to 5 percent of the setting of *pga_aggregate_target* up to the value of the undocumented parameter -"*_pga_max_size*."

In Oracle Database 9i the size of "*_pga_max_size*" was constrained to 200MB; in Oracle Database 10g depending on release this was upped to 500MB, and in Oracle Database 11g it seems to be (at least on Windows Vista and Redhat 32 bit Linux in 11.0.1.0.6) back to 200MB. On 64 bit implementations this may be different.

Let's look at the AWR report sections dealing with Process Memory statistics. Figure 28 shows the example report sections for the Process statistics.

```
 Process Memory Summary                  DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> B: Begin snap   E: End snap
-> All rows below contain absolute values (i.e. not diffed over the interval)
-> Max Alloc is Maximum PGA Allocation size at snapshot time
-> Hist Max Alloc is the Historical Max Allocation for still-connected processes
-> ordered by Begin/End snapshot, Alloc (MB) desc

                                                          Hist
                                        Avg  Std Dev     Max     Max
                Alloc      Used       Alloc    Alloc   Alloc   Alloc    Num    Num
  Category       (MB)       (MB)       (MB)     (MB)    (MB)    (MB)   Proc  Alloc
- -------- --------- --------- -------- -------- ------- ------- ------ ------
B Other        153.1       N/A      3.5      4.9      24      24     44     44
  Freeable      13.3        .0       .9       .9       4     N/A     14     14
  JAVA           1.6        1.6      .8      1.2       2       2      2      2
  SQL            1.2        .5       .1       .1       0       3     20     13
  PL/SQL          .2        .1       .0       .0       0       0     42     42
E Other        175.3       N/A      3.5      4.6      24      24     50     50
  Freeable     101.5        .0      5.3     10.1      28     N/A     19     19
  SQL           23.2       22.5      .9      1.7       5     166     26     19
  JAVA           1.6        1.6      .8      1.2       2       2      2      2
  PL/SQL          .2        .1       .0       .0       0       0     48     48
          -------------------------------------------------------------

SGA Memory Summary                      DB/Inst: AULTDB/aultdb1  Snaps: 91-92

                                                  End Size (Bytes)
SGA regions                  Begin Size (Bytes)     (if different)
---------------------------- ------------------ --------------------
Database Buffers                1,375,731,712
Fixed Size                          1,300,968
Redo Buffers                       10,858,496
Variable Size                     671,090,200
                             ------------------
sum                             2,058,981,376
          -------------------------------------------------------------

SGA breakdown difference                DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> ordered by Pool, Name
-> N/A value for Begin MB or End MB indicates the size of that Pool/Name was
   insignificant, or zero in that snapshot

Pool   Name                            Begin MB         End MB  % Diff
------ ------------------------------ -------------- -------------- -------
java   free memory                             7.7            7.7    0.00
java   joxlod exec hp                          8.1            8.1    0.00
java   joxs heap                               .3             .3    0.00
large  ASM map operations hashta               .2             .2    0.00
large  PX msg pool                             .3             .4   41.67
large  free memory                            15.5           15.4   -0.79
shared ASH buffers                             4.0            4.0    0.00
shared CCursor                                 3.4            4.0   15.27
shared Heap0: KGL                              4.0            4.1    1.74
shared KCB Table Scan Buffer                   4.0            4.0    0.00
shared KGL buckets                             3.0            3.0    0.00
shared KGL handle                              6.6            6.6    0.26
shared KGLS heap                               3.8            4.6   20.93
shared KQR L PO                                N/A            2.3     N/A
shared KQR M PO                                2.8            3.0    5.87
shared KSFD SGA I/O b                          4.0            4.0    0.00
shared PCursor                                 2.6            2.6    1.61
shared PL/SQL DIANA                           11.4           11.4    0.00
shared PL/SQL MPCODE                          12.8           13.0    1.53
shared db_block_hash_buckets                   5.9            5.9    0.00
shared dbwriter coalesce buffer                4.0            4.0    0.00
shared free memory                            41.4           32.7  -20.92
shared gcs resources                          20.2           20.2    0.00
shared gcs shadows                            15.7           15.7    0.00
shared ges big msg buffers                     4.1            4.1    0.00
shared qesblFilter_seg                         N/A            4.0     N/A
shared row cache                               3.6            3.6    0.00
shared sql area                               14.1           16.6   17.03
shared type object de                          2.7            2.7    0.02
       buffer_cache                        1,312.0        1,312.0    0.00
       fixed_sga                             1.2            1.2    0.00
       log_buffer                           10.4           10.4    0.00
         -------------------------------------------------------------
```

Figure 28: Process memory sections

Process memory summary

The process memory summary section of the AWR report expands on the previous PGA sections that showed the use of the PGA for sort type operations. In these sections we see the before and after snapshots of how the PGA areas changed from the beginning to the end AWR snapshot. By comparing the *begin* to the *end* snapshot for the different PGA sections we can determine what might need to be tuned in our PGA environment. One thing to remember here is that these numbers are totals and not averaged over the total number of processes except where explicitly stated that it is an average, sum, or maximum value.

The B (begin) and E (end) sections allow us to compare how the usage changed over the course of the snapshot period. For example, in our example the *freeable* memory jumped from 13.3 to 101.5MB, or an average of 1.8MB per process. Generally speaking, *freeable* memory is memory that was used for sort or hash activities. We also saw increases in *other* and *SQL* memory usages.

SGA Memory summary

The SGA Memory section just shows any gross changes to the main areas of the SGA. The results would be the same if you did a "show SGA" command in SQL, plus before and after and noted changes to the numbers shown. Generally, it is not of much use in tuning.

SGA Breakdown Difference

The SGA Breakdown Difference section shows us for the memory components whose allocations may have changed, how much they changed by percentage, and in what direction. This section allows us to analyze the component by component changes and also helps pinpoint possible memory leaks. One set of the statistics shown in this section that bear watching are the "free" statistics. If the "free" statistics don't change then memory may be over-allocated to those components.

Streams component sections

If you are utilizing Oracle Database streams then the AWR will populate the various streams sections of the AWR report. Essentially, if you are seeing spills to disks from the streams pool or excessive time delays in the queues (usually you will see both if you see one or the other) then you need to look at increasing the streams pool size (it should be handled automatically) or increasing the number of queue processes for either the capture queues or apply queues. If you use resource limits, the resource limit section will show how the resource limits have been applied during this period. Figure 29 shows the streams component areas. Unfortunately, because our example database is not using streams, they are not populated with actual data from the example database but instead with data derived from other actual reports.

Streams CPU/IO Usage DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> Streams processes ordered by CPU usage
-> CPU and I/O Time in micro seconds

Session Type	CPU Time	User I/O Time	Sys I/O Time
STREAMS Capture	2,128,000	0	0
STREAMS Apply Reader	609,945	2,050	1,341
Logminer Builder	185,835	0	0
Logminer Preparer	175,559	0	0
QMON Slaves	132,888	0	0
Logminer Reader	97,009	0	887,687
STREAMS Apply Server	35,580	0	0
STREAMS Apply Coordinator	747	0	0
QMON Coordinator	454	0	0
Propagation Sender	0	0	0

Streams Capture DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> Lag Change should be small or negative (in seconds)

Capture Name	Captured Per Second	Enqueued Per Second	Lag Change	Pct RuleEval Time	Pct Enqueue Time	Pct RedoWait Time	Pct Pause Time
STREAM_CAP	65	39	93	0	23	0	71

Streams Apply /Inst: AULTDB/aultdb1 Snaps: 91-92
-> Pct DB is the percentage of all DB transactions that this apply handled
-> WDEP is the wait for dependency
-> WCMT is the wait for commit
-> RBK is rollbacks
-> MPS is messages per second
-> TPM is time per message in milli-seconds
-> Lag Change should be small or negative (in seconds)

Apply Name	Applied TPS	Pct DB	Pct WDEP	Pct WCMT	Pct RBK	Applied MPS	Dequeue TPM	Apply TPM	Lag Change
STREAM_APP	0	0	0	0	0	0	0	0	0

Buffered Queues /Inst: AULTDB/aultdb1 Snaps: 91-92
-> The Spill Rate should be very close to zero
-> The Diff in Percentage Spilled should be very close to zero or negative

Queue Schema and Name	Incoming per second	Outgoing per second	Spilled per second	Diff Pct Spilled
STRMADMIN.REP_CAPTURE	39	39	0	0
STRMADMIN.REP_DEST_QU	0	0	0	0

Buffered Subscribers /Inst: AULTDB/aultdb1 Snaps: 91-92
-> All Subscribers should have a zero spill rate

Subscriber Name	Incoming per second	Outgoing per second	Spilled per second
STREAM_APP	793	793	6
STREAM_APP	0	0	0
PROXY: "STRMADMIN".""	391	391	0
PROXY: CATLIBRAR.ASU	0	0	0
PROXY: CATLIBRAR.ASU	-793	-793	-6

Rule Set /Inst: AULTDB/aultdb1 Snaps: 91-92
-> Rule Sets ordered by Evaluations

Ruleset Name	Evals	Fast Evals	SQL Execs	CPU Time	Elapsed Time
STRMADMIN.RULESET$_10	3,174	0	3,174	1,565	2,784
SYS.ALERT_QUE_R	2	0	0	0	0
STRMADMIN.RULESET$_6	0	0	0	0	0
STRMADMIN.RULESET$_3	0	0	0	0	0
STRMADMIN.RULESET$_8	0	0	0	0	0

Resource Limit Stats Inst: AULTDB/aultdb1 Snaps: 91-92
-> only rows with Current or Maximum Utilization > 80% of Limit are shown
-> ordered by resource name

Resource Name	Current Utilization	Maximum Utilization	Initial Allocation	Limit
sessions	51	65	150	150

Figure 29: Streams AWR report sections

The statistics dealing with timing and spillage to disk are probably the most important in the sections above. If any single queue is showing excessive times then adding additional queues may help with that problem; however, the addition of queues may also require changes in the receiving or sending instances of advanced queuing setup as well. In addition, there may be spillage numbers shown. If you are experiencing spillage this means you have insufficient memory allocated to the streams pool and you should increase the size of that memory component. A section we skipped over before, the Streams Pool Advisory section, would help you in deciding the needed changes to the streams pool memory allocation.

Initialization parameter changes

Unless either some automated process or the DBA has made changes during the test period for which the AWR report has been run, there should be no changes to initialization parameter settings. Therefore, if you see changes and you did not initiate them, you need to investigate the causes of the changes. Figure 30 shows the initialization parameter section. Only parameters that have values different from the default or that have changed will be shown in the initialization parameter section.

```
init.ora Parameters                    DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> if IP/Public/Source at End snap is different a '*' is displayed

                                                                  End value
Parameter Name                 Begin value                       (if different)
------------------------------ --------------------------------- --------------
audit_file_dest                /home/oracle/app/product/oracle/a
audit_trail                    DB
cluster_database               TRUE
cluster_database_instances     2
compatible                     11.1.0.0.0
control_files                  +DATA2/aultdb/controlfile/current
db_block_size                  8192
db_create_file_dest            +DATA2
db_domain
db_name                        aultdb
db_recovery_file_dest          +DATA2
db_recovery_file_dest_size     2147483648
diagnostic_dest                /home/oracle/app/product/oracle
dispatchers                    (PROTOCOL=TCP) (SERVICE=aultdbXDB
instance_number                1
memory_target                  2197815296
open_cursors                   300
processes                      150
remote_listener                LISTENERS_AULTDB
remote_login_passwordfile      EXCLUSIVE
spfile                         +DATA/aultdb/spfileaultdb.ora
star_transformation_enabled    TRUE
thread                         1
undo_tablespace                UNDOTBS1
          -----------------------------------------------------------
```

Figure 30: Initialization parameter section

It is always a good idea to verify that the initialization parameter settings are what they should be when analyzing AWR reports. A misplaced power of ten or a botched K to M to G calculation can make a world of difference in the efficiency of an SGA.

You should also look for odd, undocumented parameter settings and track down the source of the suggested values shown if you are not familiar with the settings yourself. Many times a DBA will carry settings from one version of Oracle Database to another during upgrades without considering whether or not special settings are still appropriate.

If we weren't using Oracle RAC this would be the last section of the report; however, the use of Oracle RAC adds some additional global enqueue statistics sections to the AWR report.

Global enqueue and other Oracle RAC sections

With Oracle RAC comes many new statistics and areas to monitor. The AWR report has been expanded to provide detailed Oracle RAC sections to help you troubleshoot and diagnose Oracle RAC issues. Figure 31 shows the additional Oracle RAC sections pruned of statistics that generally won't be of use.

Global Enqueue Statistics DB/Inst: AULTDB/aultdb1 Snaps: 91-92

Statistic	Total	per Second	per Trans
acks for commit broadcast(actual)	216	0.1	0.2
acks for commit broadcast(logical	242	0.1	0.2
broadcast msgs on commit(actual)	700	0.2	0.6
broadcast msgs on commit(logical)	701	0.2	0.6
broadcast msgs on commit(wasted)	53	0.0	0.0
gcs assume no cvt	45,685	12.6	38.9
gcs blocked converts	205	0.1	0.2
gcs blocked cr converts	238	0.1	0.2
gcs compatible cr basts (local)	93,290	25.8	79.4
gcs dbwr flush pi msgs	36	0.0	0.0
gcs dbwr write request msgs	141	0.0	0.1
gcs immediate (compatible) conver	87	0.0	0.1
gcs immediate (null) converts	324	0.1	0.3
gcs immediate cr (compatible) con	11,512	3.2	9.8
gcs immediate cr (null) converts	213,759	59.2	181.9
gcs indirect ast	42,995	11.9	36.6
gcs indirect fg ast	42,995	11.9	36.6
gcs msgs process time(ms)	15,443	4.3	13.1
gcs msgs received	549,546	152.1	467.7
gcs new served by master	102	0.0	0.1
gcs pings refused	8	0.0	0.0
gcs retry convert request	16,809	4.7	14.3
gcs side channel msgs actual	5,444	1.5	4.6
gcs side channel msgs logical	146,320	40.5	124.5
ges msgs process time(ms)	344	0.1	0.3
ges msgs received	15,248	4.2	13.0
global posts queue time	318	0.1	0.3
global posts queued	59	0.0	0.1
global posts requested	63	0.0	0.1
global posts sent	59	0.0	0.1
implicit batch messages received	26,591	7.4	22.6
implicit batch messages sent	28,441	7.9	24.2
messages flow controlled	2,047	0.6	1.7
messages queue sent actual	68,909	19.1	58.6
messages queue sent logical	142,750	39.5	121.5
messages received actual	236,086	65.3	200.9
messages received logical	564,794	156.3	480.7
messages sent directly	134,160	37.1	114.2
messages sent indirectly	245,700	68.0	209.1
messages sent not implicit batche	40,468	11.2	34.4
messages sent pbatched	306,413	84.8	260.8
msgs causing lmd to send msgs	5,845	1.6	5.0
msgs causing lms(s) to send msgs	19,392	5.4	16.5
msgs received queue time (ms)	41,000	11.3	34.9
msgs received queued	564,808	156.3	480.7
msgs sent queue time (ms)	146,886	40.7	125.0
msgs sent queue time on ksxp (ms)	322,477	89.2	274.4
msgs sent queued	139,264	38.5	118.5
msgs sent queued on ksxp	243,401	67.4	207.1
process batch messages received	50,530	14.0	43.0
process batch messages sent	50,185	13.9	42.7

Global CR Served Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92

Statistic	Total
CR Block Requests	486
CURRENT Block Requests	36
Data Block Requests	486
Undo Block Requests	0
TX Block Requests	7
Current Results	522
Fairness Down Converts	78
Fairness Clears	10
Flushes	4

Global CURRENT Served Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> Pins = CURRENT Block Pin Operations
-> Flushes = Redo Flush before CURRENT Block Served Operations
-> Writes = CURRENT Block Fusion Write Operations

Statistic	Total	% <1ms	% <10ms	% <100ms	% <1s	% <10s
Pins	93,484	99.95	0.00	0.05	0.00	0.00
Flushes	1	0.00	0.00	100.00	0.00	0.00
Writes	43	0.00	4.65	74.42	18.60	2.33

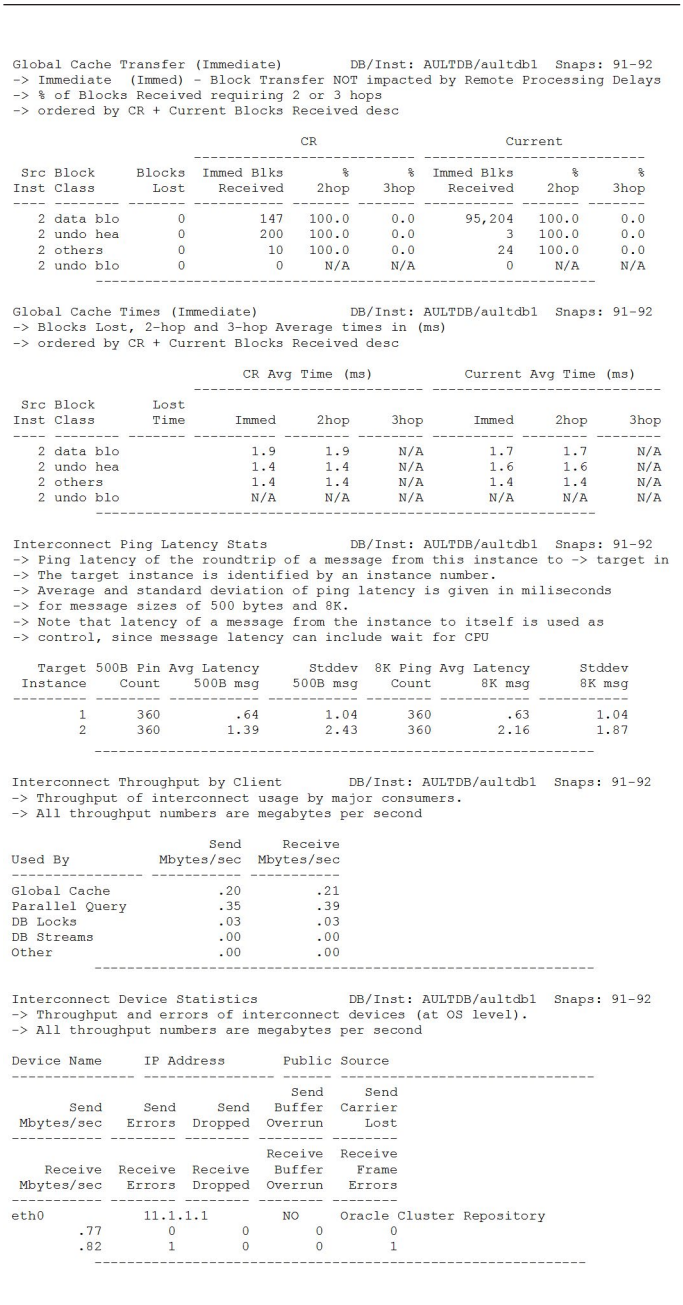
Global Cache Transfer Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> Immediate (Immed) - Block Transfer NOT impacted by Remote Processing Delays
-> Busy (Busy) - Block Transfer impacted by Remote Contention
-> Congested (Congst) - Block Transfer impacted by Remote System Load
-> ordered by CR + Current Blocks Received desc

		CR				Current			
Inst No	Block Class	Blocks Received	% Immed	% Busy	% Congst	Blocks Received	% Immed	% Busy	% Congst
2	data block	150	98.0	2.0	.0	95,402	99.8	.0	.2
2	undo header	200	100.0	.0	.0	3	100.0	.0	.0
2	Others	10	100.0	.0	.0	24	100.0	.0	.0

Global Cache Transfer Times (ms) DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> Avg Time - average time of all blocks (Immed,Busy,Congst) in ms
-> Immed, Busy, Congst - Average times in ms
-> ordered by CR + Current Blocks Received desc

		CR Avg Time (ms)				Current Avg Time (ms)			
Inst No	Block Class	All	Immed	Busy	Congst	All	Immed	Busy	Congst
2	data blo	2.5	1.9	27.6	N/A	1.8	1.7	N/A	9.1
2	undo hea	1.4	1.4	N/A	N/A	1.6	1.6	N/A	N/A
2	others	1.4	1.4	N/A	N/A	1.4	1.4	N/A	N/A
2	undo blo	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Figure 31: Global enqueue and oracle RAC statistics



```
Global Cache Transfer (Immediate)        DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Immediate  (Immed) - Block Transfer NOT impacted by Remote Processing Delays
-> % of Blocks Received requiring 2 or 3 hops
-> ordered by CR + Current Blocks Received desc

                              CR                        Current
                    -------------------------- --------------------------
 Src Block     Blocks  Immed Blks       %       %  Immed Blks       %       %
Inst Class       Lost    Received    2hop    3hop    Received    2hop    3hop
---- -------- ------- ---------- ------- ------- ---------- ------- -------
   2 data blo       0        147   100.0     0.0     95,204   100.0     0.0
   2 undo hea       0        200   100.0     0.0          3   100.0     0.0
   2 others         0         10   100.0     0.0         24   100.0     0.0
   2 undo blo       0          0     N/A     N/A          0     N/A     N/A
          -------------------------------------------------------------

Global Cache Times (Immediate)           DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Blocks Lost, 2-hop and 3-hop Average times in (ms)
-> ordered by CR + Current Blocks Received desc

                          CR Avg Time (ms)           Current Avg Time (ms)
                    ---------------------------- ----------------------------
 Src Block       Lost
Inst Class       Time      Immed     2hop     3hop      Immed     2hop     3hop
---- -------- ------- ---------- -------- -------- ---------- -------- --------
   2 data blo                1.9      1.9      N/A        1.7      1.7      N/A
   2 undo hea                1.4      1.4      N/A        1.6      1.6      N/A
   2 others                  1.4      1.4      N/A        1.4      1.4      N/A
   2 undo blo                N/A      N/A      N/A        N/A      N/A      N/A
          -------------------------------------------------------------

Interconnect Ping Latency Stats          DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Ping latency of the roundtrip of a message from this instance to -> target in
-> The target instance is identified by an instance number.
-> Average and standard deviation of ping latency is given in miliseconds
-> for message sizes of 500 bytes and 8K.
-> Note that latency of a message from the instance to itself is used as
-> control, since message latency can include wait for CPU

   Target 500B Pin Avg Latency      Stddev  8K Ping Avg Latency      Stddev
 Instance     Count    500B msg    500B msg     Count      8K msg      8K msg
--------- -------- ---------- ---------- -------- ---------- ----------
        1      360         .64        1.04      360         .63        1.04
        2      360        1.39        2.43      360        2.16        1.87
          -------------------------------------------------------------

Interconnect Throughput by Client        DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Throughput of interconnect usage by major consumers.
-> All throughput numbers are megabytes per second

                         Send     Receive
Used By             Mbytes/sec  Mbytes/sec
---------------- ---------- ----------
Global Cache              .20         .21
Parallel Query            .35         .39
DB Locks                  .03         .03
DB Streams                .00         .00
Other                     .00         .00
          -------------------------------------------------------------

Interconnect Device Statistics           DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> Throughput and errors of interconnect devices (at OS level).
-> All throughput numbers are megabytes per second

Device Name     IP Address       Public Source
--------------- ---------------- ------ ------------------------------

                                   Send     Send
      Send     Send     Send     Buffer  Carrier
 Mbytes/sec   Errors  Dropped  Overrun     Lost
---------- -------- -------- -------- --------

                                Receive  Receive
    Receive  Receive  Receive   Buffer    Frame
 Mbytes/sec   Errors  Dropped  Overrun   Errors
---------- -------- -------- -------- --------
eth0            11.1.1.1         NO     Oracle Cluster Repository
        .77        0        0        0        0
        .82        1        0        0        1
          -------------------------------------------------------------
```

Figure 31: Global enqueue and Oracle RAC statistics

Global enqueue statistics

Most of the global enqueue statistics really aren't useful unless you are trying to track down a particular type of event such as *null to s* conversion rates or other statistics that deal with how locks are converted throughout the system. It is a bit beyond this paper to delve deeply into all the ins and outs of the various GES statistics at this time.

Global CR served statistics

Global consistent read statistics show you how often blocks are being requested and sent due to consistent read activity. Too much of this shows hot blocks and the possibility that your block size is too large or you need to adjust rows-per-block.

Global current served statistics

Current blocks being transferred shows that multiple transactions are requiring the same blocks to make changes. This usually indicates a very busy system and may show a need to reduce block size or rows per block.

Usually you want to be sure that the transfer times histograms show a majority of transfers occurred at less time than your average read/write latency to the disks. If your transfer times are consistently greater than your disk latency times then your interconnect is becoming a bottleneck and you need to look at tuning it or replacing it with a higher bandwidth, lower latency technology such as Infiniband.

Global cache transfer statistics

Watching the number of blocks transferred is rather like watching waits; without the time component the number is useless. If 10,000 blocks are transferred and it takes 1 second you are happy; if 100 blocks are transferred and it takes 10 seconds you are worried (or should be). So, while knowing the number of blocks and types of blocks is good, later sections that actually give you the time breakdowns for transfers are more important.

Global cache transfer times

The global cache transfer times are the more critical of the statistics. Notice in this section how all of the statistics, save one, are less than 2 milliseconds. For data blocks (busy) however, we see that it is taking 26.7 milliseconds to transfer a data block. In this case it is probably a transaction-related locking issue because none of the other timings are bad.

Global cache transfer (immediate)

Immediate mode transfers are detailed in this section for both consistent read and current blocks. Again, this shows a breakdown of numbers of blocks by data type, which can help isolate which segments to examine for issues. However, always pair this with the next section before panicking over the number of blocks of a particular type shown to be whisking their way across the interconnect. Always look at timing data along with block counts.

Global cache times (immediate)

From looking at the times for the various types of CR and current clocks, we see that in no case were the transfer times excessive, so even if we did transfer 95,204 current blocks via two hops we did it at 1.7 millisecond per block, almost three times faster than the best disk latency (5ms). When transfer times become excessive, look at tuning the interconnect, reducing the number of blocks transferred by tuning SQL, or replacing the interconnect with a better technology.

Interconnect ping latency stats

By doing periodic pings, the system can get a rough idea of what transfer speeds should be. By looking at these ping statistics you can see for different message sizes how well the interconnect is transferring data. If the times are excessive for various ping sizes, then look at tuning the underlying TPC/UDP buffers. If that doesn't help, replace the interconnect.

Interconnect throughput by client

You should examine the throughput by client when you are seeing excessive latency in the interconnect statistics. By examining the amount of data being transferred across the interconnect you can determine if the interconnect is being overloaded and what is causing the long latencies. Usually an overloaded interconnect will start dropping blocks, and if you see the dropped blocks parameters for the various instances above zero then it may indicate the interconnect is seeing a lot of stress.

Interconnect device statistics

In this section, each of the NICs or interconnect devices is shown, along with statistics on throughput and error counts. If you see non-zero dropped statistics (one or two isn't a worry but dozens are an alarm) then either you have a buffer issue or the interconnect is overloaded.

Summary

Well, we have finally reached the end of the AWR report. The AWR report is the front line tool for tuning your Oracle Database. You should remember that the AWR is a licensed tool; however, Statspack, which contains almost all of the same information, is still available even for Oracle Database 11g.

IBM Corporation
New Orchard Road
Armonk, NY 10504

Produced in the United States of America
March 2015

This document is current as of the initial date of publication and may be changed by IBM at any time. Not all offerings are available in every country in which IBM operates.

The information in this document may include technical inaccuracies or typographical errors.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Please Recycle

ORW03038-USEN-01