AWR Reports

Introduction

AWR periodically gathers and stores system activity and workload data which is then analyzed by ADDM. Every layer of Oracle is equipped with instrumentation that gathers information on workload which will then be used to make self-managing decisions. AWR is the place where this data is stored. AWR looks periodically at the system performance (by default every 60 minutes) and stores the information found (by default up to 7 days). AWR runs by default and Oracle states that it does not add a noticeable level of overhead. A new background server process (MMON) takes snapshots of the in-memory database statistics (much like STATSPACK) and stores this information in the repository. MMON also provides Oracle with a server initiated alert feature, which notifies database administrators of potential problems (out of space, max extents reached, performance thresholds, etc.). The information is stored in the SYSAUX tablespace. This information is the basis for all self-management decisions. For example, it is possible to identify the SQL statements that have the:
* largest CPU consumption
* most buffer gets
* disk reads
* most parse calls
* shared memory

To **access Automatic Workload Repository** through **Oracle Enterprise Manager** Database Control:
On the **Administration** page, select the **Workload Repository** link under **Workload**. From the **Automatic Workload Repository** page, you can manage snapshots or modify AWR settings.
o To manage snapshots, click the link next to **Snapshots** or **Preserved Snapshot Sets**. On the **Snapshots** or **Preserved Snapshot Sets** pages, you can:
+ View information about snapshots or preserved snapshot sets (baselines).
+ Perform a variety of tasks through the pull-down **Actions** menu, including creating additional snapshots, preserved snapshot sets from an existing range of snapshots, or an ADDM task to perform analysis on a range of snapshots or a set of preserved snapshots.
o To modify AWR settings, click the **Edit** button. On the **Edit Settings** page, you can set the **Snapshot Retention** period and **Snapshot Collection** interval.

Both the snapshot frequency and retention time can be modified by the user. To see the present settings, you could use:

```
select snap_interval, retention from dba_hist_wr_control;

SNAP_INTERVAL       RETENTION
------------------- -------------------
+00000 01:00:00.0   +00007 00:00:00.0

or
select dbms_stats.get_stats_history_availability from dual;
select dbms_stats.get_stats_history_retention from dual;
```

This SQL shows that the snapshots are taken every hour and the collections are retained for 7 days

If you want to extend that retention period you can execute:

```
      execute dbms_workload_repository.modify_snapshot_settings(
          interval => 60,        -- In Minutes. Current value retained if NULL.
          retention => 43200);   -- In Minutes (= 30 Days). Current value retained if NULL
```

In this example the retention period is specified as 30 days (43200 min) and the interval between each snapshot is 60 min.

Differences or Advantage between AWR and STATSPACK report

1)The AWR is the next evolution of the STATSPACK utility.

2)The AWR repository holds all of the statistics available in STATSPACK as well as some additional statistics which are not.

3)STATSPACK does not store the Active Session History (ASH) statistics which are available in the AWR dba_hist_active_sess_history view.

4)An important difference between STATSPACK and the AWR is that STATSPACK does not store history for new metric statistics introduced in Oracle. The key AWR views are: *dba_hist_sysmetric_history* and *dba_hist_sysmetric_summary*.

5)The AWR also contains views such as *dba_hist_service_stat* , *dba_hist_service_wait_class* and *dba_hist_service_name* , which store history for performance cumulative statistics tracked for specific services.

6)The latest version of STATSPACK included with Oracle contains a set of specific tables, which track history of statistics that reflect the performance of the Oracle Streams feature. These tables are stats$streams_capture , stats$streams_apply_sum , stats$buffered_subscribers , stats$rule_set , stats$propagation_sender , stats$propagation_receiver and stats$buffered_queues . The AWR does not contain the specific tables that reflect Oracle Streams activity; therefore, if a DBA relies heavily on the Oracle Streams feature, it would be useful to monitor its performance using STATSPACK utility.

7)Statspack snapshots must be run by an external scheduler (dbms_jobs, CRON, etc.). AWR snapshots are scheduled every 60 minutes by default. Administrators can manually adjust the snapshot interval if so desired.

8)ADDM captures a much greater depth and breadth of statistics than Statspack does. During snapshot processing, MMON transfers an in-memory version of the statistics to the permanent statistics tables.

9)Statspack snapshot purges must be scheduled manually. When the Statspack tablespace runs out of space, Statspack quits working. AWR snapshots are purged automatically by MMON every night. MMON, by default, tries to keep one week's worth of AWR snapshots available. If AWR detects that the SYSAUX tablespace is in danger of running out of space, it will free space in SYSAUX by automatically deleting the oldest set of snapshots. If this occurs, AWR will initiate a server-generated alert to notify administrators of the out-of-space error condition. Administrators can manually adjust the amount of information retained by invoking the MODIFY_SNAPSHOT_SETTINGS PL/SQL stored procedure and specifying the RETENTION parameter input variable.

10)AWR snapshots provide a persistent view of database statistics. They are stored in the system-defined schema, which resides in a new tablespace called SYSAUX. A snapshot is a collection of performance statistics that are captured at a specific point in time. The snapshot data points are used to compute the rate of change for the statistic being measured. A unique SNAP_ID snapshot identifier identifies each snapshot.

Workload Repository Reports

Oracle provide two scripts to produce workload repository reports (awrrpt.sql and awrrpti.sql). They are similar in format to the statspack reports and give the option of HTML or plain text formats. The two reports give essential the same output but the awrrpti.sql allows you to select a single instance. The reports can be generated as follows:

    @$ORACLE_HOME/rdbms/admin/awrrpt.sql
    @$ORACLE_HOME/rdbms/admin/awrrpti.sql

The scripts prompt you to enter the report format (html or text), the start snapshot id, the end snapshot id and the report filename. This script looks like Statspack; it shows all the AWR snapshots available and asks for two specific ones as interval boundaries. It produces two types of output: text format, similar to that of the Statspack report but from the AWR repository, and the default HTML format, complete with hyperlinks to sections and subsections, providing quite a user-friendly report. Run the script and take a look at the report now to get an idea about capabilities of the AWR.

If you want to explore the AWR repository, feel free to do so. The AWR consists of a number of tables owned by the SYS schema and stored in the SYSAUX tablespace. All AWR table names starts with the identifier “WR.” Following WR is a mnemonic that identifies the type designation of the table followed by a dollar sign ($). AWR tables come with three different type designations:

- Metadata (WRM$)
- Historical data (WRH$)
- AWR tables related to advisor functions (WRI$)

Most of the AWR table names are pretty self-explanatory, such as WRM$_SNAPSHOT or WRH$_ACTIVE_SESSION_HISTORY.
Also Oracle offers several DBA tables that allow you to query the AWR repository. The tables all start with DBA_HIST, followed by a name that describes the table. These include tables such as DBA_HIST_FILESTATS, DBA_HIST_DATAFILE, or DBA_HIST_SNAPSHOT. The AWR history tables capture a lot more information than Statspack, including tablespace usage, filesystem usage, even operating system statistics. A complete list of these tables can be seen from the data dictionary through:

```
select view_name from user_views
where view_name like 'DBA\_HIST\_%' escape '\';
```

AWR Snapshots and Baselines

You can create a snapshot manually using:

```
EXEC dbms_workload_repository.create_snapshot;
```

You can see what snapshots are currently in the AWR by using the DBA_HIST_SNAPSHOT view as seen in this example:

```
SELECT snap_id, to_char(begin_interval_time,'dd/MON/yy hh24:mi') Begin_Interval,
       to_char(end_interval_time,'dd/MON/yy hh24:mi') End_Interval
FROM dba_hist_snapshot
ORDER BY 1;

   SNAP_ID BEGIN_INTERVAL  END_INTERVAL
---------- --------------- ---------------
       954 30/NOV/05 03:01 30/NOV/05 04:00
       955 30/NOV/05 04:00 30/NOV/05 05:00
       956 30/NOV/05 05:00 30/NOV/05 06:00
       957 30/NOV/05 06:00 30/NOV/05 07:00
       958 30/NOV/05 07:00 30/NOV/05 08:00
       959 30/NOV/05 08:00 30/NOV/05 09:00
```

Each snapshot is assigned a unique snapshot ID that is reflected in the SNAP_ID column. The END_INTERVAL_TIME column displays the time that the actual snapshot was taken.

Sometimes you might want to drop snapshots manually. The *dbms_workload_repository.drop_snapshot_range* procedure can be used to remove a range of snapshots from the AWR. This procedure takes two parameters, low_snap_id and high_snap_id, as seen in this example:

```
EXEC dbms_workload_repository.drop_snapshot_range(low_snap_id=>1107, high_snap_id=>1108);
```

The following workload repository views are available:
    * V$ACTIVE_SESSION_HISTORY - Displays the active session history (ASH) sampled every second.
    * V$METRIC - Displays metric information.
    * V$METRICNAME - Displays the metrics associated with each metric group.
    * V$METRIC_HISTORY - Displays historical metrics.
    * V$METRICGROUP - Displays all metrics groups.
    * DBA_HIST_ACTIVE_SESS_HISTORY - Displays the history contents of the active session history.
    * DBA_HIST_BASELINE - Displays baseline information.
    * DBA_HIST_DATABASE_INSTANCE - Displays database environment information.
    * DBA_HIST_SNAPSHOT - Displays snapshot information.
    * DBA_HIST_SQL_PLAN - Displays SQL execution plans.
    * DBA_HIST_WR_CONTROL - Displays AWR settings.

Finally , you can use the following query to identify the occupants of the SYSAUX Tablespace

```
select substr(occupant_name,1,40), space_usage_kbytes
  from v$sysaux_occupants;
```

AWR Automated Snapshots

Oracle uses a scheduled job, GATHER_STATS_JOB, to collect AWR statistics. This job is created, and enabled automatically, when you create a new Oracle database. To see this job, use the DBA_SCHEDULER_JOBS view as seen in this example:

```
SELECT a.job_name, a.enabled, c.window_name, c.schedule_name, c.start_date, c.repeat_interval
FROM dba_scheduler_jobs a, dba_scheduler_wingroup_members b, dba_scheduler_windows c
WHERE job_name='GATHER_STATS_JOB'
  And a.schedule_name=b.window_group_name
  And b.window_name=c.window_name;
```

You can disable this job using the dbms_scheduler.disable procedure as seen in this example:

```
Exec dbms_scheduler.disable('GATHER_STATS_JOB');
```

And you can enable the job using the dbms_scheduler.enable procedure as seen in this example:

```
Exec dbms_scheduler.enable('GATHER_STATS_JOB');
```

AWR Baselines

It is frequently a good idea to create a baseline in the AWR. A baseline is defined as a range of snapshots that can be used to compare to other pairs of snapshots. The Oracle database server will exempt the snapshots assigned to a specific baseline from the automated purge routine. Thus, the main purpose of a baseline is to preserve typical runtime statistics in the AWR repository, allowing you to run the AWR snapshot reports on the preserved baseline snapshots at any time and compare them to recent snapshots contained in the AWR. This allows you to compare current performance (and configuration) to established baseline performance, which can assist in determining database performance problems.

Creating baselines
You can use the *create_baseline* procedure contained in the dbms_workload_repository stored PL/SQL package to create a baseline as seen in this example:
```
EXEC dbms_workload_repository.create_baseline (start_snap_id=>1109, end_snap_id=>1111, baseline_name=>'EOM Baseline');
```

Baselines can be seen using the DBA_HIST_BASELINE view as seen in the following example:
```
SELECT baseline_id, baseline_name, start_snap_id, end_snap_id
FROM dba_hist_baseline;

BASELINE_ID BASELINE_NAME   START_SNAP_ID END_SNAP_ID
----------- --------------- ------------- -----------
          1 EOM Baseline             1109        1111
```

In this case, the column BASELINE_ID identifies each individual baseline that has been defined. The name assigned to the baseline is listed, as are the beginning and ending snapshot IDs.

Removing baselines
The pair of snapshots associated with a baseline are retained until the baseline is explicitly deleted. You can remove a baseline using the dbms_workload_repository.drop_baseline procedure as seen in this example that drops the “EOM Baseline” that we just created.
```
EXEC dbms_workload_repository.drop_baseline (baseline_name=>'EOM Baseline', Cascade=>FALSE);
```

Note that the cascade parameter will cause all associated snapshots to be removed if it is set to TRUE; otherwise, the snapshots will be cleaned up automatically by the AWR automated processes.

Moving AWR information

Enterprise Manager allows administrators to transfer Automatic Workload Repository snapshots to other workload repositories for offline analysis. This is accomplished by the administrator specifying a snapshot range and extracting the AWR data to a flat file. The flat file is then loaded into a user-specified staging schema in the target repository. To complete the transfer, the data is copied from the staging schema into the target repository's SYS schema. The data in the SYS schema is then used as the source for the ADDM analysis.
If the snapshot range already exists in the SYS or staging schemas, the data being imported is ignored. All data in snapshot ranges that does not conflict with existing data is loaded. Oracle contains a new package DBMS_SWRF_INTERNAL to provide AWR snapshot export and import functionality.
The example below exports a snapshot range starting with 100 and ending at 105 to the output dump file 'awr_wmprod1_101_105' in the directory '/opt/oracle/admin/awrdump/wmprod1':
```
BEGIN
DBMS_SWR_INTERNAL.AWR_EXTRACT(
DMPFILE =>'awr_export_wmprod1_101_105',
DMPDIR => '/opt/oracle/admin/awrdump/wmprod1',
BID => 101,
EID => 105)
```

We then use the AWR_LOAD procedure to load the data into our target repository staging schema:
```
BEGIN
DBMS_SWR_INTERNAL.AWR_LOAD(
SCHNAME => 'foot',
DMPFILE =>'awr_export_wmprod1_101_105',
DMPDIR => '/opt/oracle/admin/awrdump/wmprod1')
```
The last step is to transfer the data from our staging schema (FOOT) to the SYS schema for analysis:
```
BEGIN
DBMS_SWR_INTERNAL.MOVE_TO_AWR(SCHNAME => 'foot',)
```

Useful Queries

Standard reports
The SQL*Plus scripts are located in $ORACLE_HOME/rdbms/admin directory, the output in either text or HTML (default) format

ASH report (ashrpt.sql)
Helps answer questions about “What’s going on right now and who is doing it?” for a specified time period

AWR report (awrrpt.sql)
Breakdown of what was consuming “DB Time” for a specified time period

AWR “diff” report (awrddrpt.sql)
Compares and highlights what changed between two specified time periods.

Prompts for:
- Html or text type
- First pair of Begin and End snapshot ids
- Report name
- Provides a report name beginning with awrdiff…

AWR “SQL” report (awrsqrpt.sql)
Displays all recorded information about a specific SQL during a specified time period. Good when you want to focus on a particular SQL statement.

To get a quick report of any SQL statement’s execution plans within the past 7 days, if you have the statement’s SQL ID value:
```
select * from table(dbms_xplan.display_awr(‘sqlid’)) ;
```

If you don’t have the SQL statement’s SQL ID, it can be found in:
- A standard AWR report or EM DB Console or Grid Control or
- Query the V$SQL or DBA_HIST_SQLTEXT view:
```
select sql_id, sql_text from v$sql where lower(sql_text) like ‘%phrase%’ ;
select sql_id, sql_text from dba_hist_sqltext where lower(sql_text) like ‘%phrase%’ ;
```

Quick Summary on AWR Sections

This section contains detailed guidance for evaluating each section of an AWR report.
Report Summary Section:
This gives an overall summary of the instance during the snapshot period, and it contains important aggregate summary information.
- Cache Sizes: This shows the size of each SGA region after AMM has changed them. This information can be compared to the original init.ora parameters at the end of the AWR report.
- Load Profile: This section shows important rates expressed in units of per second and transactions per second.
- Instance Efficiency Percentages: With a target of 100%, these are high-level ratios for activity in the SGA.
- Shared Pool Statistics: This is a good summary of changes to the shared pool during the snapshot period.
- Top 5 Timed Events: This is the most important section in the AWR report. It shows the top wait events and can quickly show the overall database bottleneck.

Wait Events Statistics Section
This section shows a breakdown of the main wait events in the database including foreground and background database wait events as well as time model, operating system, service, and wait classes statistics.
- Time Model Statistics: Time mode statistics report how database-processing time is spent. This section contains detailed timing information on particular components participating in database processing.
- Wait Class:
- Wait Events: This AWR report section provides more detailed wait event information for foreground user processes which includes Top 5 wait events and many other wait events that occurred during the snapshot interval.
- Background Wait Events: This section is relevant to the background process wait events.
- Operating System Statistics: The stress on the Oracle server is important, and this section shows the main external resources including I/O, CPU, memory, and network usage.
- Service Statistics: The service statistics section gives information about how particular services configured in the database are operating.
- Service Wait Class Stats:

SQL Statistics Section
This section displays top SQL, ordered by important SQL execution metrics.
- SQL Ordered by Elapsed Time: Includes SQL statements that took significant execution time during processing.

- SQL Ordered by CPU Time: Includes SQL statements that consumed significant CPU time during its processing.
- SQL Ordered by Gets: These SQLs performed a high number of logical reads while retrieving data.
- SQL Ordered by Reads: These SQLs performed a high number of physical disk reads while retrieving data.
- SQL Ordered by Executions:
- SQL Ordered by Parse Calls: These SQLs experienced a high number of reparsing operations.
- SQL Ordered by Sharable Memory: Includes SQL statements cursors which consumed a large amount of SGA shared pool memory.
- SQL Ordered by Version Count: These SQLs have a large number of versions in shared pool for some reason.
- Complete List of SQL Text:

Instance Activity Stats
This section contains statistical information describing how the database operated during the snapshot period.
- Instance Activity Stats - Absolute Values: This section contains statistics that have absolute values not derived from end and start snapshots.
- Instance Activity Stats - Thread Activity: This report section reports a log switch activity statistic.

I/O Stats Section
This section shows the all important I/O activity for the instance and shows I/O activity by tablespace, data file, and includes buffer pool statistics.
- Tablespace IO Stats
- File IO Stats

Buffer Pool Statistics Section

Advisory Statistics Section
This section show details of the advisories for the buffer, shared pool, PGA and Java pool.
- Instance Recovery Stats:
- Buffer Pool Advisory:
- PGA Aggr Summary: PGA Aggr Target Stats; PGA Aggr Target Histogram; and PGA Memory Advisory.
- Shared Pool Advisory:
- SGA Target Advisory
- Stream Spool Advisory
- Java Pool Advisory

Wait Statistics Section
- Buffer Wait Statistics: This important section shows buffer cache waits statistics.
- Enqueue Activity: This important section shows how enqueue operates in the database. Enqueues are special internal structures which provide concurrent access to various database resources.

Undo Statistics Section
- Undo Segment Summary: This section gives a summary about how undo segments are used by the database.
- Undo Segment Stats: This section shows detailed history information about undo segment activity.

Latch Statistics Section:
This section shows details about latch statistics. Latches are a lightweight serialization mechanism that is used to single-thread access to internal Oracle structures.
- Latch Activity
- Latch Sleep Breakdown
- Latch Miss Sources
- Parent Latch Statistics
- Child Latch Statistics

Segment Statistics Section:
This report section provides details about hot segments using the following criteria:
- Segments by Logical Reads: Includes top segments which experienced high number of logical reads.
- Segments by Physical Reads: Includes top segments which experienced high number of disk physical reads.
- Segments by Row Lock Waits: Includes segments that had a large number of row locks on their data.
- Segments by ITL Waits: Includes segments that had a large contention for Interested Transaction List (ITL). The contention for ITL can be reduced by increasing INITRANS storage parameter of the table.
- Segments by Buffer Busy Waits: These segments have the largest number of buffer waits caused by their data blocks.

Dictionary Cache Stats Section
This section exposes details about how the data dictionary cache is operating.

Library Cache Section
Includes library cache statistics describing how shared library objects are managed by Oracle.

Memory Statistics Section
- Process Memory Summary
- SGA Memory Summary: This section provides summary information about various SGA regions.
- SGA Breakdown difference:

Streams Statistics Section
- Streams CPU/IO Usage
- Streams Capture
- Streams Apply
- Buffered Queues
- Buffered Subscribers
- Rule Set

Reading the AWR Report

The main sections in an AWR report include:

AWR Report Header:
This section shows basic information about the report like when the snapshot was taken, for how long, Cache Sizes at the beginning and end of the Snapshot, etc.

```
WORKLOAD REPOSITORY report for

DB Name         DB Id    Instance     Inst Num Release     RAC Host
------------ ----------- ------------ -------- ----------- --- ------------
IRD_PRD        492209413 ird_prd             1 10.2.0.3.0  NO  oradb1p1

              Snap Id      Snap Time      Sessions Curs/Sess
            --------- ------------------- -------- ---------
Begin Snap:      1664 24-Nov-07 05:00:57        39       6.5
  End Snap:      1716 26-Nov-07 09:00:44        53      10.5
   Elapsed:            3,119.78 (mins)
   DB Time:           17,737.68 (mins)

Cache Sizes
~~~~~~~~~~                       Begin        End
                             ---------- ----------
               Buffer Cache:       356M       352M  Std Block Size:         8K
           Shared Pool Size:       124M       124M      Log Buffer:     6,180K
```

Load Profile:
This section shows important rates expressed in units of per second and transactions per second.

```
Load Profile
~~~~~~~~~~~~                            Per Second       Per Transaction
                                   ---------------       ---------------
                  Redo size:             49,891.76            142,902.16
              Logical reads:             49,272.96            141,129.77
              Block changes:                416.82              1,193.86
             Physical reads:              4,850.46             13,892.89
            Physical writes:                173.59                497.21
```

```
                  User calls:                 14.42                 41.31
                      Parses:                  9.26                 26.51
                 Hard parses:                  1.15                  3.28
                       Sorts:                  9.85                 28.22
                      Logons:                  0.67                  1.91
                    Executes:                 42.97                123.07
                Transactions:                  0.35

  % Blocks changed per Read:    0.85    Recursive Call %:    95.32
 Rollback per transaction %:    0.00       Rows per Sort:  2335.07

Instance Efficiency Percentages (Target 100%)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            Buffer Nowait %:   99.98       Redo NoWait %:  100.00
            Buffer  Hit   %:   90.91    In-memory Sort %:  100.00
            Library Hit   %:   89.01        Soft Parse %:   87.63
         Execute to Parse %:   78.46         Latch Hit %:   99.62
Parse CPU to Parse Elapsd %:   50.49     % Non-Parse CPU:   99.58

 Shared Pool Statistics        Begin    End
                              ------  ------
             Memory Usage %:   85.54   84.14
    % SQL with executions>1:   86.18   90.92
  % Memory for SQL w/exec>1:   83.84   91.20

Load Profile
~~~~~~~~~~~~                            Per Second       Per Transaction
                                   ---------------       ---------------
                  Redo size:            351,530.67              7,007.37
              Logical reads:              5,449.81                108.64   --Key Metric
              Block changes:              1,042.08                 20.77
             Physical reads:                 37.71                  0.75   --Key Metric
            Physical writes:                134.68                  2.68   --Key Metric
                 User calls:              1,254.72                 25.01
                     Parses:                  4.92                  0.10
                Hard parses:                  0.02                  0.00
                      Sorts:                 15.73                  0.31
                     Logons:                 -0.01                  0.00
                   Executes:                473.73                  9.44   --Key Metric
               Transactions:                 50.17                         --Key Metric

  % Blocks changed per Read:   19.12    Recursive Call %:     4.71
 Rollback per transaction %:    2.24       Rows per Sort:    20.91
```

Where:
. **Redo size:** This is the amount of redo generated during this report. If you see an increase here then more DML statements are taking place (meaning your users are doing more INSERTs, UPDATEs, and DELETEs than before
. **Logical Reads:** This is calculated as Consistent Gets + DB Block Gets = Logical Reads
. **Block changes:** The number of blocks modified during the sample interval. If you see an increase here then more DML statements are taking place (meaning your users are doing more INSERTs, UPDATEs, and DELETEs than before
. **Physical Reads:** The number of requests for a block that caused a physical I/O.
. **Physical Writes:** The number of physical writes issued.
. **User Calls:** The number of queries generated
. **Parses:** Total of all parses: both hard and soft
. **Hard Parses:** Those parses requiring a completely new parse of the SQL statement. A 'hard parse' rate of greater than 100 per second indicates there is a very high amount of hard parsing on the system. High hard parse rates cause serious performance issues, and must be investigated. A high hard parse rate is usually accompanied by latch contention on the shared pool and library cache latches. Check whether waits for 'latch free' appear in the top-5 wait events, and if so, examine the latching sections of the report. Of course, we want a low number here.
. **Soft Parses:** Not listed but derived by subtracting the hard parses from parses. A soft parse reuses a previous hard parse and hence consumes far fewer resources. A high soft parse rate could be anywhere in the rate of 300 or more per second. Unnecessary soft parses also limit application scalability; optimally a SQL statement should be soft-parsed once per session, and executed many times.
. **Sorts and Logons** are all self explanatory
. **Executes:** how many statements we are executing per second / transaction
. **Transactions:** how many transactions per second we process

This gives an **overall view of the load on the server**. In this case, we are looking at a very good hard parse number and a fairly high system load.

The per-second statistics show you the changes in throughput (i.e. whether the instance is performing more work per second). For example:
• a significant increase in 'redo size', 'block changes' and 'pct of blocks changed per read' would indicate the instance is performing more inserts/updates/deletes.
• an increase in the 'redo size' without an increase in the number of 'transactions per second' would indicate a changing transaction profile.
Similarly, looking at the per-transaction statistics allows you to identify changes in the application characteristics by comparing these to the corresponding statistics from the baseline report.

Sample Analysis

```
Load Profile
~~~~~~~~~~~~                            Per Second       Per Transaction
                                   ---------------       ---------------
                  Redo size:          1,316,849.03              6,469.71
              Logical reads:             16,868.21                 82.87
              Block changes:              5,961.36                 29.29
             Physical reads:                  7.51 (B)              0.04
            Physical writes:              1,044.74 (B)              5.13
                 User calls:              8,432.99 (C)             41.43
                     Parses:              1,952.99 (D)              9.60
                Hard parses:                  0.01 (D)              0.00
                      Sorts:                  1.44                  0.01
                     Logons:                  0.05                  0.00
                   Executes:              1,954.97                  9.60
               Transactions:                203.54

  % Blocks changed per Read:   35.34 (A)   Recursive Call %:    25.90
 Rollback per transaction %:    9.55       Rows per Sort:   137.38
```

Observations:
• This system is generating a lot of redo (1mb/s), with 35% of all blocks read being updated. (A)
• Comparing the number of Physical reads per second to the number of Physical writes per second shows the physical read to physical write ratio is very low (1:49). Typical OLTP systems have a read-to-write ratio of 10:1 or 5:1 - this ratio (at 1:49) is quite unusual. (B)
• This system is quite busy, with 8,432 User calls per second.(C)
• The total parse rate (Parses per second) seems to be high, with the Hard parse rate very low, which implies the majority of the parses are soft parses. The high parse rate may tie in with the latch free event, if the latch contended for is the library cache latch, however no assumptions should be made. (D)
On the whole, this seems to be a heavy workload, with many parses, and writes.

Evaluating the Instance Efficiency Percentages Section

```
Instance Efficiency Percentages (Target 100%)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            Buffer Nowait %:   99.99       Redo NoWait %:  100.00
            Buffer  Hit   %:   95.57    In-memory Sort %:   97.55
            Library Hit   %:   99.89        Soft Parse %:   99.72
         Execute to Parse %:   88.75         Latch Hit %:   99.11
Parse CPU to Parse Elapsd %:   52.66     % Non-Parse CPU:   99.99

 Shared Pool Statistics        Begin   End
                              ------  ------
             Memory Usage %:   42.07   43.53
    % SQL with executions>1:   73.79   75.08
  % Memory for SQL w/exec>1:   76.93   77.64
```

Interpreting the ratios in this section can be slightly more complex than it may seem at first glance. While high values for the ratios are generally good (indicating high efficiency), such values can be misleading your system may be doing something efficiently that it would be better off not doing at all. Similarly, low values aren't always bad. For example, a low in-memory sort ratio (indicating a low percentage of sorts performed in memory) would not necessarily be a cause for concern in a decision- support system (DSS) environment, where user response time is less critical than in an online transaction processing (OLTP) environment.

Basically, you need to keep in mind the characteristics of your application - whether it is query-intensive or update-intensive, whether it involves lots of sorting, and so on - when you're evaluating the Instance Efficiency Percentages.

It is possible for both the **'buffer hit ratio**' and the **'execute to parse**' ratios to be negative. In the case of the buffer hit ration, the buffer cache is too small and the data in is being aged out before it can be used so it must be retrieved again. This is a form of thrashing which degrades performance immensely.

Buffer Nowait Ratio. This is the percentage of time that the instance made a call to get a buffer (all buffer types are included here) and that buffer was made available immediately (meaning it didn't have to wait for the buffer...hence "Buffer Nowait"). If the ratio is low, then could be a (hot) block(s) being contended for that should be found in the Buffer Wait Section.. If the ratio is low, check the Buffer Wait Statistics section of the report for more detail on which type of block is being contended for.
Buffer Hit Ratio. (also known as the buffer-cache hit ratio) Ideally more than 95 percent. It shows the % of times a particular block was found in buffer cache insted of performing a physical I/O (reading from disk).
Although historically known as one of the most important statistics to evaluate, this ratio can sometimes be misleading. A low buffer hit ratio does not necessarily mean the cache is too small; it may be that potentially valid full-table scans are artificially reducing what is otherwise a good ratio. Similarly, a high buffer hit ratio (say, 99 percent) normally indicates that the cache is adequately sized, but this assumption may not always be valid. For example, frequently executed SQL statements that repeatedly refer to a small number of buffers via indexed lookups can create a misleadingly high buffer hit ratio. When these buffers are read, they are placed at the most recently used (MRU) end of the buffer cache; iterative access to these buffers can artificially inflate the buffer hit ratio. This inflation makes tuning the buffer cache a challenge. Sometimes you can identify a too-small buffer cache by the appearance of the write complete waits event, which indicates that hot blocks (that is, blocks that are still being modified) are aging out of the cache while they are still needed; check the Wait Events list for evidence of this event.
Library Hit Ratio. This ratio, also known as the library-cache hit ratio, gives the percentage of pin requests that result in pin hits. A pin hit occurs when the SQL or PL/SQL code to be executed is already in the library cache and is valid to execute. If the "Library Hit ratio" is low, it could be indicative of a shared pool that is too small (SQL is prematurely pushed out of the shared pool), or just as likely, that the system did not make correct use of bind variables in the application. If the soft parse ratio is also low, check whether there's a parsing issue. A lower ratio could also indicate that bind variables are not used or some other issue is causing SQL not to be reused (in which case a smaller shared pool may only be a band-aid that will potentially fix a library latch problem which may result).
Execute to Parse. If value is negative, it means that the number of parses is larger than the number of executions. Another cause for a negative execute to parse ratio is if the shared pool is too small and queries are aging out of the shared pool and need to be reparsed. This is another form of thrashing which also degrades performance tremendously. So, if you run some SQL and it has to be parsed every time you execute it (because no plan exists for this statement) then your percentage would be 0%. The more times that your SQL statement can reuse an existing plan the higher your Execute to Parse ratio is. This is very BAD!! One way to increase your parse ratio is to use bind variables.
Parse CPU to Parse Elapsd %: Generally, this is a measure of how available your CPU cycles were for SQL parsing. If this is low, you may see "latch free" as one of your top wait events.
Redo Nowait Ratio. This ratio indicates the amount of redo entries generated for which there was space available in the redo log. The instance didn't have to wait to use the redo log if this is 100%
The redo-log space-request statistic is incremented when an Oracle process attempts to write a redo-log entry but there is not sufficient space remaining in the online redo log. Thus, a value close to 100 percent for the redo nowait ratio indicates minimal time spent waiting for redo logs to become available, either because the logs are not filling up very often or because the database is able to switch to a new log quickly whenever the current log fills up.
If your alert log shows that you are switching logs frequently (that is, more than once every 15 minutes), you may be able to reduce the amount of switching by increasing the size of the online redo logs. If the log switches are not frequent, check the disks on which the redo logs reside to see why the switches are not happening quickly. If these disks are not overloaded, they may be slow, which means you could put the files on faster disks.
In-Memory Sort Ratio. This ratio gives the percentage of sorts that were performed in memory, rather than requiring a disk-sort segment to complete the sort. Optimally, in an OLTP environment, this ratio should be high. Setting the PGA_AGGREGATE_TARGET (or SORT_AREA_SIZE) initialization parameter effectively will eliminate this problem, as a minimum you pretend to have this one in 95%
Soft Parse Ratio. This ratio gives the percentage of parses that were soft, as opposed to hard. A soft parse occurs when a session attempts to execute a SQL statement and a usable version of the statement is already in the shared pool. In other words, all data (such as the optimizer execution plan) pertaining to the statement in the shared pool is equally applicable to the statement currently being issued. A hard parse, on the other hand, occurs when the current SQL statement is either not in the shared pool or not there in a shareable form. An example of the latter case would be when the SQL statement in the shared pool is textually identical to the current statement but the tables referred to in the two statements resolve to physically different tables.
Hard parsing is an expensive operation and should be kept to a minimum in an OLTP environment. The aim is to parse once, execute many times.
Ideally, the soft parse ratio should be greater than 95 percent. When the soft parse ratio falls much below 80 percent, investigate whether you can share SQL by using bind variables or force cursor sharing by using the `init.ora` parameter `cursor_sharing`.
Before you jump to any conclusions about your soft parse ratio, however, be sure to compare it against the actual hard and soft parse rates shown in the Load Profile. If the rates are low (for example, 1 parse per second), parsing may not be a significant issue in your system. Another useful standard of comparison is the proportion of parse time that was *not* CPU-related, given by the following ratio:

```
(parse time CPU) / (parse time elapsed)
```

A low value for this ratio could mean that the non-CPU-related parse time was spent waiting for latches, which might indicate a parsing or latching problem. To investigate further, look at the shared-pool and library-cache latches in the Latch sections of the report for indications of contention on these latches.
Latch Hit Ratio. This is the ratio of the total number of latch misses to the number of latch gets for all latches. A low value for this ratio indicates a latching problem, whereas a high value is generally good. However, as the data is rolled up over all latches, a high latch hit ratio can artificially mask a low get rate on a specific latch. Cross-check this value with the Top 5 Wait Events to see if latch free is in the list, and refer to the Latch sections of the report. Latch Hit % of less than 99 percent is usually a big problem.

Also check the "Shared Pool Statistics", if the "End" value is in the high 95%-100% range ,this is a indication that the shared pool needs to be increased (especially if the "Begin" value is much smaller)

% SQL with executions>1: Shows % of SQLs executed more than 1 time. The % should be very near to value 100.
% memory for SQL w/exec>1: From the memory space allocated to cursors, shows which % has been used by cursors more than 1.

The ratio above 80% is always healthy.

Sample Analysis

```
Instance Efficiency Percentages (Target 100%)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            Buffer Nowait %:   98.56       Redo NoWait %:  100.00
            Buffer  Hit   %:   99.96    In-memory Sort %:   99.84
            Library Hit   %:   99.99        Soft Parse %:  100.00 (A)
         Execute to Parse %:    0.10 (A)     Latch Hit %:   99.37
Parse CPU to Parse Elapsd %:   58.19 (A) % Non-Parse CPU:   99.84

 Shared Pool Statistics        Begin   End
                               ------  ------
             Memory Usage %:   28.80   29.04 (B)
    % SQL with executions>1:   75.91   76.03
  % Memory for SQL w/exec>1:   83.65   84.09
```

Observations:
• The 100% soft parse ratio (A) indicates the system is not hard-parsing. However the system is soft parsing a lot, rather than only re-binding and re-executing the same cursors, as the Execute to Parse % is very low (A). Also, the CPU time used for parsing (A) is only 58% of the total elapsed parse time (see Parse CPU to Parse Elapsd). This may also imply some resource contention during parsing (possibly related to the latch free event?).
• There seems to be a lot of unused memory in the shared pool (only 29% is used) (B). If there is insufficient memory allocated to other areas of the database (or OS), this memory could be redeployed

***Please see the following NOTES on shared pool issues
[NOTE:146599.1] Diagnosing and Resolving Error ORA-04031
[NOTE:62143.1] Understanding and Tuning the Shared Pool
[NOTE:105813.1] SCRIPT TO SUGGEST MINIMUM SHARED POOL SIZE

Top 5 Timed Events Section

Just knowing the breakdown of time into the above 3 categories is not very useful so Oracle has a set of 'Wait Events' for activities in 'a' and 'c', and can record CPU utilization for 'b'. This is best illustrated with a simplified example of few seconds in the life of an Oracle shadow process:

```
State   Notes...
~~~~~   ~~~~~~~~
IDLE    Waiting for 'SQL*Net message from client'.
        Receives a SQL*Net packet requesting 'parse/execute' of a statement
ON CPU  decodes the SQL*Net packet.
WAITING Waits for 'latch free' to obtain the a 'library cache' latch
        Gets the latch.
```

```
   ON CPU  Scans for the SQL statement in the shared pool, finds a match,
           frees latch , sets up links to the shared cursor etc.. & begins to
           execute.
   WAITING Waits for 'db file sequential read' as we need a block which is
           not in the buffer cache. Ie: Waiting for an IO to complete.
   ON CPU  Block read has completed so execution can continue.
           Constructs a SQL*Net packet to send back to the user containing
           the first row of data.
   WAITING Waits on 'SQL*Net message to client' for an acknowledgement that the
           SQL*Net packet was reliably delivered.
   IDLE    Waits on 'SQL*Net message from client' for the next thing to do.
```

This section is among **the most important and relevant sections in the report**. Here is where you find out what events (typically wait events) are consuming the most time.
When you are trying to eliminate bottlenecks on your system, your report's **Top 5 Timed Events section is the first place to look and you should use the HIGHEST WAIT TIMES to guide the investigation**.
This section of the report shows the top 5 wait events, the full list of wait events, and the background wait events. If your system's TIMED_STATISTICS initialization parameter is set to true, the events are ordered in time waited, which is preferable, since all events don't show the waits. If TIMED_STATISTICS is false, the events are ordered by the number of waits.
Listing 1 shows a large number of waits related to reading a single block (db file sequential read) as well as waits for latches (latch free). You can see in this listing high waits for some of the writing to datafiles and log files. To identify which of these are major issues, you must narrow down the list by investigating the granular reports within other sections of the report.

Code Listing 1: Report showing waits related to reading a single block

```
TTop 5 Timed Events                                         Avg %Total
~~~~~~~~~~~~~~~~~~                                         wait   Call
Event                                 Waits    Time (s)   (ms)   Time Wait Class
------------------------------ ------------ ----------- ------ ------ ----------
db file sequential read         724,618,076     447,919      1   42.1   User I/O
enq: TM - contention                 76,735     225,484   2938   21.2 Applicatio
CPU time                                        183,751          17.3
enq: TX - row lock contention        44,876     115,222   2568   10.8 Applicatio
db file scattered read            7,333,326      39,885      5    3.7   User I/O
```

This section shows TOP 5 wait events the processes were waiting on during the snapshot time period. These events are helpful during analysis of any database related performance bottlenecks. The
- Waits, column provides information about no. of wait happens.
- Time(s), column provides information about total CPU time in seconds spent on the wait.
- Wait Class, column helps in classifying whether the issue is related to application or infrastructure.

Important wait events and their causes are explained in detail below

Wait Events Information Section

The following section will describe in detail most of the sections provided inside the Wait Section:

- *Time Model Statistics*: Shows where system is spending its time. Generally you want SQL processing time high, parsing and other stuff low. If SQL time>>DB CPU time then probably have IO issues. Example:

```
Time Model Statistics              DB/Inst: IRD_PRD/ird_prd  Snaps: 1664-1716
-> Total time in database user-calls (DB Time): 1064260.6s
-> 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                          1,063,540.7         99.9
DB CPU                                              183,751.3         17.3
PL/SQL execution elapsed time                         9,263.6           .9
RMAN cpu time (backup/restore)                        6,695.7           .6
parse time elapsed                                    2,829.1           .3
hard parse elapsed time                               2,370.2           .2
inbound PL/SQL rpc elapsed time                         663.0           .1
connection management call elapsed time                 404.5           .0
PL/SQL compilation elapsed time                         314.2           .0
hard parse (sharing criteria) elapsed time               59.2           .0
sequence load elapsed time                               53.4           .0
repeated bind elapsed time                               18.3           .0
hard parse (bind mismatch) elapsed time                   1.8           .0
failed parse elapsed time                                 0.0           .0
DB time                                           1,064,260.6          N/A
background elapsed time                              77,860.6          N/A
background cpu time                                  14,180.4          N/A
          -------------------------------------------------------------

Wait Class                          DB/Inst: IRD_PRD/ird_prd  Snaps: 1664-1716
-> s  - second
-> cs - centisecond -     100th of a second
-> ms - millisecond -    1000th of a second
-> us - microsecond - 1000000th of a second
-> ordered by wait time desc, waits desc

                                                                  Avg
                                   %Time       Total Wait    wait     Waits
Wait Class                   Waits  -outs        Time (s)    (ms)      /txn
-------------------- ---------------- ------ ---------------- ------- ---------
User I/O                738,298,915     .0          493,257       1  11,297.1
Application                 125,087   92.7          340,869    2725       1.9
System I/O                5,729,625     .0           42,394       7      87.7
Administrative              754,371     .1           23,653      31      11.5
Other                    98,890,477   95.6            4,859       0   1,513.2
Concurrency                 213,499    7.8              609       3       3.3
Commit                       38,681     .0              185       5       0.6
Configuration                 2,392   70.0               85      35       0.0
Network                   2,359,518     .0                5       0      36.1
          -------------------------------------------------------------
```

- Foreground Wait Events: Foreground wait events are those associated with a session or client process waiting for a resource. These are usually the most important and they are on the top 5 wait events. It has 2 sections: classes and events. Classes are the rolled up sums for waits.

- Background Wait Events: Background wait events are those not associated with a client process. They indicate waits encountered by system and non-system processes. Examples of background system processes are LGWR and DBWR. An example of a non-system background process would be a parallel query slave. Note that it is possible for a wait event to appear in both the foreground and background wait events statistics, for examples the enqueue and latch free events. The idle wait events appear at the bottom of both sections and can generally safely be ignored. Typically these type of events keep record of the time while the client is connected to the database but not requests are being made to the server.

- Service Statistics: A service is a grouping of processes. Users may be grouped in SYS$USER. Application logins (single user) may be grouped with that user name

Resolving Wait Events Section

A critical activity in Database Performance Tuning is Response Time Analysis: this consists of finding out where time is being spent in a database. Response Time Analysis for an Oracle Database is done using the following equation:

Response Time = Service Time + Wait Time

'Service Time' is measured using the statistic 'CPU used by this session'
'Wait Time' is measured by summing up time spent on Wait Events

When presented with such a list of top Wait Events it sometimes becomes easy to simply start dealing with the listed Wait Events and to forget evaluating their impact on overall Response

Time first. In situations where 'Service Time' i.e. CPU usage is much more significant than 'Wait Time', it is very likely that investigating Wait Events will not produce significant savings in 'Response Time'. Therefore, one **should always compare** the time taken by the top wait events to the 'CPU used by this session' and direct the tuning effort to the biggest consumers.

In this section we list the I/O-related Wait Events that occur most often in Oracle databases together with reference notes describing each wait.

Datafile I/O-Related Wait Events:
'db file sequential read' [NOTE:34559.1]
'db file scattered read' [NOTE:34558.1]
'db file parallel read'
'direct path read' [NOTE:50415.1]
'direct path write' [NOTE:50416.1]
'direct path read (lob)'
'direct path write (lob)'

Controlfile I/O-Related Wait Events:
'control file parallel write'
'control file sequential read'
'control file single write'

Redo Logging I/O-Related Wait Events:
'log file parallel write' [NOTE:34583.1]
'log file sync' [NOTE:34592.1]
'log file sequential read'
'log file single write'
'switch logfile command'
'log file switch completion'
'log file switch (clearing log file)'
'log file switch (checkpoint incomplete)'
'log switch/archive'
'log file switch (archiving needed)'

Buffer Cache I/O-Related Wait Events:
'db file parallel write' [NOTE:34416.1]
'db file single write'
'write complete waits'
'free buffer waits'

Common WAIT EVENTS

If you want quick instance wide wait event status, showing which events are the biggest contributors to total wait time, you can use the following query :

```
select event, total_waits,time_waited from V$system_event
  where event NOT IN
  ('pmon timer', 'smon timer', 'rdbms ipc reply', 'parallel deque wait',
  'virtual circuit', '%SQL*Net%', 'client message', 'NULL event')
order by time_waited desc;

EVENT                              TOTAL_WAITS         TIME_WAITED
------------------------          -------------       -------------
db file sequential read               35051309            15965640
latch free                             1373973             1913357
db file scattered read                 2958367             1840810
enqueue                                   2837              370871
buffer busy waits                       444743              252664
log file parallel write                 146221              123435
```

1. DB File Scattered Read.
That generally happens during **a full scan of a table** or **Fast Full Index Scans**. As full table scans are pulled into memory, they rarely fall into contiguous buffers but instead are scattered throughout the buffer cache. A large number here indicates that your table may have missing indexes or your indexes are not used. Although it may be more efficient in your situation to perform a full table scan than an index scan, check to ensure that full table scans are necessary when you see these waits. Try to cache small tables to avoid reading them in over and over again, since a full table scan is put at the cold end of the LRU (Least Recently Used) list. You can use the report to help identify the query in question and fix it.
The init.ora parameter *db_file_multiblock_read_count* specifies the maximum numbers of blocks read in that way. Typically, this parameter should have values of 4-16 independent of the size of the database but with higher values needed with smaller Oracle block sizes. If you have a high wait time for this event, you either need to reduce the cost of I/O, e.g. by getting faster disks or by distributing your I/O load better, or you need to reduce the amount of full table scans by tuning SQL statements. The appearance of the*'db file scattered read'* and *'db file sequential read'*events may not necessarily indicate a problem, as IO is a normal activity on a healthy instance. However, they can indicate problems if any of the following circumstances are true:
• The data-access method is bad (that is, the SQL statements are poorly tuned), resulting in unnecessary or inefficient IO operations
• The IO system is overloaded and performing poorly
• The IO system is under-configured for the load
• IO operations are taking too long

If this Wait Event is a significant portion of Wait Time then a number of approaches are possible:
o Find which SQL statements perform Full Table or Fast Full Index scans and tune them to make sure these scans are necessary and not the result of a suboptimal plan.
- The view V$SQL_PLAN view can help:
For Full Table scans:

```
select sql_text from v$sqltext t, v$sql_plan p
  where t.hash_value=p.hash_value
    and p.operation='TABLE ACCESS'
    and p.options='FULL'
  order by p.hash_value, t.piece;
```

For Fast Full Index scans:

```
select sql_text from v$sqltext t, v$sql_plan p
  where t.hash_value=p.hash_value
    and p.operation='INDEX'
    and p.options='FULL SCAN'
  order by p.hash_value, t.piece;
```

o In cases where such multiblock scans occur from optimal execution plans it is possible to tune the size of multiblock I/Os issued by Oracle by setting the instance parameter DB_FILE_MULTIBLOCK_READ_COUNT so that:
DB_BLOCK_SIZE x DB_FILE_MULTIBLOCK_READ_COUNT = max_io_size of system
Query tuning should be used to optimize online SQL to use indexes.

2. DB File Sequential Read.
This could indicate **poor joining** orders in your SQL or **waiting for writes to TEMP space** generally (direct loads, Parallel DML (PDML) such as parallel updates. It could mean that a lot of index reads/scans are going on. Depending on the problem it may help to tune PGA_AGGREGATE_TARGET and/or DB_CACHE_SIZE.
The sequential read event identifies Oracle reading blocks sequentially, i.e. one after each other.. A large number of waits here could indicate **poor joining orders of tables**, **unselective indexing** or you either need to reduce the cost of I/O, e.g. by getting faster disks or by distributing your I/O load better, or you need to reduce the amount of I/O by increasing the buffer cache or by tuning SQL statements. It is normal for this number to be large for a high-transaction, well-tuned system, but it can indicate problems in some circumstances. You should correlate this wait statistic with other known issues within the report, such as inefficient SQL. Check to ensure that index scans are necessary, and check join orders for multiple table joins. The DB_CACHE_SIZE will also be a determining factor in how often these waits show up. Problematic hash-area joins should show up in the PGA memory, but they're also memory hogs that could cause high wait numbers for sequential reads. They can also show up as direct path read/write waits. These circumstances are usually interrelated. When they occur in conjunction with the appearance of the '*db file scattered read*' and '*db file sequential read*' in the Top 5 Wait Events section, first you should examine the SQL Ordered by Physical Reads section of the report, to see if it might be helpful to tune the statements with the highest resource usage.
It could be because the indexes are fragmented. If that is the case, rebuilding the index will compact it and will produce to visit less blocks.
Then, to determine whether there is a potential I/O bottleneck, examine the OS I/O statistics for corresponding symptoms. Also look at the average time per read in the Tablespace and File I/O sections of the report. If many I/O-related events appear high in the Wait Events list, re-examine the host hardware for disk bottlenecks and check the host-hardware statistics for indications that a disk reconfiguration may be of benefit.

Block reads are fairly inevitable so the aim should be to minimize unnecessary I/O. I/O for sequential reads can be reduced by tuning SQL calls that result in full table scans and using the partitioning option for large tables.

3. Free Buffer Waits. When a session needs a free buffer and cannot find one, it will post the database writer process asking it to flush dirty blocks. Waits in this category may indicate that you need to **increase the DB_BUFFER_CACHE**, if all your SQL is tuned. Free buffer waits could also indicate that unselective SQL is causing data to flood the buffer cache with index blocks, leaving none for this particular statement that is waiting for the system to process. This normally indicates that there is a substantial amount of DML (insert/update/delete) being done and that the Database Writer (DBWR) is not writing quickly enough; the buffer cache could be full of multiple versions of the same buffer, causing great inefficiency. To address this, you may want to consider **accelerating incremental checkpointing**, using more DBWR processes, or increasing the number of physical disks. To investigate if this is an I/O problem, look at the report I/O Statistics. Increase the DB_CACHE_SIZE; shorten the checkpoint; tune the code to get less dirty blocks, faster I/O, use multiple DBWR's.

4. Buffer Busy Waits. A buffer busy wait happens when multiple processes concurrently want to modify the same block in the buffer cache. This typically happens during massive parallel inserts if your tables do not have free lists and it can happen if you have too few rollback segments. **Buffer busy waits should not be greater than 1 percent**. Check the Buffer Wait Statistics section (or V$WAITSTAT) to find out if the wait is on a segment header. If this is the case, increase the freelist groups or increase the pctused to pctfree gap. If the wait is on an undo header, you can address this by adding rollback segments; if it's on an undo block, you need to reduce the data density on the table driving this consistent read or increase the DB_CACHE_SIZE. If the wait is on a data block, you can move data to another block to avoid this hot block, increase the freelists on the table, or use Locally Managed Tablespaces (LMTs). If it's on an index block, you should rebuild the index, partition the index, or use a reverse key index. To prevent buffer busy waits related to data blocks, you can also use a smaller block size: fewer records fall within a single block in this case, so it's not as "hot." When a DML (insert/update/ delete) occurs, Oracle writes information into the block, including all users who are "interested" in the state of the block (Interested Transaction List, ITL). To decrease waits in this area, you can increase the initrans, which will create the space in the block to allow multiple ITL slots. You can also increase the pctfree on the table where this block exists (this writes the ITL information up to the number specified by maxtrans, when there are not enough slots built with the initrans that is specified). Buffer busy waits can be reduced by using reverse-key indexes for busy indexes and by partitioning busy tables.
Buffer Busy Wait on Segment Header – Add freelists (if inserts) or freelist groups (esp. RAC). Use ASSM.
Buffer Busy Wait on Data Block – Separate 'hot' data; potentially use reverse key indexes; fix queries to reduce the blocks popularity, use smaller blocks, I/O, Increase initrans and/or maxtrans (this one's debatable). Reduce records per block
Buffer Busy Wait on Undo Header – Add rollback segments or increase size of segment area (auto undo)
Buffer Busy Wait on Undo block – Commit more (not too much) Larger rollback segments/area. Try to fix the SQL.

5. Latch Free. Latches are low-level queuing mechanisms (they're accurately referred to as mutual exclusion mechanisms) used to protect shared memory structures in the system global area (SGA). Latches are like locks on memory that are very quickly obtained and released. Latches are used to prevent concurrent access to a shared memory structure. If the latch is not available, a latch free miss is recorded. Most latch problems are related to the failure to use bind variables (library cache latch), redo generation issues (redo allocation latch), buffer cache contention issues (cache buffers LRU chain), and hot blocks in the buffer cache (cache buffers chain). There are also latch waits related to bugs; check MetaLink for bug reports if you suspect this is the case. When latch miss ratios are greater than 0.5 percent, you should investigate the issue. If latch free waits are in the Top 5 Wait Events or high in the complete Wait Events list, look at the latch-specific sections of the report to see which latches are contended for.

6. Enqueue. An enqueue is a lock that protects a shared resource. Locks protect shared resources, such as data in a record, to prevent two people from updating the same data at the same time application, e.g. when a select for update is executed.. An enqueue includes a queuing mechanism, which is FIFO (first in, first out). Note that Oracle's latching mechanism is not FIFO. Enqueue waits usually point to the ST enqueue, the HW enqueue, the TX4 enqueue, and the TM enqueue. The ST enqueue is used for space management and allocation for dictionary-managed tablespaces. Use LMTs, or try to preallocate extents or at least make the next extent larger for problematic dictionary-managed tablespaces. HW enqueues are used with the high-water mark of a segment; manually allocating the extents can circumvent this wait. TX4s are the most common enqueue waits. TX4 enqueue waits are usually the result of one of three issues. The first issue is duplicates in a unique index; you need to commit/rollback to free the enqueue. The second is multiple updates to the same bitmap index fragment. Since a single bitmap fragment may contain multiple rowids, you need to issue a commit or rollback to free the enqueue when multiple users are trying to update the same fragment. The third and most likely issue is when **multiple users are updating the same block**. If there are no free ITL slots, a block-level lock could occur. You can easily avoid this scenario by increasing the initrans and/or maxtrans to allow multiple ITL slots and/or by increasing the pctfree on the table. Finally, TM enqueues occur during DML to prevent DDL to the affected object. If you have **foreign keys, be sure to index** them to avoid this general locking issue.
Enqueue - ST Use LMT's or pre-allocate large extents
Enqueue - HW Pre-allocate extents above HW (high water mark.)
Enqueue – TX Increase initrans and/or maxtrans (TX4) on (transaction) the table or index. Fix locking issues if TX6. Bitmap (TX4) & Duplicates in Index (TX4).
Enqueue - TM Index foreign keys; Check application (trans. mgmt.) locking of tables. DML Locks.

7. Log Buffer Space
This wait occurs because you are writing the log buffer faster than LGWR can write it to the redo logs, or because **log switches are too slow**. To address this problem, increase the size of the redo log files, or increase the size of the log buffer, or get faster disks to write to. You might even consider using solid-state disks, for their high speed.

8. Log File Switch
All commit requests are waiting for "logfile switch (archiving needed)" or "logfile switch (chkpt. Incomplete)." Ensure that the archive disk is not full or slow. DBWR may be too slow because of I/O. You may need to add more or larger redo logs, and you may potentially need to add database writers if the DBWR is the problem.

9. Log File Sync
A Log File Sync happens each time a commit (or rollback) takes place. If there are a lot of waits in this area then you may want to examine your application to see if you are committing too frequently (or at least more than you need to). When a user commits or rolls back data, the LGWR flushes the session's redo from the log buffer to the redo logs. The log file sync process must wait for this to successfully complete. To reduce wait events here, try to commit more records (**try to commit a batch of 50 instead of one at a time, use BULKS, , for example**). Put redo logs on a faster disk, or alternate redo logs on different physical disks, to reduce the archiving effect on LGWR. Don't use RAID 5, since it is very slow for applications that write a lot; potentially consider using file system direct I/O or raw devices, which are very fast at writing information. The associated event, 'log buffer parallel write' is used by the redo log writer process, and it will indicate if your actual problem is with the log file I/O. Large wait times for this event can also be caused by having too few CPU resources available for the redolog writer process.

10. Idle Event. There are several idle wait events listed after the output; you can ignore them. Idle events are generally listed at the bottom of each section and include such things as SQL*Net message to/from client and other background-related timings. Idle events are listed in the stats$idle_event table.

11. global cache cr request: (OPS) This wait event shows the amount of time that an instance has waited for a requested data block for a consistent read and the transferred block has not yet arrived at the requesting instance. See Note 157766.1 'Sessions Wait Forever for 'global cache cr request' Wait Event in OPS or RAC'. In some cases the 'global cache cr request' wait event may be perfectly normal if large buffer caches are used and the same data is being accessed concurrently on multiple instances. In a perfectly tuned, non-OPS/RAC database, I/O wait events would be the top wait events but since we are avoiding I/O's with RAC and OPS the 'global cache cr request' wait event often takes the place of I/O wait events.

12. library cache pin Library cache latch contention may be caused by not using bind variables. It is due to excessive parsing of sql statement.

13. CPU time
This is not really a wait event (hence, the new name), but rather the sum of the CPU used by this session, or the amount of CPU time used during the snapshot window. In a heavily loaded system, if the CPU time event is the biggest event, that could point to some CPU-intensive processing (for example, forcing the use of an index when a full scan should have been used), which could be the cause of the bottleneck. When CPU Other is a significant component of total Response Time the next step is to find the SQL statements that access the most blocks. Block accesses are also known as Buffer Gets and Logical I/Os. The report lists such SQL statements in section SQL ordered by Gets.

14. DB File Parallel Read This Wait Event is used when Oracle performs in parallel reads from multiple datafiles to non-contiguous buffers in memory (PGA or Buffer Cache). This is done during recovery operations or when buffer prefetching is being used as an optimization i.e. instead of performing multiple single-block reads. If this wait is an important component of Wait Time, follow the same guidelines as 'db file sequential read'.

15. PX qref latch Can often mean that the Producers are producing data quicker than the Consumers can consume it. Maybe we could increase parallel_execution_message_size to try to eliminate some of these waits or we might decrease the degree of parallelism. If the system workload is high consider to decrease the degree of parallelism. If you have DEFAULT parallelism on your object you can decrease the value of PARALLEL_THREADS_PER_CPU. Have in mind DEFAULT degree = PARALLEL_THREADS_PER_CPU * #CPU's

16. Log File Parallel Writes. Log file parallel write waits occur when waiting for writes of REDO records to the REDO log files to complete. The wait occurs in log writer (LGWR) as part of normal activity of copying records from the REDO log buffer to the current online log. The actual wait time is the time taken for all the outstanding I/O requests to complete. Even though the writes may be issued in parallel, LGWR needs to wait for the last I/O to be on disk before the parallel write is considered complete. Hence the wait time depends on the time it takes the OS to complete all requests.
Waits for log file parallel writes can be identified by looking at the "Top 5 Timed Events" or "Wait Events" section of the report.
Log file parallel write waits can be reduced by moving log files to the faster disks and/or separate disks where there will be less contention.

17. SQL*Net more data to client
This means the instance is sending a lot of data to the client. You can decrease this time by having the client bring back less data. Maybe the application doesn't need to bring back as much data as it is.

18. SQL*Net message to client
The "SQL*Net message to client" Oracle metric indicates the server (foreground process) is sending a message to the client, and it can be used to identify throughput issues over a network, especially distributed databases with slow database links. The SQL*Net more data to client event happens when Oracle writes multiple data buffers (sized per SDU) in a single logical network call.

19. enq: TX - row lock contention:
Oracle keeps data consistency with the help of locking mechanism. When a particular row is being modified by the process, either through Update/ Delete or Insert operation, oracle tries to acquire lock on that row. Only when the process has acquired lock the process can modify the row otherwise the process waits for the lock. This wait situation triggers this event. The lock is released whenever a COMMIT is issued by the process which has acquired lock for the row. Once the lock is released, processes waiting on this event can acquire lock on the row and perform DML operation

SQL Information Section

The SQL that is stored in the shared pool SQL area (Library cache) is reported in this section in different ways:
. SQL ordered by Buffer Gets
. SQL ordered by Physical Reads
. SQL ordered by Executions
. SQL ordered by Parse Calls

- SQL ordered by Gets:
This section reports the contents of the SQL area ordered by the number of buffer gets and can be used to identify the **most CPU Heavy SQL.**
- Many DBAs feel that if the data is already contained within the buffer cache the query should be efficient. This could not be further from the truth. Retrieving more data than needed, even from the buffer cache, requires CPU cycles and interprocess IO. Generally speaking, the cost of physical I/O is not 10,000 times more expensive. It actually is in the neighborhood of 67 times and actually almost zero if the data is stored in the UNIX buffer cache.
- The statements of interest are those with a large number of gets per execution especially if the number of executions is high.
- High buffer gets generally correlates with heavy CPU usage

- SQL ordered by Reads:
This section reports the contents of the SQL area ordered by the number of reads from the data files and can be used to identify SQL causing IO bottlenecks which consume the following resources.
- CPU time needed to fetch unnecessary data.
- File IO resources to fetch unnecessary data.
- Buffer resources to hold unnecessary data.
- Additional CPU time to process the query once the data is retrieved into the buffer.

- SQL ordered by Executions:
This section reports the contents of the SQL area ordered by the number of query executions. It is primarily useful in identifying the most frequently used SQL within the database so that they can be monitored for efficiency. Generally speaking, a small performance increase on a frequently used query provides greater gains than a moderate performance increase on an infrequently used query. Possible reasons for high Reads per Exec are use of unselective indexes require large numbers of blocks to be fetched where such blocks are not cached well in the buffer cache, index fragmentation, large Clustering Factor in index etc.

- SQL ordered by Parse Calls:
This section shows the number of times a statement was parsed as compared to the number of times it was executed. One to one parse/executions may indicate that:
- Bind variables are not being used.
  The shared pool may be too small and the parse is not being retained long enough for multiple executions.
- cursor_sharing is set to exact (this should NOT be changed without considerable testing on the part of the client).

Generate Execution Plan for given SQL statement

If you have identified one or more problematic SQL statement, you may want to check the execution plan. Remember the "Old Hash Value" from the report above (1279400914), then execute the scrip to generate the execution plan.

```
sqlplus perfstat/perfstat
SQL> @?/rdbms/admin/sprepsql.sql
Enter the Hash Value, in this example: 1279400914

SQL Text
~~~~~~~~
create table test as select * from all_objects

Known Optimizer Plan(s) for this Old Hash Value
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shows all known Optimizer Plans for this database instance, and the Snap Id's
they were first found in the shared pool.  A Plan Hash Value will appear
multiple times if the cost has changed
-> ordered by Snap Id

  First        First          Plan
 Snap Id     Snap Time     Hash Value        Cost
--------- -------------- ----------- ----------
        6 14 Nov 04 11:26   1386862634        52

Plans in shared pool between Begin and End Snap Ids
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shows the Execution Plans found in the shared pool between the begin and end
snapshots specified.  The values for Rows, Bytes and Cost shown below are those
which existed at the time the first-ever snapshot captured this plan - these
values often change over time, and so may not be indicative of current values
-> Rows indicates Cardinality, PHV is Plan Hash Value
-> ordered by Plan Hash Value


--------------------------------------------------------------------------------
| Operation                      | PHV/Object Name     |  Rows | Bytes|   Cost |
--------------------------------------------------------------------------------
|CREATE TABLE STATEMENT          |----- 1386862634 ----|       |      |     52 |
|LOAD AS SELECT                  |                     |       |      |        |
| VIEW                           |                     |     1K|  216K|     44 |
|  FILTER                        |                     |       |      |        |
|   HASH JOIN                    |                     |     1K|  151K|     38 |
|    TABLE ACCESS FULL           |USER$                |    29 |  464 |      2 |
|    TABLE ACCESS FULL           |OBJ$                 |     3K|  249K|     35 |
|   TABLE ACCESS BY INDEX ROWID  |IND$                 |     1 |    7 |      2 |
|    INDEX UNIQUE SCAN           |I_IND1               |     1 |      |      1 |
|   NESTED LOOPS                 |                     |     5 |  115 |     16 |
|    INDEX RANGE SCAN            |I_OBJAUTH1           |     1 |   10 |      2 |
|    FIXED TABLE FULL            |X$KZSRO              |     5 |   65 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   FIXED TABLE FULL             |X$KZSPR              |     1 |   26 |     14 |
|   VIEW                         |                     |     1 |   13 |      2 |
|    FAST DUAL                   |                     |     1 |      |      2 |
--------------------------------------------------------------------------------
```

Instance Activity Stats Section

The statistics section shows the overall database statistics. These are the statistics that the summary information is derived from. A list of the statistics maintained by the RDBMS kernel can be found in Appendix C of the Oracle Reference manual for the version being utilized.

Instance Activity Stats for DB: PHS2 Instance: phs2 Snaps: 100 -104

```
Statistic                                   Total  per Second    per Trans
-------------------------------- ---------------- ----------- ------------
CPU used by this session                   84,161        23.4      3,825.5
CPU used when call started                196,346        54.5      8,924.8
CR blocks created                             709         0.2         32.2
DBWR buffers scanned                            0         0.0          0.0
DBWR checkpoint buffers written               245         0.1         11.1
DBWR checkpoints                               33         0.0          1.5
DBWR cross instance writes                     93         0.0          4.2
DBWR free buffers found                         0         0.0          0.0
....

Statistic                                     Total    per Second  per Trans
-------------------------------- ------------------ ------------- ----------
branch node splits                            7,162           0.1        0.0
consistent gets                  12,931,850,777     152,858.8    3,969.5
current blocks converted for CR              75,709           0.9        0.0
db block changes                        343,632,442       4,061.9      105.5
db block gets                           390,323,754       4,613.8      119.8
hot buffers moved to head of LRU        197,262,394       2,331.7       60.6
leaf node 90-10 splits                       26,429           0.3        0.0
leaf node splits                            840,436           9.9        0.3
logons cumulative                            21,369           0.3        0.0
physical reads                          504,643,275       5,965.1      154.9
physical writes                          49,724,268         587.8       15.3
session logical reads                13,322,170,917     157,472.5    4,089.4
sorts (disk)                                  4,132           0.1        0.0
sorts (memory)                            7,938,085          93.8        2.4
sorts (rows)                            906,207,041      10,711.7      278.2
table fetch continued row                25,506,365         301.5        7.8
table scans (long tables)                       111           0.0        0.0
table scans (short tables)                1,543,085          18.2        0.5
```

Instance Activity Terminology

Statistic	Description
Session Logical Reads	All reads cached in memory. Includes both consistent gets and also the db block gets.
Consistent Gets	These are the reads of a block that are in the cache. They are NOT to be confused with consistent read (cr) version of a block in the buffer cache (usually the current version is read).
Db block gets	These are block gotten to be changed. MUST be the CURRENT block and not a cr block.
Db block changes	These are the db block gets (above) that were actually changed.
Physical Reads	Blocks not read from the cache. Either from disk, disk cache or O/S cache; there are also physical reads direct which bypass cache using Parallel Query (not in hit ratios).

Of particular interest are the following statistics.
- CPU USED BY THIS SESSION, PARSE TIME CPU or RECURSIVE CPU USAGE: These numbers are useful to diagnose CPU saturation on the system (usually a query tuning issue). The formula to calculate the CPU usage breakdown is:
Service (CPU) Time = other CPU + parse time CPU
Other CPU = "CPU used by this session" - parse time CPU
Some releases do not correctly store this data and can show huge numbers.

recursive cpu usage = This component can be high if large amounts of PL/SQL are being processed. It is outside the scope of this document to go into detail with this, but you will need to identify your complete set of PL/SQL, including stored procedures, finding the ones with the highest CPU load and optimize these. If most work done in PL/SQL is procedural processing (rather than executing SQL), a high recursive cpu usage can actually indicate a potential tuning effort.

parse time cpu= Parsing SQL statements is a heavy operation, that should be avoided by reusing SQL statements as much as possible. In precompiler programs, unnecessary parting of implicit SQL statements can be avoided by increasing the cursor cache (MAXOPENCURSORS parameter) and by reusing cursors. In programs using Oracle Call Interface, you need to write the code, so that it re-executes (in stead of reparse) cursors with frequently executed SQL statements. The v$sql view contains PARSE_CALLS and EXECUTIONS columns, that can be used to identify SQL, that is parsed often or is only executed once per parse.

other cpu= The source of other cpu is primarily handling of buffers in the buffer cache. It can generally be assumed, that the CPU time spent by a SQL statement is approximately proportional to the number of buffer gets for that SQL statements, hence, you should identify and sort SQL statements by buffer gets in v$sql. In your report, look at the part ‘SQL ordered by Gets for DB’. Start tuning SQL statements from the top of this list. In Oracle, the v$sql view contain a column, CPU_TIME, which directly shows the cpu time associated with executing the SQL statement.

- DBWR BUFFERS SCANNED: the number of buffers looked at when scanning the lru portion of the buffer cache for dirty buffers to make clean. Divide by "dbwr lru scans" to find the average number of buffers scanned. This count includes both dirty and clean buffers. The average buffers scanned may be different from the average scan depth due to write batches filling up before a scan is complete. Note that this includes scans for reasons other than make free buffer requests.
- DBWR CHECKPOINTS: the number of checkpoints messages that were sent to DBWR and not necessarily the total number of actual checkpoints that took place. During a checkpoint there is a slight decrease in performance since data blocks are being written to disk and that causes I/O. If the number of checkpoints is reduced, the performance of normal database operations improve but recovery after instance failure is slower.
- DBWR TIMEOUTS: the number of timeouts when DBWR had been idle since the last timeout. These are the times that DBWR looked for buffers to idle write.
- DIRTY BUFFERS INSPECTED: the number of times a foreground encountered a dirty buffer which had aged out through the lru queue, when foreground is looking for a buffer to reuse. This should be zero if DBWR is keeping up with foregrounds.
- FREE BUFFER INSPECTED: the number of buffers skipped over from the end of the LRU queue in order to find a free buffer. The difference between this and "dirty buffers inspected" is the number of buffers that could not be used because they were busy or needed to be written after rapid aging out. They may have a user, a waiter, or being read/written.
- RECURSIVE CALLS: Recursive calls occur because of cache misses and segment extension. In general if recursive calls is greater than 30 per process, the data dictionary cache should be optimized and segments should be rebuilt with storage clauses that have few large extents. Segments include tables, indexes, rollback segment, and temporary segments.
NOTE: PL/SQL can generate extra recursive calls which may be unavoidable.
- **REDO BUFFER ALLOCATION RETRIES:** total number of retries necessary to allocate space in the redo buffer. Retries are needed because either the redo writer has gotten behind, or because an event (such as log switch) is occurring
- REDO LOG SPACE REQUESTS: indicates how many times a user process waited for space in the redo log buffer. Try increasing the init.ora parameter LOG_BUFFER so that zero Redo Log Space Requests are made.
- REDO WASTAGE: Number of bytes "wasted" because redo blocks needed to be written before they are completely full. Early writing may be needed to commit transactions, to be able to write a database buffer, or to switch logs
- SUMMED DIRTY QUEUE LENGTH: the sum of the lruw queue length after every write request completes. (divide by write requests to get average queue length after write completion)
- TABLE FETCH BY ROWID: the number of rows that were accessed by a rowid. This includes rows that were accessed using an index and rows that were accessed using the statement where rowid = 'xxxxxxxx.xxxx.xxxx'.
- TABLE FETCH BY CONTINUED ROW: indicates the number of rows that are chained to another block. In some cases (i.e. tables with long columns) this is unavoidable, but the ANALYZE table command should be used to further investigate the chaining, and where possible, should be eliminated by rebuilding the table.
- Table Scans (long tables) is the total number of full table scans performed on tables with more than 5 database blocks. If the number of full table scans is high the application should be tuned to effectively use Oracle indexes. Indexes, if they exist, should be used on long tables if less than 10-20% (depending on parameter settings and CPU count) of the rows from the table are returned. If this is not the case, check the db_file_multiblock_read_count parameter setting. It may be too high. You may also need to tweak optimizer_index_caching and optimizer_index_cost_adj.
- Table Scans (short tables) is the number of full table scans performed on tables with less than 5 database blocks. It is optimal to perform full table scans on short tables rather than using indexes.

Tablespace I/O Stats Section

IO Activity Input/Output (IO) statistics for the instance are listed in the following sections/formats:

- Tablespace I/O Stats for DB: Ordered by total IO per tablespace.
- File I/O Stats for DB: Ordered alphabetically by tablespace, filename.

If the statistic "Buffer Waits" for a tablespace is greater than 1000, you may want to consider tablespace reorganization in order to spread tables within it across another tablespaces.

Note that Oracle considers average read times of greater than 20 ms unacceptable. If a datafile consistently has average read times of 20 ms or greater then:
- The queries against the contents of the owning tablespace should be examined and tuned so that less data is retrieved.
- If the tablespace contains indexes, another option is to compress the indexes so that they require less space and hence, less IO.
- The contents of that datafile should be redistributed across several disks/logical volumes to more easily accommodate the load.
- If the disk layout seems optimal, check the disk controller layout. It may be that the datafiles need to be distributed across more disk sets.

```
Tablespace IO Stats               DB/Inst: IRD_PRD/ird_prd  Snaps: 1664-1716
-> 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)
-------------- ------- ------ ------- ------------ -------- ---------- ------
PPP_INDX
   566,599,780   3,027    0.3     1.0      494,209        3      7,386    0.9
UNDOTBS1
   157,107,113     839    1.9     1.0      185,640        1  1,789,665    2.4
PPP_DATA
     8,546,632      46    6.1    13.8    9,587,570       51      3,381    2.6
TEMP
     3,142,350      17    3.9    16.1      268,866        1      3,744    0.5
SYSTEM
       627,187       3    2.8     1.1       24,346        0      1,397    4.5
SYSAUX
       102,691       1    9.0     1.1       46,352        0          0    0.0
PPP_DATA_ARCH
       122,859       1   54.3    15.1            0        0          0    0.0
QUEST
           856       0  136.5     1.5            0        0          0    0.0
PPP_INDX_ARCH
             1       0 ######     1.0            0        0          0    0.0
          -------------------------------------------------------------

File IO Stats                     DB/Inst: IRD_PRD/ird_prd  Snaps: 1664-1716
-> 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)
-------------- ------- ------ ------- ------------ -------- ---------- ------
PPP_DATA                 /oradb3p/d03/oradata/ird_prd/ppp_data01.dbf
     2,837,164      15    6.0    14.1    4,557,614       24         44    3.2
PPP_DATA                 /oradb3p/d07/oradata/ird_prd/ppp_data03.dbf
     2,869,543      15    5.9    13.7      999,436        5      3,303    2.6
PPP_DATA                 /oradb3p/d10/oradata/ird_prd/ppp_data02.dbf
     2,839,925      15    6.5    13.8    4,030,520       22         34    1.8
PPP_DATA_ARCH            /oradb3p/d03/oradata/ird_prd/ppp_data_arch01.dbf
        39,428       0   27.2    15.1            0        0          0    0.0
PPP_DATA_ARCH            /oradb3p/d07/oradata/ird_prd/ppp_data_arch03.dbf
        41,938       0   62.7    15.1            0        0          0    0.0
PPP_DATA_ARCH            /oradb3p/d11/oradata/ird_prd/ppp_data_arch02.dbf
        41,493       0   71.4    15.1            0        0          0    0.0
PPP_INDX                 /oradb3p/d04/oradata/ird_prd/ppp_indx01.dbf
   289,555,801   1,547    0.3     1.0      246,684        1      2,608    0.9
PPP_INDX                 /oradb3p/d09/oradata/ird_prd/ppp_indx02.dbf
   277,043,979   1,480    0.3     1.0      247,525        1      4,778    0.9
PPP_INDX_ARCH            /oradb3p/d04/oradata/ird_prd/ppp_indx_arch01.dbf
             1       0 ######     1.0            0        0          0    0.0
QUEST                    /oradb3p/d02/oradata/ird_prd/quest01.dbf
           856       0  136.5     1.5            0        0          0    0.0
SYSAUX                   /oradb3p/d02/oradata/ird_prd/sysaux01.dbf
       102,691       1    9.0     1.1       46,352        0          0    0.0
SYSTEM                   /oradb3p/d02/oradata/ird_prd/system01.dbf
       627,187       3    2.8     1.1       24,346        0      1,397    4.5
TEMP                     /oradb3p/d02/oradata/ird_prd/temp01.dbf
     3,142,350      17    3.9    16.1      268,866        1      3,744    0.5
UNDOTBS1                 /oradb3p/d02/oradata/ird_prd/undotbs01.dbf
    52,997,517     283    1.9     1.0       79,116        0    675,685    2.5
UNDOTBS1                 /oradb3p/d11/oradata/ird_prd/undotbs02.dbf
    53,960,561     288    1.8     1.0       48,901        0    504,274    2.1
UNDOTBS1                 /oradb3p/d11/oradata/ird_prd/undotbs03.dbf
     2,081,951      11    2.1     1.0        9,909        0      9,842    4.5
UNDOTBS1                 /oradb3p/d12/oradata/ird_prd/undotbs04.dbf
    48,067,084     257    1.8     1.0       47,714        0    599,864    2.5
          -------------------------------------------------------------
```

PGA Statistics

Shows the following information:
- Process Global Area –User processes
- Sort, Hash, Bitmap, Global Temporary operations
- Only Sorts really tracked
- Investigate if temporary IO high, but no disk sorts indicated
- V$SORT_USAGE good source
- V$SQL_WORKAREA_ACTIVE good for sorts and hashes
- Rule of thumb for size based on histogram:
    - High Optimal*20=PGA_AGGREGATE_TARGET

PGA Aggregate Summary
- 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.
- Low values mean we need a higher PGA_AGGREGATE_TARGET setting

```
PGA Aggr Summary                  DB/Inst: IRD_PRD/ird_prd  Snaps: 1664-1716
-> 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
--------------- ------------------ --------------------------
           69.6            237,123                    103,569
          -------------------------------------------------------------

PGA Aggr Target Stats              DB/Inst: IRD_PRD/ird_prd  Snaps: 1664-1716
-> B: Begin snap   E: End snap (rows dentified with B or E contain data
   which is absolute i.e. not diffed over the interval)
-> Auto PGA Target - actual workarea memory target
-> W/A PGA Used    - amount of memory used for all Workareas (manual + auto)
-> %PGA W/A Mem    - percentage of PGA memory allocated to workareas
-> %Auto W/A Mem   - percentage of workarea memory controlled by Auto Mem Mgmt
-> %Man W/A Mem    - percentage of workarea memory under manual control

                                                    %PGA  %Auto   %Man
    PGA Aggr   Auto PGA   PGA Mem    W/A PGA     W/A    W/A    W/A Global Mem
   Target(M)  Target(M)  Alloc(M)    Used(M)     Mem    Mem    Mem   Bound(K)
- ---------- ---------- ---------- ---------- ------ ------ ------ ----------
B        598        425      258.3        1.0     .4     .0  100.0    102,400
E        598        412      267.3        1.0     .4     .0  100.0    102,400
          -------------------------------------------------------------

PGA Aggr Target Histogram           DB/Inst: IRD_PRD/ird_prd  Snaps: 1664-1716
-> Optimal Executions are purely in-memory operations

  Low     High
Optimal Optimal    Total Execs  Optimal Execs 1-Pass Execs M-Pass Execs
------- ------- -------------- -------------- ------------ ------------
     2K      4K      1,021,048      1,021,048            0            0
    64K    128K          3,618          3,618            0            0
   128K    256K            764            764            0            0
   256K    512K          1,805          1,805            0            0
   512K   1024K          8,044          8,044            0            0
     1M      2M          4,796          4,796            0            0
     2M      4M          2,226          1,605          621            0
     4M      8M          5,822          2,466        3,356            0
     8M     16M          3,851          2,660        1,191            0
    16M     32M          2,953          1,029        1,924            0
    32M     64M          1,513            889          624            0
    64M    128M              4              4            0            0
          -------------------------------------------------------------

PGA Memory Advisory                      DB/Inst: IRD_PRD/ird_prd  Snap: 1716
-> When using Auto Memory Mgmt, minimally choose a pga_aggregate_target value
   where Estd PGA Overalloc Count is 0

                                       Estd Extra    Estd PGA   Estd PGA
PGA Target    Size           W/A MB   W/A MB Read/      Cache  Overalloc
  Est (MB)   Factr        Processed Written to Disk     Hit %      Count
---------- ------- ---------------- ---------------- -------- ----------
        75     0.1        340,986.8        638,004.2     35.0      4,495
       150     0.3        340,986.8        248,995.6     58.0        323
       299     0.5        340,986.8         96,445.0     78.0          0
       449     0.8        340,986.8         72,431.0     82.0          0
       598     1.0        340,986.8         70,410.7     83.0          0
       718     1.2        340,986.8              0.0    100.0          0
       837     1.4        340,986.8              0.0    100.0          0
       957     1.6        340,986.8              0.0    100.0          0
     1,076     1.8        340,986.8              0.0    100.0          0
     1,196     2.0        340,986.8              0.0    100.0          0
     1,794     3.0        340,986.8              0.0    100.0          0
     2,392     4.0        340,986.8              0.0    100.0          0
     3,588     6.0        340,986.8              0.0    100.0          0
     4,784     8.0        340,986.8              0.0    100.0          0
          -------------------------------------------------------------
```

PGA Memory Advisory

PGA Target Est (MB)	Size Factr	W/A MB Processed	Estd Extra W/A MB Read/ Written to Disk	Estd PGA Cache Hit %	Estd PGA Overalloc Count
384	0.13	812,978.60	178,424.72	82.00	3,834
768	0.25	812,978.60	172,475.22	82.00	3,763
1,536	0.50	812,978.60	30,048.28	96.00	0
2,304	0.75	812,978.60	25,473.80	97.00	0
3,072	1.00	812,978.60	25,473.80	97.00	0
3,686	1.20	812,978.60	10,796.70	99.00	0
4,301	1.40	812,978.60	10,796.70	99.00	0
4,915	1.60	812,978.60	10,796.70	99.00	0
5,530	1.80	812,978.60	10,796.70	99.00	0
6,144	2.00	812,978.60	10,796.70	99.00	0
9,216	3.00	812,978.60	10,796.70	99.00	0
12,288	4.00	812,978.60	10,796.70	99.00	0
18,432	6.00	812,978.60	10,796.70	99.00	0
24,576	8.00	812,978.60	10,796.70	99.00	0

Similar to Buffer Pool Advisory, the statistic provides information on how the increase or decrease in PGA memory will cause increase or decrease in Estd PGA Cahce Hit %.

Starting point here is “Size Factor” = 1.0. This gives current memory allocation for PGA. In this example 3072 MB is being allocated to PGA. With this allocation the Estd PGA Cahce Hit % is 97, which is good. Hence even if we increase PGA to 3686 MB we will get 2% increase in Estd PGA Cahce Hit %. Hence it won’t be advisable to increase PGA further.

Shared Pool Advisory Section

Use this section to evaluate your shared pool size parameter.
Some times not very useful. Depend more on the shrink and grow sections or V$SGA_RESIZE_OPS

Other Advisories
- SGA Target
    - Helps for SGA_TARGET settings
- Streams Pool
    - Only if streams are used, if you are getting spills, indicates pool is too small
- Java Pool
    - Only if you are using internal Java, similar to the PL/SQL area in the library caches

```
Shared Pool Advisory                     DB/Inst: IRD_PRD/ird_prd  Snap: 1716
-> 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
   Size(M) Factr      (M)      Mem Obj     (s)  Factr     (s)  Factr    Obj Hits
---------- ----- -------- ------------ ------- ------ ------- ------ -----------
       108    .9       19        1,875 #######     .8 #######   55.8  29,872,126
       124   1.0       34        2,676 #######    1.0   4,131    1.0  29,920,851
       140   1.1       49        3,362 #######    1.1       1     .0  29,963,026
       156   1.3       63        4,230 #######    1.2       1     .0  29,993,364
       172   1.4       77        4,816 #######    1.3       1     .0  30,012,627
       188   1.5       92        5,339 #######    1.3       1     .0  30,024,685
       204   1.6      107        5,943 #######    1.3       1     .0  30,031,986
       220   1.8      122        6,667 #######    1.3       1     .0  30,036,402
       236   1.9      137        7,230 #######    1.3       1     .0  30,039,491
       252   2.0      152        8,080 #######    1.3       1     .0  30,041,771
          -------------------------------------------------------------

SGA Target Advisory                      DB/Inst: IRD_PRD/ird_prd  Snap: 1716

SGA Target   SGA Size       Est DB     Est Physical
  Size (M)     Factor     Time (s)            Reads
---------- ---------- ------------ ----------------
       250        0.5    1,908,027    1,996,326,970
       375        0.8    1,557,766    1,460,001,663
       500        1.0    1,195,413      898,517,855
       625        1.3      921,319      473,698,613
       750        1.5      722,518      149,243,816
       875        1.8      722,399      137,832,639
     1,000        2.0      722,403      137,832,639
          -------------------------------------------------------------

Streams Pool Advisory                    DB/Inst: IRD_PRD/ird_prd  Snap: 1716

  Size for      Size   Est Spill   Est Spill Est Unspill Est Unspill
  Est (MB)    Factor       Count    Time (s)       Count    Time (s)
---------- --------- ----------- ----------- ----------- -----------
         4       0.5           0           0           0           0
         8       1.0           0           0           0           0
        12       1.5           0           0           0           0
        16       2.0           0           0           0           0
        20       2.5           0           0           0           0
        24       3.0           0           0           0           0
        28       3.5           0           0           0           0
        32       4.0           0           0           0           0
        36       4.5           0           0           0           0
        40       5.0           0           0           0           0
        44       5.5           0           0           0           0
        48       6.0           0           0           0           0
        52       6.5           0           0           0           0
        56       7.0           0           0           0           0
        60       7.5           0           0           0           0
        64       8.0           0           0           0           0
        68       8.5           0           0           0           0
        72       9.0           0           0           0           0
        76       9.5           0           0           0           0
        80      10.0           0           0           0           0
          -------------------------------------------------------------

Java Pool Advisory                       DB/Inst: IRD_PRD/ird_prd  Snap: 1716

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

Buffer Wait Statistics             DB/Inst: IRD_PRD/ird_prd  Snaps: 1664-1716
-> ordered by wait time desc, waits desc

Class                    Waits Total Wait Time (s)  Avg Time (ms)
------------------ ----------- ------------------- -------------
undo block           1,782,880               4,311             2
data block              12,027                  21             2
segment header           3,763                   2             1
undo header              6,770                   2             0
1st level bmb              112                   0             1
2nd level bmb                1                   0             0
          -------------------------------------------------------------
```

Shared Pool Advisory

Shared Pool Size(M)	SP 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
252	0.90	28	6,586	6,921,060	0.93	614,333	10.64	58,238,179
280	1.00	55	7,434	7,477,642	1.00	57,751	1.00	58,323,146
308	1.10	82	8,379	8,020,189	1.07	1	0.00	58,403,382
336	1.20	109	9,352	8,461,984	1.13	1	0.00	58,459,714
364	1.30	136	10,362	8,790,828	1.18	1	0.00	58,496,766
392	1.40	163	11,293	9,040,418	1.21	1	0.00	58,523,514
420	1.50	190	12,185	9,242,662	1.24	1	0.00	58,544,538
448	1.60	217	13,193	9,414,811	1.26	1	0.00	58,561,861
476	1.70	244	14,377	9,567,814	1.28	1	0.00	58,576,296
504	1.80	271	15,293	9,703,054	1.30	1	0.00	58,588,125
532	1.90	298	16,196	9,814,500	1.31	1	0.00	58,597,966
560	2.00	325	17,328	9,907,118	1.32	1	0.00	58,606,158

Similar to Buffer Pool Advisory and PGA, the statistic provides information on how the increase or decrease in Shared pool memory will cause increase or decrease in Estd LC Load Time (s).

Starting point here is “SP Size Factor” = 1.0. This gives current memory allocation for shared pool. In this example 280 MB is being allocated to shared pool. With this allocation the Estd LC Load Time (s) is 57,751. If we increase the shared pool size to 308 then Estd LC Load Time (s) will come down to value 1. Hence shared pool should be set to 308 MB.

SGA Target Advisory

SGA Target Size (M)	SGA Size Factor	Est DB Time (s)	Est Physical Reads
512	0.50	3,062,724	1,038,371,906
768	0.75	2,689,285	844,082,787
1,024	1.00	2,417,105	676,023,376
1,280	1.25	2,246,714	570,766,536
1,536	1.50	2,084,044	470,241,860
1,792	1.75	1,885,603	347,611,220
2,048	2.00	1,885,608	347,611,220

The statistic provides information on how the increase or decrease in SGA memory will cause decrease or increase in Estd Physical Reads.

Starting point here is “SGA Size Factor” = 1.0. This gives current memory allocation for SGA. In this example 1024 MB is being allocated to SGA. With this allocation the Estd Physical Reads is 844,082,787. If we increase the SGA size by 1024 MB i.e. to 2048 MB then Estd Physical Reads will come down by 328,412,156. Since there is 50% reduction in Estd Physical Reads, the SGA should be increased to 2048 MB.

Buffer cache Activity Information

The buffer statistics are comprised of two sections:

- *Buffer Pool Statistics:* This section can have multiple entries if multiple buffer pools are allocated. A baseline of the database's buffer pool statistics should be available to compare with the current report buffer pool statistics. A change in that pattern unaccounted for by a change in workload should be a cause for concern. Also check the **Buffer Pool Advisory** to identify if increasing that parameter (db_cache_size) would help to reduce Physical Reads.

- *Buffer Wait Statistics:* This section shows a breakdown of each type of object waited for. This section follows the Instance Recovery Stats for DB.

```
Buffer Pool Statistics             DB/Inst: IRD_PRD/ird_prd  Snaps: 1664-1716
-> 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       42,442   91  9,161,057,638  837,987,530  25,228,808    0    0  1,805,588
          -------------------------------------------------------------

Instance Recovery Stats             DB/Inst: IRD_PRD/ird_prd  Snaps: 1664-1716
-> 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    89    58       1130    204197    322560     322560    435941          N/A
E    89    15        305      5409     18073     322560     18073          N/A
          -------------------------------------------------------------
```

Buffer Pool Advisory

P	Size for Est (M)	Size Factor	Buffers for Estimate	Est Phys Read Factor	Estimated Physical Reads
D	72	0.10	8,622	4.35	2,937,669,401
D	144	0.20	17,244	2.03	1,371,604,952
D	216	0.30	25,866	1.54	1,038,346,123
D	288	0.40	34,488	1.42	959,213,827
D	360	0.49	43,110	1.33	899,213,101
D	432	0.59	51,732	1.25	844,100,667
D	504	0.69	60,354	1.17	793,922,143
D	576	0.79	68,976	1.11	752,192,096
D	648	0.89	77,598	1.06	716,041,579
D	720	0.99	86,220	1.01	680,211,989
D	728	1.00	87,178	1.00	676,024,586
D	792	1.09	94,842	0.95	642,521,274
D	864	1.19	103,464	0.89	603,505,502
D	936	1.29	112,086	0.84	570,790,229
D	1,008	1.38	120,708	0.80	543,401,299
D	1,080	1.48	129,330	0.77	518,133,473
D	1,152	1.58	137,952	0.73	491,055,525
D	1,224	1.68	146,574	0.70	470,256,432
D	1,296	1.78	155,196	0.66	447,139,441
D	1,368	1.88	163,818	0.59	396,093,338
D	1,440	1.98	172,440	0.51	347,627,804

The section provides estimates on, how the increase or descrease of buffer cache size will casue decrease or increase in physical reads. This information is just an estimated data and not an actual data.
Starting point here is “Size Factor” = 1.0. This gives current memory allocation for Buffer Cache. In this example, 728 MB is being allocated to buffer cache. With this setting the estimated amount of Physical Reads are 676,024,586. In case we increase the memory allocation for buffer cache to say 1440 MB (“Size Factor” = 1.98) then estimated physical reads will be 347,627,804. This means by allocating additional 712 MB for Buffer Cache, total estimated physical reads will come down by 328,396,782.
On the other hand, by reducing Buffer Cache to say 216 MB (“Size Factor” = 0.30) estimated physical reads increase to 1,038,346,123.
The statistics acts as an input to DBA in order to tune the Buffer Cache memory.

Buffer Waits Statistics
- In the old days we just got “buffer waits”
- Now they break it down for you
    - Data block –blocksharing (block too big)
    - Undo header –Insufficient number of undo segments
    - File header block –freelist, freelistgroup issues
    - 1st level bmb–ASSM bitmap issues
    - Segment header –freelist, freelistgroup issues.
    - 2nd level bmb. –ASSM bitmap issues
- Look at the ITL waits section for more guidance

Enqueue Activity

An enqueue is simply a locking mechanism. This section is very useful and must be used when the wait event "enqueue" is listed in the "Top 5 timed events".

```
Enqueue Activity                   DB/Inst: IRD_PRD/ird_prd  Snaps: 1664-1716
-> 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)
------------ ------------ ----------- ----------- ------------ --------------
RO-Multiple Object Reuse (fast object reuse)
      24,066       24,066           0       2,663          122          45.97
KO-Multiple Object Checkpoint (fast object checkpoint)
       6,066        6,066           0         652           44          68.02
CF-Controlfile Transaction
     147,515      147,462          53         174           17          96.04
TX-Transaction (row lock contention)
       5,577        5,575           0       5,525           11           2.08
TQ-Queue table enqueue (DDL contention)
          27           27           0           2            3       1,576.00
TX-Transaction (index contention)
         273          273           0           2            3       1,261.50
PS-PX Process Reservation
      83,250       74,358       8,892       3,703            2            .43
TC-Tablespace Checkpoint
           9            9           0           3            2         511.33
TQ-Queue table enqueue (TM contention)
           3            3           0           2            1         469.00
UL-User-defined
          13           13           0           2            0         163.50
          -------------------------------------------------------------
```

The action to take depends on the lock type that is causing the most problems. The most common lock waits are generally for:

- TX (Transaction Lock): Generally due to application concurrency mechanisms, or table setup issues. The TX lock is acquired when a transaction initiates its first change and is held until the transaction does a COMMIT or ROLLBACK. It is used mainly as a queuing mechanism so that other resources can wait for a transaction to complete.

- TM (DML enqueue): Generally due to application issues, particularly if foreign key constraints have not been indexed. This lock/enqueue is acquired when performing an insert, update, or delete on a parent or child table.

- ST (Space management enqueue): Usually caused by too much space management occurring. For example: create table as select on large tables on busy instances, small extent sizes, lots of sorting, etc. These enqueues are caused if a lot of space management activity is occurring on the database (such as small extent size, several sortings occurring on the disk).

V$SESSION_WAIT and V$LOCK give more data about enqueues
- The P1, P2 and P3 values tell what the enqueuemay have been waiting on
- For BF we get node#, parallelizer#, and bloom#

```
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%';
```

Undo Segment Information

Undo information is provided in the following sections:
- Undo Segment Summary
- Undo Segment Stats

The examples below show typical performance problem related to Undo (rollback) segments:

- Undo Segment Summary for DB

```
Undo Segment Summary for DB: S901  Instance: S901  Snaps: 2 -3
-> Undo segment block stats:
-> uS - unexpired Stolen,   uR - unexpired Released,   uU - unexpired reUsed
-> eS - expired   Stolen,   eR - expired   Released,   eU - expired   reUsed

Undo           Undo        Num  Max Qry     Max Tx Snapshot Out of uS/uR/uU/
 TS#         Blocks      Trans  Len (s)   Concurcy  Too Old  Space eS/eR/eU
---- -------------- ---------- -------- ---------- -------- ------ ------------
   1         20,284      1,964        8         12        0      0 0/0/0/0/0/0
```

The description of the view V$UNDOSTAT in the Oracle Database Reference guide provides some insight as to the columns definitions. Should the client encounter SMU problems, monitoring this view every few minutes would provide more useful information.

- Undo Segment Stats for DB

```
Undo Segment Stats for DB: S901  Instance: S901  Snaps: 2 -3
-> ordered by Time desc

                     Undo      Num Max Qry   Max Tx  Snap   Out of uS/uR/uU/
End Time           Blocks    Trans Len (s)    Concy Too Old  Space eS/eR/eU
------------ ------------ -------- ------- -------- ------- ------ -------------
12-Mar 16:11       18,723    1,756       8       12       0      0 0/0/0/0/0/0
12-Mar 16:01        1,561      208       3       12       0      0 0/0/0/0/0/0
```

This section provides a more detailed look at the statistics in the previous section by listing the information as it appears in each snapshot.

Use of UNDO_RETENTION can potentially increase the size of the undo segment for a given period of time, so the retention period should not be arbitrarily set too high. The UNDO tablespace still must be sized appropriately. The following calculation can be used to determine how much space a given undo segment will consume given a set value of UNDO_RETENTION.
Undo Segment Space Required = (undo_retention_time * undo_blocks_per_seconds)

As an example, an UNDO_RETENTION of 5 minutes (default) with 50 undo blocks/second (8k blocksize) will generate:

Undo Segment Space Required = (300 seconds * 50 blocks/ seconds * 8K/block) = 120 M

The retention information (transaction commit time) is stored in every transaction table block and each extent map block. When the retention period has expired, SMON will be signaled to perform undo reclaims, done by scanning each transaction table for undo timestamps and deleting the information from the undo segment extent map. Only during extreme space constraint issues will retention period not be obeyed.

Latch Information

Latch information is provided in the following three sections:
. Latch Activity
. Latch Sleep breakdown
. Latch Miss Sources

This information should be checked whenever the "latch free" wait event or other latch wait events experience long waits. This section is particularly useful for determining latch contention on an instance. Latch contention generally indicates resource contention and supports indications of it in other sections. Latch contention is indicated by a Pct Miss of greater than 1.0% or a relatively high value in Avg Sleeps/Miss. While each latch can indicate contention on some resource, the more common latches to watch are:

cache buffer chain= The cache buffer chain latch protects the hash chain of cache buffers, and is used for each access to cache buffers. Contention for this latch can often only be reduced by reducing the amount of access to cache buffers. Using the X$BH fixed table can identify if some hash chains have many buffers associated with them. Often, a single hot block, such as an index root block, can cause contention for this latch. Contention on this latch confirms a hot block issue.
shared pool= The shared pool latch is heavily used during parsing, in particular during hard parse. If your application is written so that it generally uses literals in stead of bind variables, you will have high contention on this latch. Contention on this latch in conjunction with reloads in the SQL Area of the library cache section indicates that the shared pool is too small. You can set the cursor_sharing parameter in init.ora to the value ‘force’ to reduce the hard parsing and reduce some of the contention for the shared pool latch. Applications that are coded to only parse once per cursor and execute multiple times will almost completely avoid contention for the shared pool latch.

- Literal SQL is being used. See Note 62143.1 'Understanding and Tuning the Shared Pool for an excellent discussion of this topic.
- The parameter session_cached_cursors might need to be set. See enhancement bug 1589185 for details.

library cache= The library cache latch is heavily used during both hard and soft parsing. If you have high contention for this latch, your application should be modified to avoid parsing if at all possible. Setting the cursor_sharing parameter in init.ora to the value ‘force’ provides some reduction in the library cache latch needs for hard parses, and setting the session_cached_cursors sufficiently high provides some reduction in the library cache latch needs for repeated soft parsing within a single session. There is minor contention for this latch involved in executing SQL statements, which can be reduced further by setting cursor_space_for_time=true, if the application is properly written to parse statements once and execute multiple times.
row cache= The row cache latch protects the data dictionary information, such as information about tables and columns. During hard parsing, this latch is used extensively. The cursor_sharing parameter can be used to completely avoid the row cache latch lookup during parsing.
cache buffer lru chain= The buffer cache has a set of chains of LRU block, each protected by one of these latches. Contention for this latch can often be reduced by increasing the db_block_lru_latches parameter or by reducing the amount of access to cachebuffers.

Library Cache Statistics

This section of the report shows information about the different sub-areas activity in the library cache.

```
Library Cache Activity for DB: S901  Instance: S901  Snaps: 2 -3
->"Pct Misses"  should be very low
                         Get  Pct        Pin        Pct               Invali-
Namespace           Requests  Miss     Requests     Miss     Reloads  dations
--------------- ------------ ------ -------------- ------ ---------- --------
BODY                 310,879    0.0        310,880    0.0          0        0
CLUSTER                1,009    0.3          1,007    0.6          0        0
INDEX                 14,713    0.2         17,591    0.3          0        0
```

```
SQL AREA          14,184,204    0.1    313,089,592    1.9  5,793,355   38,421
TABLE/PROCEDURE   46,190,602    0.0     91,843,902    0.0        825        0
TRIGGER              148,809    0.0        148,809    0.0          2        0
```

Values in **Pct Misses** or **Reloads** in the SQL Area, Tables/Procedures or Trigger rows indicate that the shared pool may be too small.
Values in Invalidations in the SQL Area indicate that a table definition changed while a query was being run against it or a PL/SQL package being used was recompiled.

Segment Statistics

Segments by Logical Reads

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Logical Reads	%Total
RRA_OWNER	RRA_INDEX	RRA_REPBAL_IX_03		INDEX	817,720,480	73.37
RRA_OWNER	RRA_DATA	TRRA_BALANCE_STATUS		TABLE	68,558,256	6.15
RRA_OWNER	RRA_DATA	TRRA_TRANSACTION_STATUS		TABLE	58,728,272	5.27
RRA_OWNER	RRA_INDEX	RRA_MSGOUT_IX_05		INDEX	17,506,640	1.57
RRA_OWNER	RRA_DATA	TRRA_SENT_REGISTER_FILENAMES		TABLE	6,542,848	0.59

The statistic displays segment details based on logical reads happened. Data displayed is sorted on “Logical Reads” column in descending order. It provides information about segments for which more logical reads are happening. Most of these SQLs can be found under section SQL Statistics -> SQL ordered by Gets.

Segments by Physical Reads

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Physical Reads	%Total
RRA_OWNER	RRA_DATA	TRRA_BALANCE_STATUS		TABLE	2,112,621	32.09
RRA_OWNER	RRA_INDEX	RRA_PROCSTATUS_IX_03		INDEX	1,649,590	25.05
RRA_OWNER	RRA_INDEX	RRA_BALSTAT_IX_03		INDEX	776,458	11.79
RRA_OWNER	RRA_DATA	TRRA_TRANSACTIONS		TABLE	381,302	5.79
RRA_OWNER	RRA_INDEX	RRA_PROCSTATUS_IX_01		INDEX	259,326	3.94

The statistic displays segment details based on physical reads happened. Data displayed is sorted on “Physical Reads” column in descending order. It provides information about segments for which more physical reads are happening.

Queries using these segments should be analysed to check whether any FTS is happening on these segments. In case FTS is happening then proper indexes should be created to eliminate FTS. Most of these SQLs can be found under section SQL Statistics -> SQL ordered by Reads.

Segments by Row Lock Waits

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Row Lock Waits	% of Capture
RRA_OWNER	RRA_DATA	RRA_SRF_IX_04		INDEX	6,907	20.18
RRA_OWNER	RRA_INDEX	RRA_PROCSTATUS_IX_03		INDEX	3,918	11.45
RRA_OWNER	RRA_INDEX	RRA_REPTRN_PK		INDEX	3,118	9.11
RRA_OWNER	RRA_DATA	TRRA_BALANCE_STATUS		TABLE	1,750	5.11
RRA_OWNER	RRA_INDEX	RRA_PROCSTATUS_IX_02		INDEX	1,178	3.44

The statistic displays segment details based on total “Row lock waits” which happened during snapshot period. Data displayed is sorted on “Row Lock Waits” column in descending order. It provides information about segments for which more database locking is happening.

DML statements using these segments should be analysed further to check the possibility of reducing concurrency due to row locking.

Segments by ITL Waits

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	ITL Waits	% of Capture
RRA_OWNER	RRA_INDEX	RRA_MSGBLOBS_IX_01		INDEX	23	27.06
RRA_OWNER	RRA_INDEX	RRA_TRN_IX_08		INDEX	15	17.65
RRA_OWNER	RRA_INDEX	RRA_INT_CLOB_IX_01		INDEX	10	11.76
RRA_OWNER	RRA_INDEX	RRA_TRN_IX_05		INDEX	10	11.76
RRA_OWNER	RRA_INDEX	RRA_TRN_IX_10		INDEX	8	9.41

Whenver a transaction modifies segment block, it first add transaction id in the Internal Transaction List table of the block. Size of this table is a block level configurable parameter. Based on the value of this parameter those many ITL slots are created in each block.

ITL wait happens in case total trasactions trying to update same block at the same time are greater than the ITL parameter value.

Total waits happening in the example are very less, 23 is the Max one. Hence it is not recommended to increase the ITL parameter value.

Segments by Buffer Busy Waits

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Buffer Busy Waits	% of Capture
RRA_OWNER	RRA_INDEX	RRA_REPTRN_PK		INDEX	4,577	26.69
RRA_OWNER	RRA_INDEX	RRA_REPBAL_IX_03		INDEX	1,824	10.64
RRA_OWNER	RRA_INDEX	RRA_PROCSTATUS_IX_03		INDEX	1,715	10.00
RRA_OWNER	RRA_INDEX	RRA_MSGINB_PK		INDEX	827	4.82
RRA_OWNER	RRA_INDEX	RRA_PROCSTATUS_PK		INDEX	696	4.06

Buffer busy waits happen when more than one transaction tries to access same block at the same time. In this scenario, the first transaction which acquires lock on the block will able to proceed further whereas other transaction waits for the first transaction to finish.

If there are more than one instances of a process continuously polling database by executing same SQL (to check if there are any records available for processing), same block is read concurrently by all the instances of a process and this result in Buffer Busy wait event.

Library Cache Activity

Namespace	Get Requests	Pct Miss	Pin Requests	Pct Miss	Reloads	Invali- dations
BODY	17	35.29	1,260,957	0.38	1,875	0

CLUSTER	14,335	1.10	41,182	0.49	43	0
INDEX	113	68.14	340	37.94	35	0
SQL AREA	846,225	3.54	14,397,753	26.05	3,143,888	3,028,117
TABLE/PROCEDURE	314,825	0.64	3,419,828	13.37	256,865	0
TRIGGER	4	0.00	200,436	0.69	1,085	0

“Pct Misses” should be very low. In the example Pct Misses are above 10 for some of the library cache components. This indicates that the shared pool is not sufficiently sized. In case AMM (Automatic Memory Management) is used then the DBA can increase the SGA component. In case AMM is not used then increase the SHARED_POOL memory component.

Retrieve SQL and Execution Plan from AWR Snapshots

This is a simple script that can help you to collect SQL statements executed since yesterday (configurable) that contains a specific value in its sentence.

```
col parsed format a6
col sql_text format a40
set lines 200
set pages 300
select sql_text, parsing_schema_name as parsed, elapsed_time_delta/1000/1000 as elapsed_sec, stat.snap_id,
       to_char(snap.end_interval_time,'dd.mm hh24:mi:ss') as snaptime, txt.sql_id
from dba_hist_sqlstat stat, dba_hist_sqltext txt, dba_hist_snapshot snap
where stat.sql_id=txt.sql_id
  and stat.snap_id=snap.snap_id
  and snap.begin_interval_time >= sysdate-1
  and lower(sql_text) like '%&sql_test%'
  and parsing_schema_name not in ('SYS','SYSMAN','MDSYS','WKSYS')
order by elapsed_time_delta asc;
```

This will show something like:

```
Enter value for sql_test: delete
old   7:   and lower(sql_text) like '%&sql_test%'
new   7:   and lower(sql_text) like '%delete%'
SQL_TEXT                                 PARSED ELAPSED_SEC    SNAP_ID SNAPTIME
      SQL_ID
---------------------------------------- ------ ----------- ---------- ---------
----- -------------
DELETE FROM WWV_FLOW_FILE_OBJECTS$ WHERE APEX_0     .484942        688 23.08 16:
00:54 8rpn8jtjnuu73
 SECURITY_GROUP_ID = 0                   30200
```

Then with that sql_id, we can retrieve the execution plan from the snapshots:

```
select plan_table_output from table (dbms_xplan.display_awr('&sqlid'));

Enter value for sqlid: 8rpn8jtjnuu73
old   1: select plan_table_output from table (dbms_xplan.display_awr('&sqlid'))
new   1: select plan_table_output from table (dbms_xplan.display_awr('8rpn8jtjnuu73'))

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

SQL_ID 8rpn8jtjnuu73
--------------------
DELETE FROM WWV_FLOW_FILE_OBJECTS$ WHERE SECURITY_GROUP_ID = 0
Plan hash value: 358826532

-------------------------------------------------------------------------------------
| Id  | Operation         | Name                      | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------
|   0 | DELETE STATEMENT  |                           |       |       |     1 (100)|
|   1 |  DELETE           | WWV_FLOW_FILE_OBJECTS$    |       |       |            |
|   2 |   INDEX RANGE SCAN| WWV_FLOW_FILES_SGID_FK_IDX |     1 |   202 |     0   (0)|
-------------------------------------------------------------------------------------
```

Get Data from ASH

If you need to quickly check a performance problem on your DB, ASH is great here
We sample the Wait-Events of active sessions every second into the ASH-Buffer.
It is accessed most comfortable with the Enterprise Manager GUI from the Performance Page (Button ASH Report there).
Or with little effort from the command line like this:

```
-----------------------------------------
-- Top 10 CPU consumers in last 5 minutes
-----------------------------------------
select *
from (select session_id, session_serial#, count(*)
         from v$active_session_history
         where session_state= 'ON CPU'
           and sample_time > sysdate - interval '5' minute
         group by session_id, session_serial#
         order by count(*) desc
     )
where rownum <= 10;


--------------------------------------------
-- Top 10 waiting sessions in last 5 minutes
--------------------------------------------
select *
from (select session_id, session_serial#,count(*)
        from v$active_session_history
        where session_state='WAITING'
          and sample_time >  sysdate - interval '5' minute
        group by session_id, session_serial#
        order by count(*) desc
      )
where rownum <= 10;



-- These 2 queries should spot the most incriminating sessions of the last 5 minutes.
-- But who is that and what SQL was running?


--------------------
-- Who is that SID?
--------------------
set lines 200
col username for a10
col osuser for a10
col machine for a10
col program for a10
col resource_consumer_group for a10
col client_info for a10

select  serial#, username, osuser, machine, program, resource_consumer_group, client_info
from v$session
where sid = &sid;
```

```
-------------------------
-- What did that SID do?
-------------------------
select distinct sql_id, session_serial#
from v$active_session_history
where sample_time >  sysdate - interval '5' minute
and session_id = &sid;


----------------------------------------------
-- Retrieve the SQL from the Library Cache:
----------------------------------------------
col sql_text for a80
select sql_text from v$sql where sql_id='&sqlid';
```

Removing and Disabling AWR Information

There may be times when a DBA might desire to disable AWR (Automatic Workload Repository).
One reason might be to avoid licensing issues, because AWR isn't part of the standard or even enterprise database – as it requires the optional (extra cost) Oracle Enterprise Manager (OEM) Diagnostic pack. So even though your database automatically collects AWR data every sixty minutes and retains it for a week – you cannot legally use the Oracle supplied PL/SQL packages (i.e. DBMS_WORKLOAD_REPOSITORY), the OEM screens for AWR, ADMM and ASH, or even the AWR data dictionary views (i.e. DBA_HIST_*) if you're not licensed.
If you query the DBA_HIST_* data dictionary views, you better have purchased the OEM Diagnostic pack! For those of you who prefer to directly access the SYS data dictionary tables – that means don't even select from tables with names like WRM$*, WRH$* or WRI$*!
So assuming that you prefer to disable AWR so as not to accidentally (or purposefully) violate your Oracle licensing agreement, here are some ways to disable AWR for a given database (you'll need to do one of these to every database you manage):
Many Ways to Disable AWR:

1. Download Meta-Link script dbms_awr.plb, compile this package, then execute the PL/SQL package dbms_awr.disable_awr() [Metalink Note 436386.1].
2. Set your init.ora parameter `STATISTICS_LEVEL = BASIC`
3. Execute the Oracle provided PL/SQL package: `dbms_workload_repository.modify_snapshot_settings(interval=>0)`
4. Execute the Oracle provided PL/SQL package: `dbms_scheduler.disable('GATHER_STATS_JOB')`
5. You can use Toad for #3: Main Menu->Database->Monitor->ADDM/AWR Reports screen, choose the Snapshot Management tab, set the interval to all zeroes, and then press the green checkmark in upper left corner to commit the change.
6. You can use Toad for #4: Main Menu->Schema Browser, choose the Sched. Job tab and disable the GATHER_STATS_JOB job.
7. You can use OEM for #4: Main Menu->Workload->Automatic Workload Repository, select the "Edit" button and then select the last radio group item labeled: Turn off Snapshot Collection, finally press OK
8. You can use OEM for #5: Main Menu->Scheduler->Jobs, select the data grid row for GATHER_STATS_JOB, choose the disable drop-down action, then finally press OK
9. Create your own database creation scripts (i.e. do not use DBCA) and make sure not to run the CATAWRTB.sql script [Note – Oracle upgrade process may undo this]
10. Run the `$ORACLE_HOME\rdbms\admin\catnoawr.sql` script to drop the AWR Repository tables [Note – Oracle upgrade process may undo this]

If you want to rebuild the AWR Repository Tables later, you need to perform the following:
- Execute (again) the script $ORACLE_HOME/rdbms/admin/catnoawr.sql
- Execute the script $ORACLE_HOME/rdbms/admin/catawrtb.sql
- Bounce the database.
- On re-start of the database instance, the AWR tables will be populated with the required data.

Links with AWR Analyzer

Excel Performance Analyzer (Perfsheet v2.0)
http://www.oraperf.com
http://www.txmemsys.com/statspack-reg.htm (Statspack Analyzer)
http://www.dbapool.com/dbanalyzer.php (Analyze your AWR or Statspack)
http://www.softpedia.com/get/Internet/Servers/Database-Utils/spReporter.shtml (Download Tool to Analyze AWR or Statspack Reports)
http://www.ondatafine.com/ (web based application. It processes plain-text statspack or AWR)
http://www.spviewer.com/index.html (STATSPACK and AWR Viewer software)
Scripts: http://www.evdbt.com/tools.htm