Monitoring and Diagnostics

--- without OracleEnterpriseManager

Hemant K Chitale
http://hemantoracledba.blogspot.com
@HemantKChitale
SoS 14-July2015

`whoami`

- Been a DBA / SME for more than 20years
- Began with V6. Worked on various Unix flavours, Linux, Windows. OPS and RAC
- Used OEM only for a couple of years in 2006-2007 (yes, the last time I used OEM was mid-2007)
- I will make these slides (with Notes) available via my blog and slideshare in a few days

I am not OEM savvy !
The notes sections in this presentation also provide links to some of my blog posts

Privileges

- What privileges do you need ?
  - CREATE SESSION
  - SELECT_CATALOG_ROLE **
  - EXECUTE ON DBMS_XPLAN (present)
  - EXECUTE ON DBMS_WORKLOAD_REPOSITORY (optional)
- What privileges do you NOT need ?
  - The DBA Role
  - SELECT ANY DICTIONARY
  - SELECT ANY TABLE

** Note : Direct Privileges required if using PLSQL

Only a small set of privileges are really required.
Also look at the OEM_MONITOR role that has ANALYZE ANY , SELECT ANY, SELECT ANY DICTIONARY and ADVISOR privileges, none of which I use (although ADVISOR may be useful).

For a short note on the difference between SELECT ANY DICTIONARY and SELECT_CATALOG_ROLE see http://hemantoracledba.blogspot.com/2014/02/the-difference-between-select-any.html

SELECT ANY can be useful if you need to query data (e.g. To identify distribution patterns and skew) in a schema – but the owner might prefer to grant you SELECT privileges on only a subset of tables --- that sort of requirement comes in handy when doing Performance Tuning, which is outside of the scope of this presentation.

If you plan to use PLSQL (e.g. to schedule jobs to collect this monitoring information), you *will* need direct privileges on the underlying views. For example, on V_$SESSION, V_$SQL etc.

There are many System Privileges and Object level privileges that can be granted to Junior DBAs, Performance Analysts etc without having to grant the DBA role.

Creating the User

```
SYSTEM>create user diag_mon identified by diag_mon;
User created.
SYSTEM>grant create session to diag_mon;
Grant succeeded.
SYSTEM>grant select_catalog_role to diag_mon;
Grant succeeded.
SYSTEM>select privilege, grantee from dba_tab_privs
  2  where table_name = 'DBMS_XPLAN';
PRIVILEGE                                GRANTEE
---------------------------------------- ------------------------------
EXECUTE                                  PUBLIC
SYSTEM>connect sys/oracle as sysdba
Connected.
SYS>grant execute on dbms_workload_repository to diag_mon;
Grant succeeded.
SYS>
SYS>connect diag_mon/diag_mon
Connected.
DIAG_MON>
```

These are the privileges I need. I don't use ADVISORs but do use AWR.

Some of the Views we'll look at here

- V$SESSION
- V$ACTIVE_SESSION_HISTORY (and DBA_HIST_ACTIVE_SESS_HISTORY)
- V$SQL and V$SQLSTATS
- V$SESSION_LONGOPS

See
select table_name, privilege from dba_tab_privs
where grantee = 'SELECT_CATALOG_ROLE'

The fewer V$ joins you need to make the better. Also, some V$ views are preferable over others. Remember that Oracle does not provide the same read-consistency for V$ (and X$!) views as for permanent tables and views. Joining V$ views (to each other or to DBA_% views) does not guarantee read consistency across the join.

Active / Long Running Sessions

- V$SESSION :
  - STATUS
  - LAST_CALL_ET
  - SEQ#, EVENT, STATE, SECONDS_IN_WAIT
  - Interpreting correlation between STATE, EVENT and SECONDS_IN_WAIT
  - LAST_CALL_ET and PLSQL and SQL
- V$ACTIVE_SESSION_HISTORY

Columns that have been retrieved from V$SESSION_WAIT are now available in V$SESSION. V$SQL preferred over V$SQLAREA because the latter does an aggregation (across all Child Cursors).
When I join V$SESSION to V$SQL, I join on both SQL_ID and CHILD_NUMBER.

Similarly, I prefer V$SEGSTAT over V$SEGMENT_STATISTICS – the former is faster.

If you don't have the Diagnostic Pack for V$ASH, you could sample V$SESSION quickly with custom code.

Active Running Session

```
DIAG_MON>l
  1  select username, sid, serial#, sql_id, status, last_call_et, seq#, event, state,
     seconds_in_wait
  2  from v$session
  3  where status = 'ACTIVE'
  4  and last_call_et > 4
  5* and type != 'BACKGROUND'
DIAG_MON>/

no rows selected

DIAG_MON>/

USERNAME        SID    SERIAL# SQL_ID        STATUS   LAST_CALL_ET       SEQ#
-------- ---------- ---------- ------------- -------- ------------ ----------
EVENT                                                                 STATE
    SECONDS_IN_WAIT
---------------------------------------------------------------- ------------------- --
    -------------
HEMANT            9          3 gr4zw7dd73r1x ACTIVE              5      16400
direct path read                                                      WAITED SHORT TIME
    0
```

What I intend to show is how to interpret the STATE. WAITED_SHORT_TIME means that it is *not* currently in a Wait. The EVENT is the last wait, not current.
So, although the session is ACTIVE, it is not in a Wait. It is most likely on CPU.

```
00:13:53 DIAG_MON>l
  1  select sql_id, status, last_call_et, seq#, event, state, seconds_in_wait
  2  from v$session
  3* where sid=195
00:14:35 DIAG_MON>/

SQL_ID        STATUS   LAST_CALL_ET       SEQ# EVENT                          STATE
    SECONDS_IN_WAIT
------------- -------- ------------ ---------- ---------------------------- ------------------- -----
    ----------
addfgac99vuxw ACTIVE              8      50735 db file scattered read       WAITED SHORT TIME
    0

00:14:36 DIAG_MON>
00:14:46 DIAG_MON>/

SQL_ID        STATUS   LAST_CALL_ET       SEQ# EVENT                          STATE
    SECONDS_IN_WAIT
------------- -------- ------------ ---------- ---------------------------- ------------------- -----
    ----------
addfgac99vuxw ACTIVE             19      53225 free buffer waits            WAITING
    0

00:14:47 DIAG_MON>
00:15:21 DIAG_MON>/

SQL_ID        STATUS   LAST_CALL_ET       SEQ# EVENT                          STATE
    SECONDS_IN_WAIT
------------- -------- ------------ ---------- ---------------------------- ------------------- -----
    ----------
addfgac99vuxw ACTIVE             53      60431 db file scattered read       WAITING
    0

00:15:21 DIAG_MON>
```

Note the change to the WAITING on “free buffer waits”. That *is* the CURRENT Wait as at the time of the snapshot. (Similarly, the “db file scattered read” wait after that). So, at the bottom of this slide, the session’s current SQL has been active for 55 seconds and is currently in a multiblock read wait.

```
00:15:45 DIAG_MON>/

SQL_ID        STATUS   LAST_CALL_ET       SEQ# EVENT                         STATE
   SECONDS_IN_WAIT
------------- -------- ------------ ---------- ---------------------------- -----------
   -------- ---------------
cdctxzddqb067 ACTIVE             78        518 db file scattered read       WAITING
   0

00:15:46 DIAG_MON>
00:16:23 DIAG_MON>/

SQL_ID        STATUS   LAST_CALL_ET       SEQ# EVENT                         STATE
   SECONDS_IN_WAIT
------------- -------- ------------ ---------- ---------------------------- -----------
   -------- ---------------
330q95smuwnv9 ACTIVE            116      11364 db file sequential read      WAITED
   SHORT TIME                 0

00:16:24 DIAG_MON>
```

Note how the *current* wait status can keep changing. Have you noted SEQ# incrementing ? That indicates that the Wait Events *are* changing.

SQL called from PLSQL

- Interpreting SQL_ID, STATUS, LAST_CALL_ET
- For example, for a DBMS_STATS.GATHER_TABLE_STATS call, you will see the SQL_ID keep changing from that of the DBMS_STATS call to one of the child SQL calls that actually reads table / column / index data.
- The LAST_CALL_ET would be the time since the *parent* DBMS_STATS call began !
- This also applies to PLSQL that calls SQL

Recursive SQLs called by DBMS_STATS (or any PLSQL procedure) are are at a depth level below. The top level is dep=0, the succeeding levels are 1 and beyond.

SQL called fromPSQL

```
HEMANT>exec dbms_stats.gather_table_stats('','LARGE_TABLE',method_opt=>'FOR ALL COLUMNS SIZE 250');

22:58:09 DIAG_MON>-- running monitoring query
22:58:10 DIAG_MON>/

       SID    SERIAL# SQL_ID        STATUS   LAST_CALL_ET       SEQ#
---------- ---------- ------------- -------- ------------ ----------
EVENT                                                           STATE               SECONDS_IN_WAIT
---------------------------------------------------------------- ------------------- ---------------
       135         65 98bht7550nwkr ACTIVE             11       2652
db file scattered read                                          WAITING                           0
22:58:14 DIAG_MON>/

       SID    SERIAL# SQL_ID        STATUS   LAST_CALL_ET       SEQ#
---------- ---------- ------------- -------- ------------ ----------
EVENT                                                           STATE               SECONDS_IN_WAIT
---------------------------------------------------------------- ------------------- ---------------
       135         65 52p6wm9z96c00 ACTIVE             21       7278
db file sequential read                                         WAITING                           0

22:58:23 DIAG_MON>/

       SID    SERIAL# SQL_ID        STATUS   LAST_CALL_ET       SEQ#
---------- ---------- ------------- -------- ------------ ----------
EVENT                                                           STATE               SECONDS_IN_WAIT
---------------------------------------------------------------- ------------------- ---------------

       135         65 8y9qnq9m089b6 ACTIVE             25      20565
PX Deq: Execute Reply                                           WAITING                           0
```

The GATHER_TABLE_STATS ran for 25seconds. The LAST_CALL_ET against SID=135 (HEMANT's session) was incremented across all the SQL calls, even though the SQL calls (at lower depths of 1 and below) were changing. Therefore, in this case, the LAST_CALL_ET is not for the SQL that was executing at that instant but for the calling PLSQL – the DBMS_STATS.GATHER_TABLE_STATS call. So, when running PLSQL beware that LAST_CALL_ET may not reflect the current SQL !

(Note the SQL_ID 8y9… statement – it was waiting on PQ slaves (I haven't shown those PQ slave sessions, but they id exist)

SQL called from PLSQL

```
DIAG_MON>l
  1  select sql_id, sql_text from v$sql
  2* where sql_id in ('98bht7550nwkr','52p6wm9z96c00','8y9qnq9m089b6','53nh88nc8mx1b') order by 1
DIAG_MON>/

SQL_ID
-------------
SQL_TEXT
--------------------------------------------------------------------------------------------------------------
    ----
52p6wm9z96c00
SELECT  topology   FROM  SDO_TOPO_METADATA_TABLE a, TABLE(a.Topo_Geometry_Layers) b   WHERE b.owner =
    :owner AND b.table_name = :tab

53nh88nc8mx1b
BEGIN dbms_stats.gather_table_stats('','LARGE_TABLE',method_opt=>'FOR ALL COLUMNS SIZE 250'); END;

8y9qnq9m089b6
select /*+  parallel_index(t, "LARGE_TABLE_OWNER",4)  dbms_stats cursor_sharing_exact
    use_weak_name_resl dynamic_sampling(0) no_monitoring no_substrb_pad  no_expand
    index_ffs(t,"LARGE_TABLE_OWNER") */ count(*) as nrw,count(distinct
    sys_op_lbid(114855,'L',t.rowid))


98bht7550nwkr
/* SQL Analyze(0) */ select /*+  full(t)    no_parallel(t) no_parallel_index(t) dbms_stats
    cursor_sharing_exact use_weak_name_resl dynamic_sampling(0) no_monitoring no_substrb_pad
    */to_char(count("OWNER")),to_char(substrb(dump(min("OWNER"),16,0,32),1,120)),to_cha
```

This slide is not necessarily part of this presentation. It is just to demonstrate that a PLSQL (the DBMS_STATS.GATHER_TABLE_STATS in this case) can call multiple SQLs, at different recursive depth levels. (From dep=0 to dep=4 in this case). So, when you are monitoring V$SQL_ID in V$SESSION, you might get a statement that is at a much lowe depth.

Where's the Catch ?

```
23:40:33 DIAG_MON>l
  1  select username, sid, serial#, sql_id, status, last_call_et, seq#, event, state, seconds_in_wait
  2  from v$session
  3  where username = 'HEMANT'
  4* and type != 'BACKGROUND'
23:40:35 DIAG_MON>/

USERNAME        SID    SERIAL# SQL_ID        STATUS   LAST_CALL_ET       SEQ#
-------- ---------- ---------- ------------- -------- ------------ ----------
EVENT                                                            STATE               SECONDS_IN_WAIT
---------------------------------------------------------------- ------------------- ---------------
HEMANT            9          3 gr4zw7dd73r1x INACTIVE          492      17821
SQL*Net message from client                                      WAITING                         492


23:40:35 DIAG_MON>
```

Note here that this session is WAITING on a message from client. Is it Idle ? Should the DBA ignore this ? Let's look at the next slide.

Where's the Catch ?

```
23:40:35 DIAG_MON>
23:41:04 DIAG_MON>/

USERNAME        SID    SERIAL# SQL_ID        STATUS   LAST_CALL_ET       SEQ#
-------- ---------- ---------- ------------- -------- ------------ ----------
EVENT                                                    STATE               SECONDS_IN_WAIT
---------------------------------------------------------------- ------------------- ---------------
HEMANT            9          3 9t7c0jy3v19pn INACTIVE           0      24807
SQL*Net message from client                                      WAITING                           0

23:41:07 DIAG_MON>/

USERNAME        SID    SERIAL# SQL_ID        STATUS   LAST_CALL_ET       SEQ#
-------- ---------- ---------- ------------- -------- ------------ ----------
EVENT                                                    STATE               SECONDS_IN_WAIT
---------------------------------------------------------------- ------------------- ---------------
HEMANT            9          3 9t7c0jy3v19pn INACTIVE           0      32663
SQL*Net message from client                                      WAITING                           0

23:41:22 DIAG_MON>/

USERNAME        SID    SERIAL# SQL_ID        STATUS   LAST_CALL_ET       SEQ#
-------- ---------- ---------- ------------- -------- ------------ ----------
EVENT                                                    STATE               SECONDS_IN_WAIT
---------------------------------------------------------------- ------------------- ---------------
HEMANT            9          3 9t7c0jy3v19pn INACTIVE           0      64336
SQL*Net message from client                                      WAITING                           0
```

Note SEQ# incrementing ? Even though the status is INACTIVE and the wait is "SQL*Net message from client" (In theory an "idle wait") ?

What is really happening here is that the user or application server is running a query that is retrieving a large number of rows. Between every ARRAY FETCH, the server process has to wait for the client to acknowledge having received the rows – that is the "SQL*Net message from client" wait.

Here the user says that his query is ACTIVE. And he is right ! The user is complaining that the query is long running but the actual Database Time the DBA sees is very low. Most of the time is being spent sending rows and waiting for the acknowledgement.

Where's the Catch ?

```
23:42:21 DIAG_MON>/

USERNAME        SID    SERIAL# SQL_ID        STATUS   LAST_CALL_ET       SEQ#
-------- ---------- ---------- ------------- -------- ------------ ----------
EVENT                                                              STATE               SECONDS_IN_WAIT
------------------------------------------------------------------ ------------------- ---------------
HEMANT            9          3 9t7c0jy3v19pn INACTIVE             0      54223
SQL*Net message from client                                        WAITING                           0

23:42:46 DIAG_MON>/

USERNAME        SID    SERIAL# SQL_ID        STATUS   LAST_CALL_ET       SEQ#
-------- ---------- ---------- ------------- -------- ------------ ----------
EVENT                                                              STATE               SECONDS_IN_WAIT
------------------------------------------------------------------ ------------------- ---------------
HEMANT            9          3 9t7c0jy3v19pn INACTIVE             4      34020
SQL*Net message from client                                        WAITING                           4

….
```

Note the last wait at SEQ#34020. It is now 4 seconds ! Let's see the next slide.

Where's the Catch ?

```
23:52:02 DIAG_MON>/

USERNAME        SID    SERIAL# SQL_ID        STATUS   LAST_CALL_ET       SEQ#
-------- ---------- ---------- ------------- -------- ------------ ----------
EVENT                                                    STATE               SECONDS_IN_WAIT
---------------------------------------------------------------- ------------------- ---------------
HEMANT            9          3 9t7c0jy3v19pn INACTIVE          560      34020
SQL*Net message from client                                      WAITING                         560

23:52:03 DIAG_MON>
23:52:26 DIAG_MON>/

USERNAME        SID    SERIAL# SQL_ID        STATUS   LAST_CALL_ET       SEQ#
-------- ---------- ---------- ------------- -------- ------------ ----------
EVENT                                                    STATE               SECONDS_IN_WAIT
---------------------------------------------------------------- ------------------- ---------------
HEMANT            9          3 9t7c0jy3v19pn INACTIVE          584      34020
SQL*Net message from client                                      WAITING                         584

23:52:27 DIAG_MON>
```

The SEQ# is still 34020. Now, this is truly an idle "SQL*Net message from client" wait. The server is waiting for the client to send the next SQL command. It has finished sending the results of the last SQL statement. LAST_CALL_ET now reflect the time it is idle – because it is the time since the last call ended.

Get Execution Plan (without SELECT ANY TABLE)

```
DIAG_MON>select * from table(dbms_xplan.display_cursor('gr4zw7dd73r1x'));
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------
     --------------------------

SQL_ID  gr4zw7dd73r1x, child number 0
-------------------------------------
select count(*) from (select * from large_table minus select * from
another_large_table)

Plan hash value: 1789283771
--------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name                | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                     |       |       |       |   126K(100)|          |
|   1 |  SORT AGGREGATE       |                     |     1 |       |       |            |          |
|   2 |   VIEW                |                     |  4802K|       |       |   126K  (1)| 00:25:14 |
|   3 |    MINUS              |                     |       |       |       |            |          |
|   4 |     SORT UNIQUE       |                     |  4802K|   448M|   615M|   126K  (1)| 00:25:14 |
|   5 |      TABLE ACCESS FULL| LARGE_TABLE         |  4802K|   448M|       | 18935   (1)| 00:03:48 |
|   6 |     SORT UNIQUE       |                     |     1 |   207 |       |     3  (34)| 00:00:01 |
|   7 |      TABLE ACCESS FULL| ANOTHER_LARGE_TABLE |     1 |   207 |       |     2   (0)| 00:00:01 |
--------------------------------------------------------------------------------------------------------

Note
-----
   - dynamic sampling used for this statement (level=2)

24 rows selected.
DIAG_MON>
```

I don't have SELECT privilege on the underlying table in the HEMANT schema. Yet, I can get the execution plan. I don't need the SELECT privilege on the underlying table(s) or the SELECT ANY privilege or the DBA role to be able to do this.

Check Query Tables

```
DIAG_MON>select owner, table_name, num_rows, last_analyzed
  2  from dba_tables
  3  where table_name in ('LARGE_TABLE','ANOTHER_LARGE_TABLE');

OWNER                          TABLE_NAME                       NUM_ROWS LAST_ANAL
------------------------------ ------------------------------ ---------- ---------
HEMANT                         LARGE_TABLE                       4802944 27-FEB-15
HEMANT                         ANOTHER_LARGE_TABLE

DIAG_MON>
```

Because I have SELECT_CATALOG_ROLE, I can query the underlying statistics.

AWR

```
DIAG_MON>exec dbms_workload_repository.create_snapshot;

PL/SQL procedure successfully completed.

DIAG_MON>
DIAG_MON>@?/rdbms/admin/awrrpt  {OR Use SQLDeveloper 4.0.1}


Extracted from the AWR Report :

 Elapsed                  Elapsed Time
        Time (s)    Executions  per Exec (s)  %Total   %CPU    %IO    SQL Id
---------------- -------------- ------------- ------ ------ ------ -------------
            52.7              1         52.68   16.1   15.2    9.1 cx87ww8c8t206
Module: SQL*Plus
insert into another_large_table select * from large_table
```

I can create AWR snapshots because I have EXECUTE on DBMS_WORKLOAD_REPOSITORY. I don't need to be granted the DBA role.

I can use SQLDeveloper 4.0.1 to generate an AWR report without logging in to the server as "oracle"

XPLAN from AWR

```
DIAG_MON>select * from table(dbms_xplan.display_awr('cx87ww8c8t206'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
     ---------------------------
SQL_ID cx87ww8c8t206
--------------------
insert into another_large_table select * from large_table

Plan hash value: 1101256009

-------------------------------------------------------------------------------------------
| Id  | Operation                | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT         |             |       |       | 18935 (100)|          |
|   1 |  LOAD TABLE CONVENTIONAL |             |       |       |            |          |
|   2 |   TABLE ACCESS FULL      | LARGE_TABLE |  4802K|   448M| 18935   (1)| 00:03:48 |
-------------------------------------------------------------------------------------------


14 rows selected.

DIAG_MON>
```

I can retrieve historical execution plans captured by AWR.

V$SESSION_LONGOPS

- CAVEAT ! It is based on *operations*, not on the SQL Execution Time. An SQL execution can actually consist of multiple operations.
- An SQL that consists of multiple steps in the execution plan can consist of multiple operations, each appearing independently in this view.
- Parallel Query operations are split by block ranges (or partitions) – so may appear repeatedly, once for each block range / partition
- Nested Loop Full Table Scans can appear repeatedly
- Entries in this view may not be cleared quickly, you may see entries for sessions / queries that have completed

Too many people on the Internet think that this view shows how long the query is running and how long it is expected to continue.

It shows the *current operation* not the whole SQL. An SQL Query can consist of multiple operations. Even Parallel Query can run different block ranges using multiple passes, each pass is a separate operation (and each PQ slave a separate session, so a separate row in this view).

Session with Long Op ??

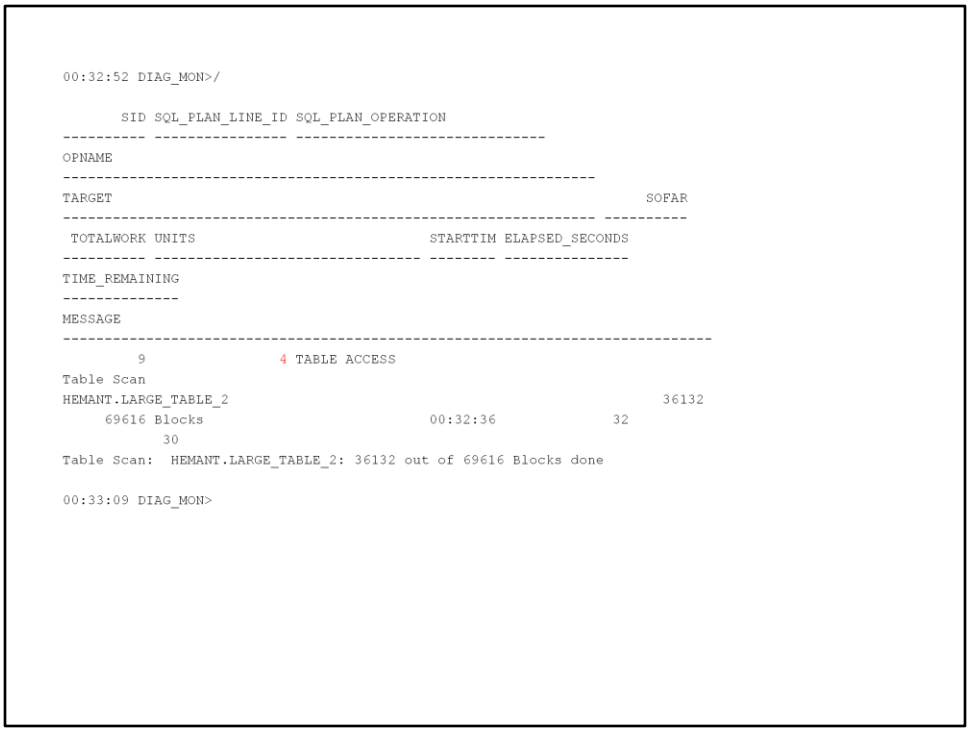
```
HEMANT>set time on
00:31:02 HEMANT>create table a_vl_table
00:31:18   2  as select * from large_table
00:31:38   3  union all
00:31:40   4  select * from large_table_2
00:31:46   5  union all
00:31:50   6  select * from large_table_3
00:31:55   7  /

Table created.

00:35:09 HEMANT>

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------
    ------------------------------
Plan hash value: xxxxxxxx
-------------------------------------------------------------------------------------------
| Id  | Operation              | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | CREATE TABLE STATEMENT |               |    14M|  1346M| 99106  (67)| 00:19:50 |
|   1 |  LOAD AS SELECT        | A_VL_TABLE    |       |       |            |          |
|   2 |   UNION-ALL            |               |       |       |            |          |
|   3 |    TABLE ACCESS FULL   | LARGE_TABLE   |  4802K|   448M| 18935   (1)| 00:03:48 |
|   4 |    TABLE ACCESS FULL   | LARGE_TABLE_2 |  4802K|   448M| 18935   (1)| 00:03:48 |
|   5 |    TABLE ACCESS FULL   | LARGE_TABLE_3 |  4802K|   448M| 18935   (1)| 00:03:48 |
-------------------------------------------------------------------------------------------

12 rows selected.

HEMANT>
```

Here is an SQL that a user is executing. He is querying 3 different tables. Let's see in the next few slides if V$SESSION_LONGOPS shows the execution of the whole SQL or parts (operations) only.

```
00:32:38 DIAG_MON>l
  1  select sid, sql_plan_line_id, sql_plan_operation, opname, target, sofar,
  2  totalwork, units, to_char(start_time,'HH24:MI:SS') StartTime,
  3  elapsed_seconds, time_remaining, message
  4  from v$session_longops
  5  where sofar != totalwork
  6* order by start_time
00:32:38 DIAG_MON>/
no rows selected
00:32:39 DIAG_MON>/
no rows selected
00:32:41 DIAG_MON>/
       SID SQL_PLAN_LINE_ID SQL_PLAN_OPERATION
---------- ---------------- ----------------------------
OPNAME
----------------------------------------------------------------
TARGET                                                           SOFAR
---------------------------------------------------------------- ----------
 TOTALWORK UNITS                          STARTTIM ELAPSED_SECONDS
---------- -------------------------------- -------- ---------------
TIME_REMAINING
--------------
MESSAGE
--------------------------------------------------------------------------------
         9                4 TABLE ACCESS
Table Scan
HEMANT.LARGE_TABLE_2                                             25661
     69616 Blocks                         00:32:36              15
            26
Table Scan:  HEMANT.LARGE_TABLE_2: 25661 out of 69616 Blocks done
```

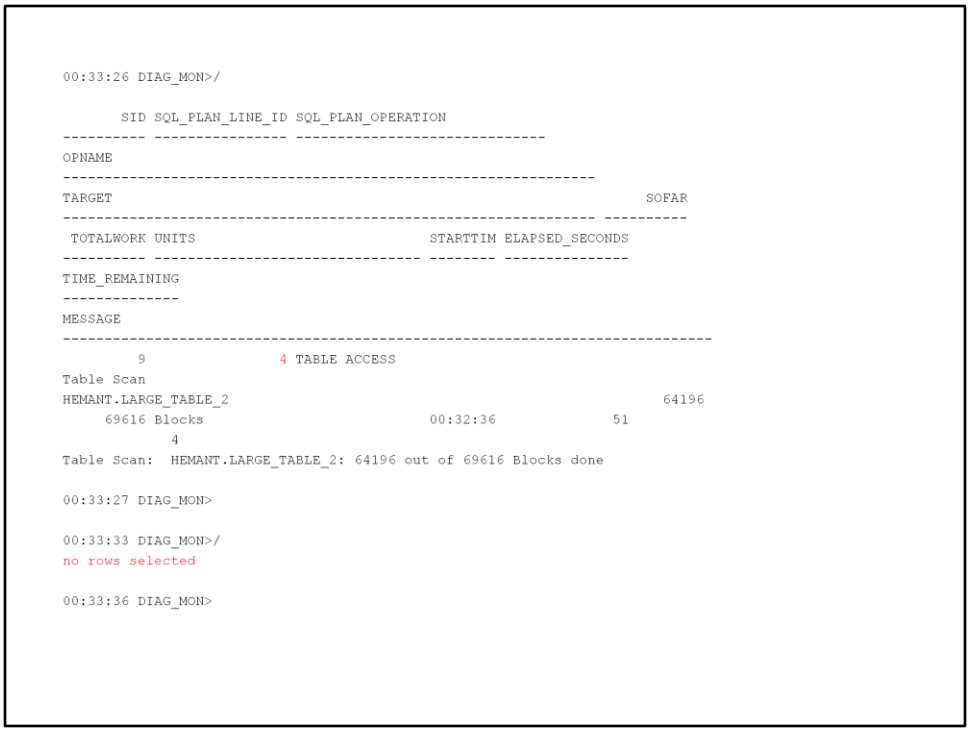
Look a the SQL_PLAN_LINE_ID. This extract from V$SESSION_LONGOPS is for only one step in the Execution Plan. The SOFAR, ELAPSED_SECONDS and TIME_REMAINING are for that one operation – reading table HEMANT.LARGE_TABLE_2

The estimated time for this operation is 15+26 = 41seconds.



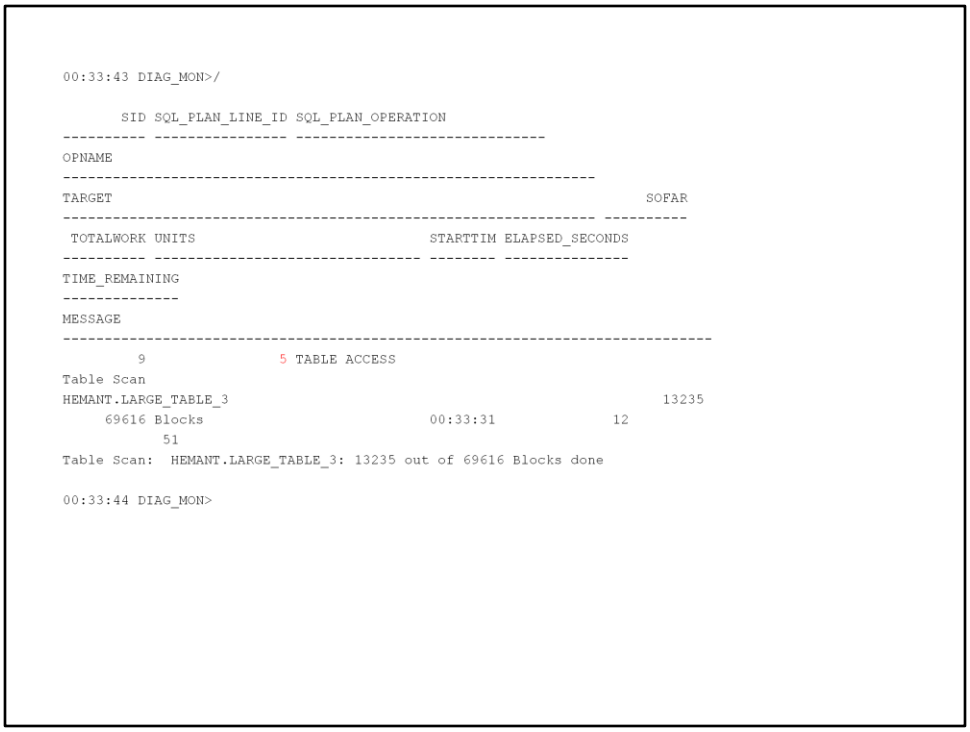
```
00:32:52 DIAG_MON>/

       SID SQL_PLAN_LINE_ID SQL_PLAN_OPERATION
---------- --------------- ------------------------------
OPNAME
----------------------------------------------------------------
TARGET                                                          SOFAR
---------------------------------------------------------------- ----------
 TOTALWORK UNITS                          STARTTIM ELAPSED_SECONDS
---------- ------------------------------ -------- --------------
TIME_REMAINING
--------------
MESSAGE
--------------------------------------------------------------------------------
         9               4 TABLE ACCESS
Table Scan
HEMANT.LARGE_TABLE_2                                              36132
     69616 Blocks                         00:32:36             32
           30
Table Scan:  HEMANT.LARGE_TABLE_2: 36132 out of 69616 Blocks done

00:33:09 DIAG_MON>
```

The operation has been running for 32seconds and still needs another 30seconds (i.e. 62seconds in all, not the 42seconds estimated earlier). Oracle is continously revising the estimated.



```
00:33:26 DIAG_MON>/

       SID SQL_PLAN_LINE_ID SQL_PLAN_OPERATION
---------- --------------- ------------------------------
OPNAME
----------------------------------------------------------------
TARGET                                                          SOFAR
---------------------------------------------------------------- ----------
 TOTALWORK UNITS                          STARTTIM ELAPSED_SECONDS
---------- ------------------------------ -------- --------------
TIME_REMAINING
--------------
MESSAGE
--------------------------------------------------------------------------------
         9                4 TABLE ACCESS
Table Scan
HEMANT.LARGE_TABLE_2                                              64196
     69616 Blocks                         00:32:36              51
              4
Table Scan:  HEMANT.LARGE_TABLE_2: 64196 out of 69616 Blocks done

00:33:27 DIAG_MON>

00:33:33 DIAG_MON>/
no rows selected

00:33:36 DIAG_MON>
```

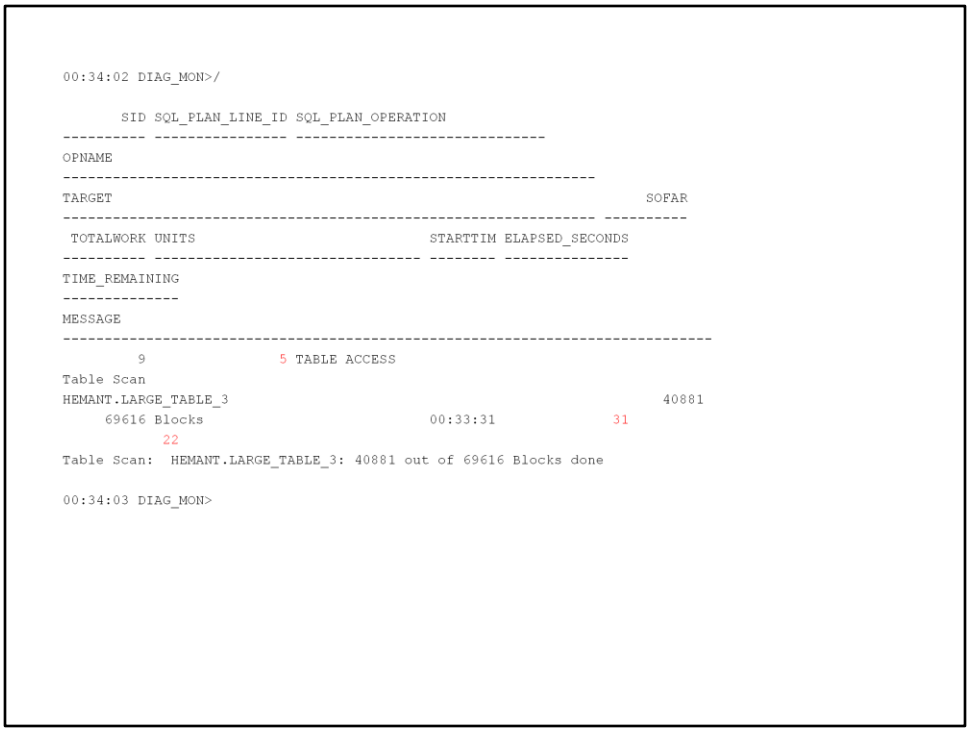
After the operation on Execution Plan Step 4 had completed, V$SESSION_LONGOPS stopped reporting this session. But is the session still active ?



```
00:33:43 DIAG_MON>/

       SID SQL_PLAN_LINE_ID SQL_PLAN_OPERATION
---------- --------------- ------------------------------
OPNAME
----------------------------------------------------------------
TARGET                                                          SOFAR
---------------------------------------------------------------- ----------
 TOTALWORK UNITS                          STARTTIM ELAPSED_SECONDS
---------- ------------------------------ -------- --------------
TIME_REMAINING
--------------
MESSAGE
--------------------------------------------------------------------------------
         9               5 TABLE ACCESS
Table Scan
HEMANT.LARGE_TABLE_3                                              13235
     69616 Blocks                         00:33:31             12
           51
Table Scan:  HEMANT.LARGE_TABLE_3: 13235 out of 69616 Blocks done

00:33:44 DIAG_MON>
```

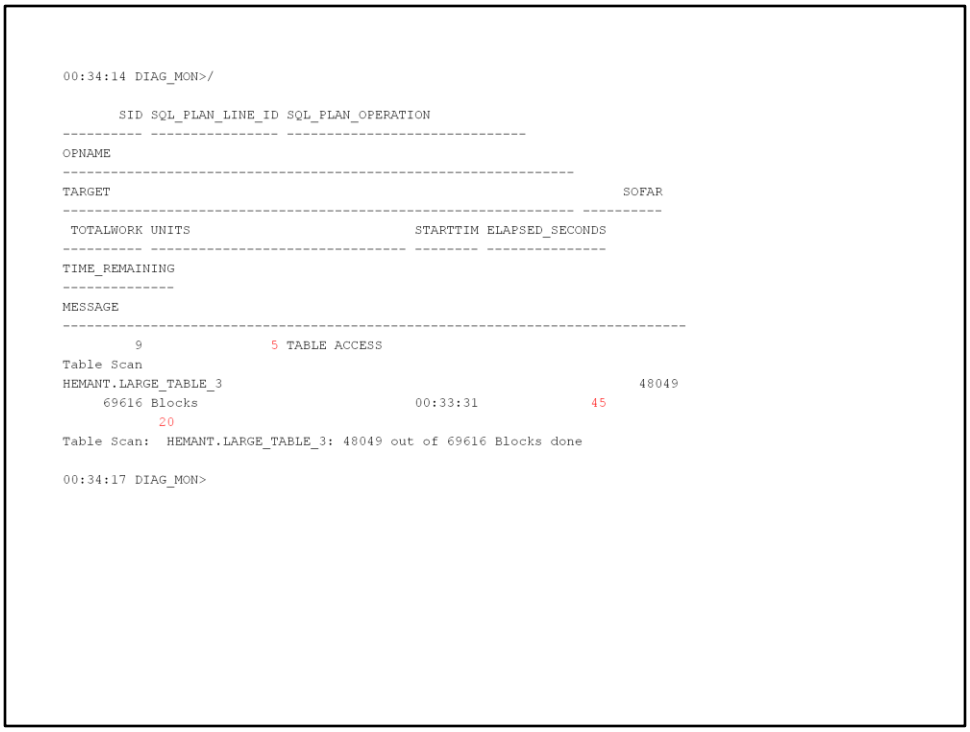
The session is still active. It is now on Execution Plan Step 5 – the next table in the SQL operation.

A database server process can do only 1 thing at a time. If it is querying LARGE_TABLE_2, it cannot also be querying LARGE_TABLE_3 at the same time. The retrieval of rows from LARGE_TABLE_3 is sequentially done later ! (Parallel Query is a way around the fundamental rule that a process can be doing only one thing at any time – PQ spawns multiple processes to do multiple things (reads from different block ranges and/or partitions of the same table) concurrently)



```
00:34:02 DIAG_MON>/

       SID SQL_PLAN_LINE_ID SQL_PLAN_OPERATION
---------- --------------- ------------------------------
OPNAME
----------------------------------------------------------------
TARGET                                                           SOFAR
---------------------------------------------------------------- ----------
 TOTALWORK UNITS                          STARTTIM ELAPSED_SECONDS
---------- ------------------------------ -------- --------------
TIME_REMAINING
--------------
MESSAGE
--------------------------------------------------------------------------------
         9                5 TABLE ACCESS
Table Scan
HEMANT.LARGE_TABLE_3                                               40881
     69616 Blocks                         00:33:31              31
            22
Table Scan:  HEMANT.LARGE_TABLE_3: 40881 out of 69616 Blocks done

00:34:03 DIAG_MON>
```

In the previous slide, the estimate for the read from HEMANT.LARGE_TABLE_3 was (12 + 51) 63seconds. It is now 53seconds.



```
00:34:14 DIAG_MON>/

       SID SQL_PLAN_LINE_ID SQL_PLAN_OPERATION
---------- --------------- ------------------------------
OPNAME
----------------------------------------------------------------
TARGET                                                           SOFAR
---------------------------------------------------------------- ----------
 TOTALWORK UNITS                          STARTTIM ELAPSED_SECONDS
---------- ------------------------------ -------- ---------------
TIME_REMAINING
--------------
MESSAGE
--------------------------------------------------------------------------------
         9               5 TABLE ACCESS
Table Scan
HEMANT.LARGE_TABLE_3                                             48049
     69616 Blocks                         00:33:31              45
           20
Table Scan:  HEMANT.LARGE_TABLE_3: 48049 out of 69616 Blocks done

00:34:17 DIAG_MON>
```

The estimate has now changed to 65seconds.

For another example of misreading V$SESSION_LONGOPS on a DML that does a Full Table Scan see http://hemantoracledba.blogspot.com/2009/01/when-not-to-use-vsessionlongops.html

Array Size

- Clients / ETL servers can use differing Array Sizes to fetch rows
- This means that the number of round-trips can vary
- The (assumed to be idle) wait event 'SQL*Net message from client' can be an indicator

I had mentioned earlier that a query that sends multiple rows to a client / application server sends the rows in batches – based on the ARRAY Size. Search my blog for examples of ARRAYSIZE (and LINESIZE and PAGESIZE if using an SQLPlus Client) {I have a few different blogposts on this}

I have shown earlier how the "SQL*Net message from client" isn't always an Idle Event. The presence of this wait event, with increasing SEQ# can indicate array fetches.

Array Size

```
select sql_id, sql_text, rows_processed, fetches
from v$sqlstats
where upper(sql_text) like
'SELECT % FROM HKC_TARGET_1% ORDER BY %'
order by 1

SQL_ID
-------------
SQL_TEXT
------------------------------------------------------------------------------------------
    ------------------------------------------------------------

ROWS_PROCESSED    FETCHES
------------- ----------
0g29ksyksnxyw
select * from hkc_Target_1 tt order by 2
       1142966      11431

637hm0n25b5gh
select * from hkc_target_1 order by 1
       1142966    1142967
```

Note the two queries retrieved the same number of rows. The elapsed time reported by the client would have included the “SQL*Net message wait from client” wait event on the server for the multiple round trips. The FETCHES count indicates the round-trips. The first SQL used an ARRAYSIZE of 100, the second was doing a Row-By-Row FETCH. (The extra 1 FETCH is always present when you run an SQL, you’ll even see it in the trace file as a FETCH with 0 rows executed first).

V$ACTIVE_SESSION_HISTORY

- A snapshot is collected every second. Thus, NOT every Wait or ON_CPU status is collected.
- To see if waits are increasing, also monitor SEQ#
- To track an SQL, look for the SQL_ID appearing in consecutive snapshots for the same SESSION_ID, SESSION_SERIAL#
- Never try to add TIME_WAITED from V$A_S_H
- Note : Querying V$A_S_H requires the Diagnostic Pack Licence. If you don't have access, you'll have to use V$SESSION and sample it frequently
- V$ASH availability is based on shared_pool_size and load, there is no guarantee how far back you can look

DBA_HIST_ACTIVE_SESS_HISTORY is a sample every 10seconds (not "1 in 10samples"). If I want to see the "distribution of a session or SQL over the CPU and wait events", I look at the number of samples, not a summation of TIME_WAITED.

The composite key for an SQL execution within a session is SQL_ID, SQL_EXEC_START, SQL_EXEC_ID

Remember : SQL Operations that completed between 2 snapshots (SAMPLE_TIME) are *not* captured !!

Here I present only a few examples of analysis using this view. There are many more useful columns like BLOCKING%, CURRENT%, QC% (e.g. I've seen people look at PGA_ALLOCATED and TEMP_SPACE_ALLOCATED and have a query based on these as well).

Where a session has spent time (note : add filter by sample_time)

```
DIAG_MON>l
  1  select session_state, event, count(*)
  2  from V$active_session_history
  3  where session_id=195
  4   - and sample_time between '  ' and ' '
  5  group by session_state, event
  6* order by 1,2
DIAG_MON>/

SESSION EVENT                                                          COUNT(*)
------- ---------------------------------------------------------------- ----------
ON CPU                                                                          2
WAITING cursor: pin S wait on X                                                 3
WAITING db file sequential read                                                10
WAITING log file sync                                                           1
WAITING read by other session                                                   5

DIAG_MON>
```

I've not shown the filter by SAMPLE_TIME here. It is a very short elapsed time of operations by Session 195. I can see that most of the samples indicate waiting on "db file sequential read" – more so than On CPU. I say "more samples" rather than "more time" as being more accurate.

Distribution across SQLs (note : add filter by sample_time if necessary)

```
DIAG_MON>l
  1  select sql_id, session_state, event, count(*)
  2  from V$active_session_history
  3  where session_id=195
  4  group by sql_id, session_state, event
  5  having count(*) > 2
  6* order by 1,4,2,3
DIAG_MON>/

SQL_ID        SESSION EVENT                       COUNT(*)
------------- ------- ------------------------ ----------
5ys3vrapmbx6w WAITING direct path read                 18
64mgk3gjr8pdk ON CPU                                    8
83n9t3c9rsxfj WAITING Data file init write              3
83n9t3c9rsxfj WAITING buffer busy waits                10
83n9t3c9rsxfj WAITING db file scattered read           23
83n9t3c9rsxfj WAITING free buffer waits                26
83n9t3c9rsxfj ON CPU                                   30
83n9t3c9rsxfj WAITING log buffer space                 76
calqgpqu9c2v4 WAITING read by other session             4
g0jvz8csyrtcf WAITING db file sequential read           4

10 rows selected.

DIAG_MON>
```

Here, the session ran multiple SQL statements. I can see the distribution of CPU and Wait Events amongst the different SQL.

By Module, rather than session (note : add filter by sample_time)

```
DIAG_MON>l
  1  select sql_id, session_state, event, count(*)
  2  from V$active_session_history
  3  where module  = 'SQL*Plus'
  4  group by sql_id, session_state, event
  5  having count(*) > 2
  6* order by 1,4,2,3
DIAG_MON>/

SQL_ID        SESSION EVENT                       COUNT(*)
------------- ------- ------------------------ ----------
5ys3vrapmbx6w WAITING direct path read                 18
64mgk3gjr8pdk ON CPU                                    8
83n9t3c9rsxfj WAITING Data file init write              3
83n9t3c9rsxfj WAITING buffer busy waits                10
83n9t3c9rsxfj WAITING db file scattered read           23
83n9t3c9rsxfj WAITING free buffer waits                26
83n9t3c9rsxfj ON CPU                                   30
83n9t3c9rsxfj WAITING log buffer space                 76
              WAITING log file sync                    12

9 rows selected.

DIAG_MON>
```

Here, I filter for a MODULE, rather than a SESSION (I've not shown the filter by SAMPLE_TIME). (Note : "log file sync" wait may not always show which the SQL_ID was that was waiting on the Event).

By SQL (which SQL is “busiest”)

```
DIAG_MON>l
  1  select sql_id, session_state, event, count(*)
  2  from V$active_session_history
  3  where
  4  sample_time > sysdate - 0.5/24
  5  and session_type = 'FOREGROUND'
  6  group by sql_id, session_state, event
  7  having count(*) > 2
  8* order by 1,4,2,3
DIAG_MON>/

SQL_ID        SESSION EVENT                          COUNT(*)
------------- ------- ---------------------------- ----------
5ys3vrapmbx6w WAITING direct path read                     18
64mgk3gjr8pdk ON CPU                                        8
83n9t3c9rsxfj WAITING Data file init write                  3
83n9t3c9rsxfj WAITING buffer busy waits                    10
83n9t3c9rsxfj WAITING db file scattered read               23
83n9t3c9rsxfj WAITING free buffer waits                    26
83n9t3c9rsxfj ON CPU                                       30
83n9t3c9rsxfj WAITING log buffer space                     76
cvn54b7yz0s8u WAITING db file sequential read               4
              WAITING log file sync                        12

10 rows selected.

DIAG_MON>
```

Here, I query for the last 30minutes.

By SQL Plan Line

```
DIAG_MON>l
  1  select sample_time, sql_id, sql_plan_line_id,session_state, event
  2  from v$active_Session_history
  3  where session_id=140
  4  and sample_time > sysdate-0.5/24
  5* order by 1
DIAG_MON>/
SAMPLE_TIME               SQL_ID        SQL_PLAN_LINE_ID SESSION EVENT
------------------------- ------------- ---------------- ------- ----------------------
    ---
20-JUN-15 11.34.41.811 PM 8zk9mqxkr2jsq                  ON CPU
20-JUN-15 11.37.30.274 PM 0f59859k2n07p                1 ON CPU
20-JUN-15 11.39.20.646 PM f90msdgm5858k               37 WAITING direct path read
20-JUN-15 11.39.21.646 PM f90msdgm5858k                2 ON CPU
20-JUN-15 11.39.22.646 PM f90msdgm5858k               37 WAITING direct path read
…
20-JUN-15 11.39.27.646 PM f90msdgm5858k               37 ON CPU
20-JUN-15 11.39.28.656 PM f90msdgm5858k               37 WAITING direct path read
…
20-JUN-15 11.39.43.666 PM f90msdgm5858k               37 WAITING direct path read
20-JUN-15 11.41.20.212 PM drna7u98myhhx               18 ON CPU
20-JUN-15 11.41.21.212 PM drna7u98myhhx               18 ON CPU
20-JUN-15 11.41.22.212 PM drna7u98myhhx               19 ON CPU
20-JUN-15 11.41.23.212 PM drna7u98myhhx               18 ON CPU
…
20-JUN-15 11.41.37.232 PM drna7u98myhhx               19 ON CPU
```

Here, I can also see, for each SQL, the time on each Step in the Execution Plan. So, I know which Step is likely to have accounted for more time (on the basis of the number of times it was sampled) !

Distribution of Approx CPU usage – by Service

```
select s.name, count(*) On_CPU_Count,
      trunc((ratio_to_report(count(*)) over ())*100,0) percentage
from dba_hist_active_sess_history h, dba_services s
where h.service_hash=s.name_hash
and h.sample_time between to_date('&from_date','DD-MON-RR')
                      and to_date('&to_date','DD-MON-RR')
and h.session_state='ON CPU'
group by s.name
order by 3,1
/
```

NAME	ON_CPU_COUNT	PERCENTAGE
SYS$BACKGROUND	6351	27
SYS$USERS	16826	72

Distribution of CPU Usage (this is an approximation because it is based on a sample taken every 10seconds only !) This is based on number of occurrences in samples, not actual time spent on CPU. But we could approximate the one for the other.

CPU and Waits for a User

```
select DECODE(On_CPU.Sample_Time,NULL,In_Wait.Sample_Time, On_CPU.Sample_Time) as Sample_Time, NVL(On_CPU.CNT,0) as On_CPU,
        NVL(In_Wait.CNT,0) as In_Wait
from
(
select  sample_time, count(*) CNT
from
(
select  h.sample_time,h.session_state, h.event
from v$active_session_history h
where 1=1
and h.user_id='&&userid'
and h.sample_time >sysdate-1/24
)
where session_state='ON CPU'
group by  sample_time
) On_CPU FULL OUTER JOIN
(
select  sample_time, count(*) CNT
from
(
select  h.sample_time, h.session_state, h.event
from v$active_session_history h
where 1=1
and h.user_id='&&userid'
and h.sample_time >sysdate-1/24
)
where session_state='WAITING'
-- and event='db file sequential read'
group by  sample_time
) In_Wait
ON In_Wait.sample_time=On_CPU.sample_time
order by 1,2
/
```

To chart the number of sessions in ON_CPU and in WAITING states. You could select for a specific wait event as well.
(SQLPLUS allows you to use SET COLSEP ‘,’ and SET PAGESIZE 0 to spool to a CSV file)

Note : Information available in V$ACTIVE_SESSION_HISTORY is limited by memory space allocated.

Active Sessions Count

SAMPLE_TIME	ON_CPU	IN_WAIT
12-JUL-15 02.34.00.239 PM	1	0
12-JUL-15 02.34.15.329 PM	0	1
12-JUL-15 02.34.18.339 PM	3	0
12-JUL-15 02.34.19.339 PM	0	5
12-JUL-15 02.34.20.349 PM	5	0
12-JUL-15 02.34.21.359 PM	6	0
12-JUL-15 02.34.22.359 PM	7	1
12-JUL-15 02.34.23.369 PM	0	5
12-JUL-15 02.34.24.369 PM	0	5
12-JUL-15 02.34.25.399 PM	0	7
12-JUL-15 02.34.26.409 PM	0	7
12-JUL-15 02.34.27.409 PM	0	5
12-JUL-15 02.34.28.409 PM	0	4
12-JUL-15 02.34.29.419 PM	0	4
12-JUL-15 02.34.30.429 PM	0	5
12-JUL-15 02.34.31.429 PM	3	2
12-JUL-15 02.34.32.439 PM	0	5
12-JUL-15 02.34.33.439 PM	0	5
12-JUL-15 02.34.34.439 PM	0	5
12-JUL-15 02.34.35.449 PM	0	5
12-JUL-15 02.34.36.459 PM	0	5
12-JUL-15 02.34.37.459 PM	0	5

This is another way to represent Active Sessions.

SQL Executions History

```
select  sq.snap_id snap_id,
to_char(plan_hash_value) Plan_Hash_Value,
executions_delta Executions,
elapsed_time_delta/1000000 Total_Elapsed_Time_Secs,
elapsed_time_delta/1000000/decode(executions_delta,0,1,executions_delta)
    Elapsed_Time_Secs_PerExec,
disk_reads_delta/decode(executions_delta,0,1,executions_delta) Disk_Reads_PerExec,
buffer_gets_delta/decode(executions_delta,0,1,executions_delta) Buffer_Gets_PerExec,
rows_processed_delta/decode(executions_delta,0,1,executions_delta)
    Rows_Processed_PerExec
from dba_hist_sqlstat sq, dba_hist_snapshot  ss
where
sq.dbid=ss.dbid
and sq.snap_id=ss.snap_id
and
sql_id = '&&sql_id'
--and executions_delta <> 0
order by  snap_id
/
```

Note : Requires Diagnostic Pack Licence

Extract previous occurrences of an SQL from AWR history

SQL Executions History

```
Snap_ID Plan_Hash_Valu   EXECUTIONS TOTAL_ELAPSED_TIME_SECS ELAPSED_TIME_SECS_PEREXEC
   DISK_READS_PEREXEC BUFFER_GETS_PEREXEC ROWS_PROCESSED_PEREXEC

---------- ------------ ------------ ---------------------- ------------------------
- ----------------- ------------------ ---------------------

      1087 1586852085                3                      65
     22            144,175             187,402               768,094

      1088 1586852085                1                      98
     98            782,464             930,067             3,168,456

      1089 1586852085                2                      72
     36            198,984             330,982               676,449
```

The same SQL has had varying numbers of Buffer Gets and Rows Processed. For example, in Snapshot 1088, there was 1 execution for 3million rows. The other two snapshots had 2 or 3 executions for 676K or 768K rows each.

Note : PLAN_HASH_VALUE does not change if ROWS or COST changes. See http://hemantoracledba.blogspot.com/2014/03/plan-hashvalue-remains-same-for-same.html (So, two queries with the same P_H_V don't have to have the same expected runtime !)

CPU Consumption by Resource Consumer Group

```
DIAG_MON>l
  1  select to_char(begin_Time,'DD-MON HH24:MI') From_Time, to_char(end_time,'DD-MON HH24:MI')
     To_Time,
  2  consumer_group_name,
  3  60 * (select value from v$parameter where name = 'cpu_count') max_db_sec,
  4  cpu_consumed_time/1000 consumed_sec
  5  from v$rsrcmgrmetric_history
  6* order by begin_time, consumer_group_name
DIAG_MON>/
```

Note : View shows statistics only for the past 1 hour

FROM_TIME	TO_TIME	CONSUMER_GROUP_NAME	MAX_DB_SEC	CONSUMED_SEC
11-JUL 23:56	11-JUL 23:57	OTHER_GROUPS	240	.589
11-JUL 23:56	11-JUL 23:57	PX_QUERY_USER	240	12.55
11-JUL 23:56	11-JUL 23:57	_ORACLE_BACKGROUND_GROUP_	240	0
11-JUL 23:57	11-JUL 23:58	OTHER_GROUPS	240	.02
11-JUL 23:57	11-JUL 23:58	PX_QUERY_USER	240	10.085
11-JUL 23:57	11-JUL 23:58	_ORACLE_BACKGROUND_GROUP_	240	0
11-JUL 23:58	11-JUL 23:59	OTHER_GROUPS	240	.016
11-JUL 23:58	11-JUL 23:59	PX_QUERY_USER	240	18.993
11-JUL 23:58	11-JUL 23:59	_ORACLE_BACKGROUND_GROUP_	240	0
11-JUL 23:59	12-JUL 00:00	OTHER_GROUPS	240	1.527
11-JUL 23:59	12-JUL 00:00	PX_QUERY_USER	240	0
11-JUL 23:59	12-JUL 00:00	_ORACLE_BACKGROUND_GROUP_	240	0

If Resource Manager is implemented, I can also look at the CPU usage by each Consumer Group.

Note : When “Virtual CPUs” (as in VMs or HyperThreading) are used, the CPU Count may be larger than the number of actual cores so CPU time may be misleading relative to the actual number of cores.

Parallel Query Usage

```
DIAG_MON>l
  1  select to_char(sysdate,'DD-MON HH24:MI'), p.qcsid, s.sql_id, p.req_degree, p.degree, count(*)
  2  from v$px_session p, v$session s
  3  where p.sid=s.sid
  4  and p.serial#=s.serial#
  5  and p.req_degree is not null
  6* group by to_char(sysdate,'DD-MON HH24:MI'), p.qcsid, s.sql_id, p.req_degree, p.degree
DIAG_MON>
DIAG_MON>/
TO_CHAR(SYSDATE,'DD-M      QCSID SQL_ID        REQ_DEGREE     DEGREE   COUNT(*)
-------------------- ---------- ------------- ---------- ---------- ----------
12-JUL 00:15                135 20mwzfa97gczs          4          4          8

DIAG_MON>/
TO_CHAR(SYSDATE,'DD-M      QCSID SQL_ID        REQ_DEGREE     DEGREE   COUNT(*)
-------------------- ---------- ------------- ---------- ---------- ----------
12-JUL 00:17                 18 4k9ruvz7nmuhj          4          4          8
12-JUL 00:17                135 2f8xqbm44zxnn          4          4          4

DIAG_MON>/
TO_CHAR(SYSDATE,'DD-M      QCSID SQL_ID        REQ_DEGREE     DEGREE   COUNT(*)
-------------------- ---------- ------------- ---------- ---------- ----------
12-JUL 00:17                135 20mwzfa97gczs          4          4          8

DIAG_MON>/
TO_CHAR(SYSDATE,'DD-M      QCSID SQL_ID        REQ_DEGREE     DEGREE   COUNT(*)
-------------------- ---------- ------------- ---------- ---------- ----------
12-JUL 00:17                135 dfpjydp33tvnr          4          4          4
```

I can see how many PQs are running (look at QCSID as the Query Co-ordinator). I can see if each QC did get the actual degree (DEGREE) that was requested (REQ_DEGREE). So, if the server doesn't have enough PARALLEL_MAX_SERVERS, the actual degree may be less than the requested degree.

Thus, at 00:15, there was 1 query requesting DoP=4, getting Dop=4 but actually running with 8 PQ slaves (Parallel Execution Servers). Why, because a query may take 2 (or, in very rare cases, more) Slave Sets.

At 00:17, there were two queries from two different sessions executing with a total of 12 PX servers.

See my blog posts on PX Servers.

Transactions

- Look at START_TIME, USED_UBLK and USED_UREC in V$TRANSACTION. These do NOT map to Table Rows (or Rows + Index Entries)
- A Transaction can consist of multiple SQL calls
- You cannot identify which SQL in the transaction generated how much Undo
- USED_UREC, USED_UBLK start reducing when a rollback begins (user gets message only when rollback is completed)
- (join V$TRANSACTION to V$SESSSION on T.ADDR=S.TADDR)
- Direct Path operations (INSERT APPEND or PARALLEL) generate only 1 Undo Record

Very important : UNDO RECORDS is NOT the same (or same size) as Table Rows or Index Entries.

Active Transaction

```
00:28:14 DIAG_MON>l
  1   select s.sid, s.serial#, s.username, substr(s.program,1,10) Prognm,
  2  t.xidusn, t.used_ublk, t.used_urec,t.start_time,
  3  s.last_call_et, s.sql_id Current_SQL_ID_if_available
  4  from v$session s, v$transaction t
  5* where s.taddr=t.addr
00:28:14 DIAG_MON>/

       SID    SERIAL# USERNAME PROGNM         XIDUSN  USED_UBLK  USED_UREC START_TIME
    LAST_CALL_ET CURRENT_SQL_I
---------- ---------- -------- ---------- ---------- ---------- ---------- -------------------- -----
    ------- -------------
       135        173 HEMANT   sqlplus@lo          9       5361     184675 07/12/15 00:27:00
    74 gaplj5qttgc62

00:28:15 DIAG_MON>
0:28:48 DIAG_MON>/


       SID    SERIAL# USERNAME PROGNM         XIDUSN  USED_UBLK  USED_UREC START_TIME
    LAST_CALL_ET CURRENT_SQL_I
---------- ---------- -------- ---------- ---------- ---------- ---------- -------------------- -----
    ------- -------------
       135        173 HEMANT   sqlplus@lo          9       5069     174809 07/12/15 00:27:00
    196

00:30:18 DIAG_MON>00:30:18 DIAG_MON>
```

You can check the distribution of transactions across different Undo Segments.

Remember : The SQL_ID is only the current SQL. A transaction can consist of multiple DMLs, including SELECT queries ! The current SQL may be a SELECT but the transaction may have 1 or 100 or 1000 INSERT/UPDATE/DELETE statements before this.

The transaction started rolling back after 00:28:15. (Also note that Current SQL_ID is not always available)

Instance and Session Statistics and Events

- Use V$SYSSTAT and V$SESSTAT (joined to V$STATNAME) for Cumulative Instance / Session Statistics.
- Use V$SYSTEM_EVENT and V$SESSION_EVENT for Cumulative Instance / Session Events (Waits)
- It is possible to do a query that Union of the STATS and EVENT views. ("CPU used by this session" is in the STATS view)
- I use this for a "Profile" view somewhat similar to the header information in AWR (*without* the "Hit Ratios")

I can compare CPU time with Wait Events time. I can identify the top Wait. I can filter for statistics or waits that I am particularly interested in.

Profiling Waits and Statistics

```
DIAG_MON>l
  1  select 'Event : ' || event Event_or_Stat, time_waited, total_waits,
     total_timeouts, average_wait
  2  from v$system_event
  3  where
  4      event not like 'rdbms ipc message%'
  5  and event not like 'SQL*Net%'
  6  and event not like 'pipe get%'
  7  and event not like 'pmon timer%'
  8  and event not like 'smon timer%'
  9  union
 10  select  'Statistic : ' || name a, value b,  null c, null d, null e
 11  from v$sysstat
 12  where name in ('CPU used by this session')
 13  union
 14  select  'Statistic : ' || name a, null b, value c, null d, null e
 15  from v$sysstat
 16  where name in
 17  ('consistent changes','consistent gets','db block gets', 'db block
     changes','physical reads',
 18  'redo blocks written', 'redo entries', 'redo size', 'user calls', 'user commits')
 19* order by 1
DIAG_MON>
```

Event Wait Time and CPU used are in CentiSeconds

I can select which Statistics and which Wait Events are of interest to me for this particular database instance. (Different application profiles may have different interesting Statistics and Waits !)

This gives me a “Profile” view of the Instance

Instance Waits and Stastics

EVENT_OR_STAT	TIME_WAITED	TOTAL_WAITS	TOTAL_TIMEOUTS	AVERAGE_WAIT
Event : ARCH wait for process	202	2	2	101.16
Event : db file parallel read	99	41	0	2.41
Event : db file scattered read	4282	61863	0	.07
Event : db file sequential rea	63417	203607	0	.31
Event : db file single write	1	32	0	.03
Event : direct path read	159831	936937	0	.17
Event : direct path read temp	2	496	0	0
Event : log buffer space	571	20	0	28.57
Statistic : CPU used by this s	63995			
Statistic : consistent changes		17863		
Statistic : consistent gets		16648379		
Statistic : db block changes		920998		
Statistic : db block gets		762072		
Statistic : physical reads		15934919		
Statistic : redo blocks writte		312956		
Statistic : redo entries		456759		
Statistic : redo size		154028216		
Statistic : user calls		226709		
Statistic : user commits		1074		
	---			
sum	24803993			

Note : Time_Waited and CPU used are both in Centi-seconds.
Here I can compare CPU time with time on major wait events. I can also look at major statistics that I can filter.