From 10046 to AWR / ADDM / RTM / ASH
Modern Techniques for Diagnostics

John Hurley
Senior Database Developer
Federal Reserve Bank of Cleveland

Twitter handle: @grumpyolddba
Email: hurleyjohnb@yahoo.com

Working with Oracle for 20 years Oracle Ace

I attend one or two conferences a year

Retired President of Northeast Ohio Oracle Users Group NEOOUG

Great Lakes Oracle Conference in late May 2018

Outside interests:

Running / SJA cross country team / Music / Science Fiction

Learning to play guitar


DO NOT PAY
ATTENDANT

Agenda for presentation:

- **My experience when I started working with Oracle**
- **A history of built in database instrumentation within Oracle software including some of the tools and approaches available at various stages**
- **An emphasis on critical ideas and context plus a few tips/tricks on how to maybe follow the yellow brick road**
- **10046 tracing overview / methodology / some details**
- **It's about time**
- **AWR/ADDM/RTM**
- **ASH**

GLOC 2019

GLOC 2019

My transition from IBM Mainframe environment to Oracle database:

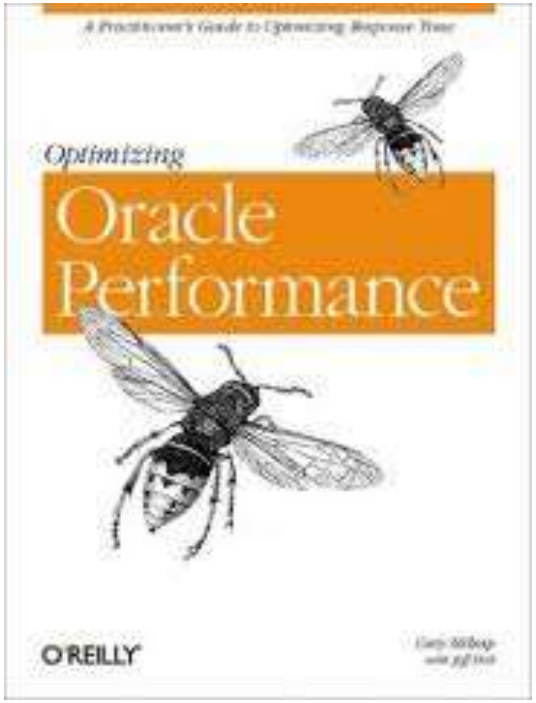
Several of the companies that I had worked for doing mainframe work had documented transaction processing times for their critical lines of business.

Does anyone still do this anymore? I hope so but I don't see that much anymore.

How fast does it have to be? Is anyone screaming about it?

I was confused about how to solve oracle performance problems the tools that I had in the mainframe world (CICS / IMS) were gone.


execute


Optimizing
Oracle
Performance
O'REILLY

Method C: The Trial-and-Error Method That Dominates the Oracle Performance Tuning Culture Today

1. Hypothesize that some performance metric *x* has an unacceptable value.
2. Try things with the intent of improving *x*. Undo any attempt that makes performance noticeably worse.
3. If users do *not* perceive a satisfactory response time improvement, then go to step 1.
4. If the performance improvement *is* satisfactory, then go to step 1 anyway, because it may be possible to produce other performance improvements if you just keep searching.

```
SQL> -- Get initial Buffer Hit Ratio reading...
SQL> SELECT ROUND((1-(phy.value / (cur.value + con.value)))*100,2) "Cache Hit Ratio"
  2    FROM v$sysstat cur, v$sysstat con, v$sysstat phy
  3   WHERE cur.name = 'db block gets'
  4     AND con.name = 'consistent gets'
  5     AND phy.name = 'physical reads'
  6  /

Cache Hit Ratio
---------------
         90.75
```

STATISTIC	EXAMPLE 1
Current gets	250
Consistent gets	900
Physical reads	150
DBCHR	87.0

Connor McDonalds “pick a cache hit buffer ratio” script.

All we have to do is put in a sql statement that chews some CPU and time and does another 89k consistent gets and bingo!

STATISTIC	EXAMPLE 1	EXAMPLE 2
Current gets	250	250
Consistent gets	900	90000
Physical reads	150	150
DBCHR	87.0	99.8

Even in the early days of Oracle Performance Tuning some smart people were doing things a lot better.

V$SESSION

How many ACTIVE sessions that are not BACKGROUND are running at 9 am / 10 am / 1 pm / 2 pm ...
What SQL shows up in these ACTIVE sessions
How long does it take these SQL statements to process

```
--
SELECT substr(sq.sql_text,1,100) what_sql,
       vs.last_call_et,
       vs.sql_id,
       vs.sid, vs.serial#,
       vs.username, vs.machine, vs.module
FROM   v$session vs,
       v$sql     sq
WHERE  vs.status           =  'ACTIVE'
  AND  vs.type             <> 'BACKGROUND'
  AND  vs.schemaname       <> 'SYS'
  AND  vs.sql_id           =   sq.sql_id
  AND  vs.sql_child_number =   sq.child_number
ORDER BY 2 desc;
```

- Database Time = Total time spent by sessions in the database server actively working (on CPU) or actively waiting (non-idle wait)

- Active Sessions = The number of sessions active (working or waiting) in the database server at a particular time

- Average Active Sessions = DB Time / Elapsed Time

WHAT_SQL	LAST_CALL_ET
select * from a, b where a.START_DATETIME > to	5733
SELECT /* */ aat.agency_app_trans_id, aat	266
SELECT /* */ AGENCY_NAME, AGENCY_SHORT_NAME,	11
SELECT /* */ AGENCY_NAME, AGENCY_SHORT_NAME,	8
SELECT /* */ AGENCY_NAME, AGENCY_SHORT_NAME,	4
SELECT tas_id, tas_label, tas_string, decode(trim(tas_string), '','COMPONENT', null,'COMPONE	0
SELECT substr(sq.sql_text,1,100) what_sql, vs.last_call_et, vs.sql_id, vs.sid,	0
SELECT aasv.agency_app_id, aasv.agency_app_name, aasv.agency_id,	0

LAST_CALL_ET	SQL_ID	SID	SERIAL#	USERNAME	MACHINE	MODULE
5808	4hjxz687p2m9m	885	47663	JET		SQL Developer

DBMS_XPLAN formatted:

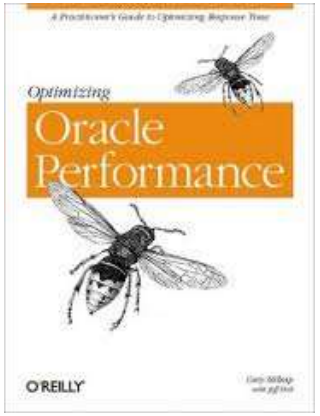
```
SQL_ID  4hjxz687p2m9m, child number 0
-------------
select * from                                    a,                                    b
where  a.START_DATETIME > to_date ('03/26/2017 07:39:00','mm/dd/yyyy
hh24:mi:ss') and a.START_DATETIME < to_date ('03/26/2017
10:44:00','mm/dd/yyyy hh24:mi:ss') and b.                    = '70050098'
and b.                  ='04' order by 3

Plan hash value: 2213452100

-----------------------------------------------------------------------------
| Id  | Operation                | Name                | Rows  | Bytes |TempSpc|
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                     |       |       |       |
|   1 |  SORT ORDER BY           |                     |  5575 |  1268K|  1608K|
|   2 |   MERGE JOIN CARTESIAN   |                     |  5575 |  1268K|       |
|*  3 |    TABLE ACCESS FULL     |                     |     7 |   602 |       |
|   4 |    BUFFER SORT           |                     |   832 |   119K|       |
|   5 |     PARTITION RANGE ALL  |                     |   832 |   119K|       |
|*  6 |      TABLE ACCESS FULL   |                     |   832 |   119K|       |
-----------------------------------------------------------------------------
```

Oracle Performance Tuning:

Cary Millsap and Method R provided a light at the end of the tunnel.

“Optimizing Oracle response time is, for the most part, a solved problem.”

Method R based on instrumentation provided via the Oracle Wait Interface.



Method R is a Repeatable Methodology!

No more silver bullets. You don’t need the Lone Ranger.

Method R emphasizes no it DEMANDS that you pay attention to pretty much only response time.

Method R: A Response Time–Based Performance Improvement Method That Yields Maximum Economic Value to Your Business

1. Select the user actions for which the *business* needs improved performance.
2. Collect properly scoped diagnostic data that will allow you to identify the causes of response time consumption for each selected user action while it is performing sub-optimally.
3. Execute the candidate optimization activity that will have the greatest net payoff to the business. If even the best net-payoff activity produces insufficient net payoff, then suspend your performance improvement activities until something changes.
4. Go to step 1.

Notes from Chapter 11:

Clues to the response time components that show up in resource profiles associate directly to instrumented kernel code

Tanel Poder: “It’s all 1 big C program (mostly)”

V$EVENT_NAME
+ CPU service (CPU TIME)
+ unaccounted for time (usually very minor)

Number of wait events in various Oracle releases?

7.3.4 106 wait events
10.0.1 500
11.2.0.4 1365 of them

Example 12-3. Resource profile for Oracle Purchasing program

Response Time Component	Duration		# Calls	Dur/Call
CPU service	1,527.5s	60.8%	158,257	0.009652s
db file sequential read	432.0s	17.2%	62,495	0.006913s
unaccounted-for	209.6s	8.3%		
global cache lock s to x	99.9s	4.0%	3,434	0.029083s
global cache lock open s	85.9s	3.4%	3,507	0.024502s
global cache lock open x	57.9s	2.3%	1,930	0.029990s
latch free	26.8s	1.1%	1,010	0.026505s
SQL*Net message from client	19.1s	0.8%	6,714	0.002846s
write complete waits	11.1s	0.4%	155	0.071806s
enqueue	11.1s	0.4%	330	0.033606s

Example 12-4. The Hotsos Profiler identifies the statement

SQL Statement Id	Duration	
704365403	1,066.4s	69.8%
3277176312	371.9s	24.3%

Method R summary:

Find out what you need to investigate for performance anomalies

Determine how you need to trace the critical sections of the problem

Turn on the tracing
Recreate the issue while trace is running
Turn off the trace

BINGO You have a medium to large file with lots and lots of wait event information and the SQL involved.

Put the trace file through some kind of resource profiler (tkprof / hotsos or method r profiler / MR trace / orasrp / others)

Investigate by looking at wait event diagnostics for things taking the most time (the most response time).

Top down approach: work on the items/symptoms/wait events causing the most delays (causing the longest response time).

Tim Hall: https://oracle-base.com/articles/misc/sql-trace-10046-trcsess-and-tkprof

It can be a little tricky getting a trace started and stopped at times. Lots of different ways to invoke it.

OEM and/or Toad and/or Cloud Control can be used (if access level is high enough) to invoke tracing.

Diagnostics and instrumentation invoked from the inside while the code is executing cause information to be written out to a trace file.

How do you digest the output of a trace file? Well you can eyeball it line by line but not recommended.

A resource profiler takes 10046 trace file output and helps make it actionable.

TKPROF or Method R Profiler … Toad even has something now? Free software OraSRP …

Triggers are evil?

Trick #1 … Sample login trigger to turn on trace …

```
CREATE OR REPLACE TRIGGER USER_TRACE_TRG
AFTER LOGON ON DATABASE
DECLARE
   v_program varchar2(48);
   v_audsid  NUMBER;
   CURSOR get_sid IS
      SELECT sid, serial#, osuser, machine, program FROM v$session WHERE audsid=v_audsid;
   v_sid_rec get_sid%ROWTYPE; -- v_sid NUMBER; v_serial NUMBER;
BEGIN
   v_audsid  := sys_context('USERENV','SESSIONID');
   OPEN   get_sid;
   FETCH  get_sid INTO v_sid_rec;
   IF get_sid%FOUND THEN
      v_program := v_sid_rec.program;
   END IF;
   CLOSE get_sid;
   IF ( instrv(v_program,'SQL*Plus') > 0 ) THEN -- instr(v_program,'ocpppp29') > 0
      -- IF SYS_CONTEXT('USERENV','HOST') = '*******' AND
            SYS_CONTEXT('USERENV','SESSION_USER') = '*******' THEN
      -- EXECUTE IMMEDIATE 'alter session set statistics_level=ALL';
      -- EXECUTE IMMEDIATE 'alter session set max_dump_file_size=UNLIMITED';
      EXECUTE IMMEDIATE 'alter session set events ''10046 trace name context forever, level
12''';
      -- sys.dbms_system.set_ev(v_sid,v_serial,10046,12,'');
  END IF;
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/

alter trigger SYS.USER_TRACE_TRG DISABLE;
```

Things have changed a lot since my Oracle early days

A lot of the heavy lifting involved in accessing relevant diagnostics has been internalized in the Oracle code base

Licensing: Diagnostics pack and Tuning pack

10g brings AWR and ADDM and ASH

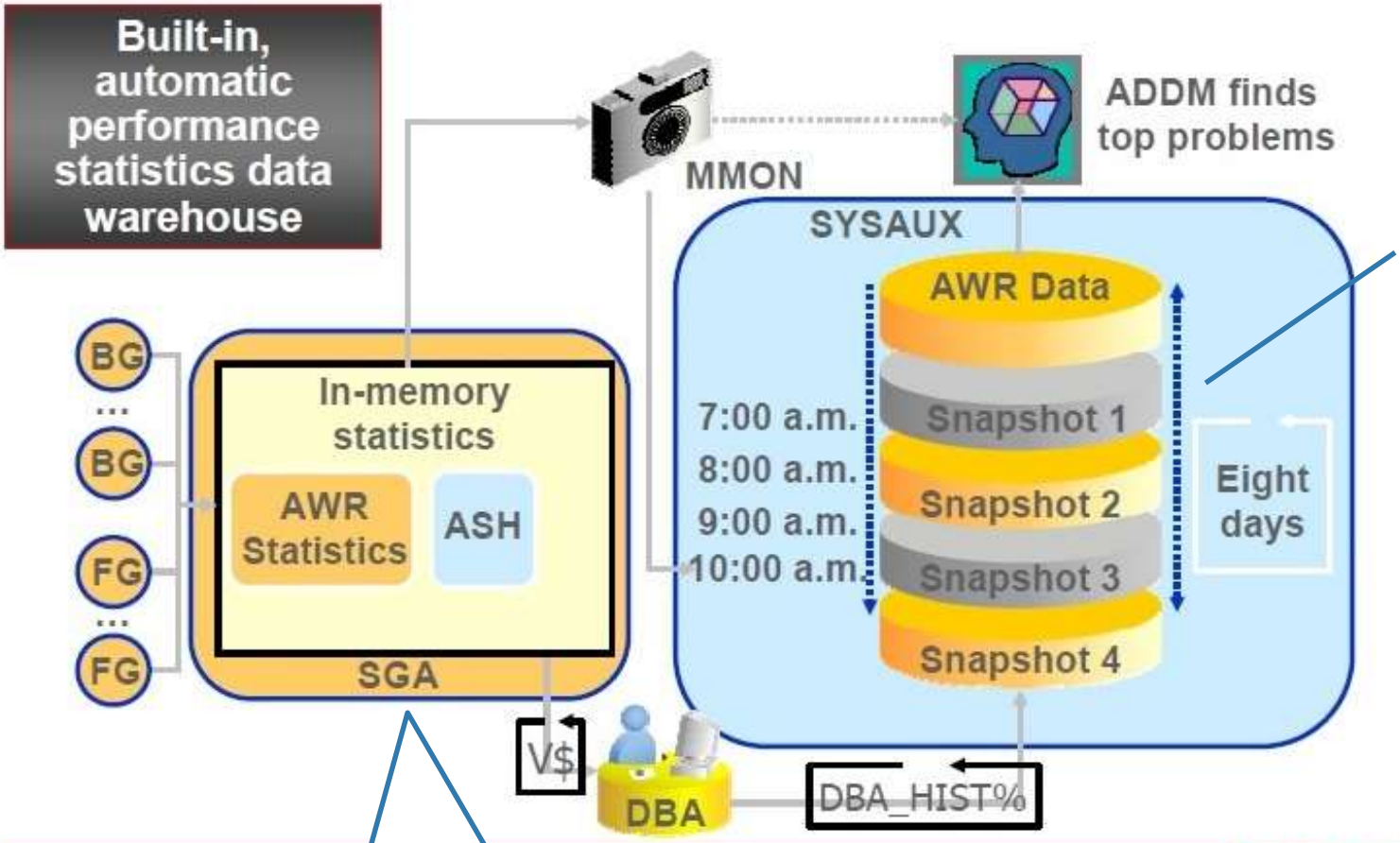
11g lots of improvements in AWR/ADDM/ASH

11g has RTM

RTM compared to a 10046 trace?

ASH compared to a 10046 trace?

Automatic Workload Repository (AWR)

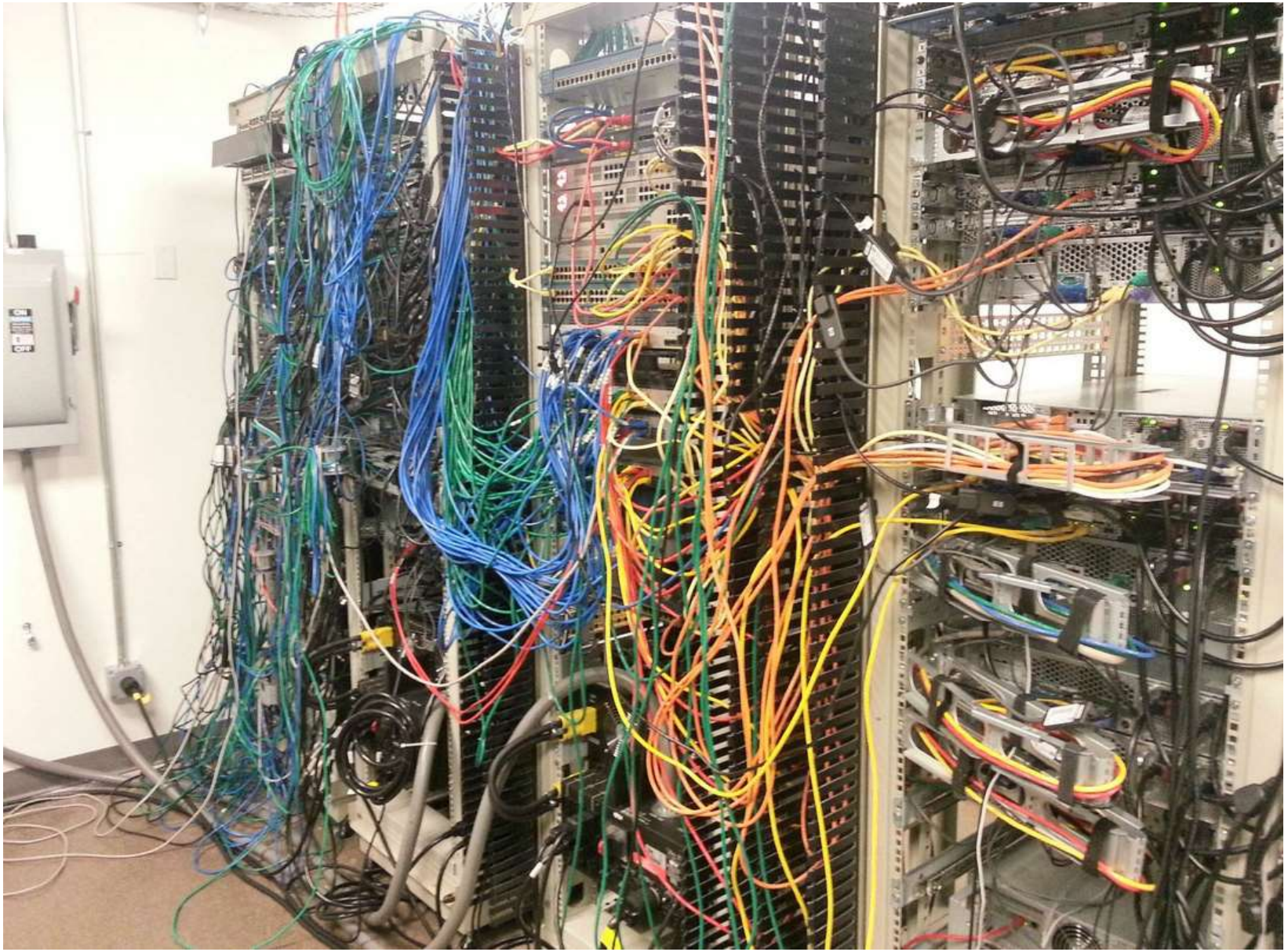


Let's change the retention and interval settings.

Interval will be set to 10 minutes and retention to 35 days (35x24x60 = 50400)

```
exec dbms_workload_repository.
modify_snapshot_settings
(interval => 10,
retention => 50400)
```

GLOC 2019



GLOC 2019

In 11g forward Oracle automatically monitors SQL statements if they are run in parallel, or consume 5 or more seconds of CPU or I/O in a single execution.

This allows resource intensive SQL to be monitored as it is executing, as well as giving access to detailed information about queries once they are complete.

SQL monitoring requires the STATISTICS_LEVEL parameter to be set to 'TYPICAL' or 'ALL', and the **CONTROL_MANAGEMENT_PACK_ACCESS** parameter set to **'DIAGNOSTIC+TUNING'.**

MONITOR Hint
The MONITOR hint will force on SQL monitoring for statements

```
SELECT /*+ MONITOR */ d.dname, WM_CONCAT(e.ename) AS employees
FROM   emp e
       JOIN dept d ON e.deptno = d.deptno
GROUP BY d.dname
ORDER BY d.dname;
```

Also if you have SQL statements you don't want to monitor
you can use the **NO_MONITOR** hint

Rows in v$sql_monitor may not be around for very long (my experience seems to be an hour at most often less) after monitoring has completed. While the status is executing you can look at RTM information while it is "changing".

select distinct status from v$sql_monitor

STATUS
DONE
DONE (ALL ROWS)
DONE (ERROR)
EXECUTING

With the right access you could "copy off" relevant rows from sql_monitor into your own table and research exciting items offline.

_sqlmon_binds_xml_format	n/a	○	format of column binds_xml in [G]V$SQL_MONITOR
_sqlmon_max_plan	○	○	Maximum number of plans entry that can be monitored. Defaults to 20 per CPU
_sqlmon_max_planlines	○	○	Number of plan lines beyond which a plan cannot be monitored
_sqlmon_recycle_time	n/a	○	Minimum time (in s) to wait before a plan entry can be recycled
_sqlmon_threshold	○	○	CPU/IO time threshold before a statement is monitored. 0 is disabled

I think (underscore sqlmon_recycle_time) defaults to 60 seconds is minimum retention time after execution ends.

alter session set "_sqlmon_max_planlines" = 500; ← 300 is default

Use RTM as a substitute for 10046 trace:

```
select * from v$sql_monitor
 where status = 'EXECUTING'
 group by sql_id order by 2 desc
```

```
SELECT substr(sq.sql_text,1,100) what_sql,
       vs.last_call_et,
       vs.sql_id,
       vs.sid, vs.serial#,
       vs.username, vs.machine, vs.module
  FROM v$session vs,
       v$sql sq
WHERE vs.status            = 'ACTIVE'
   AND vs.type            <> 'BACKGROUND'
   AND vs.schemaname      <> 'SYS'
   AND vs.sql_id           =  sq.sql_id
   AND vs.sql_child_number =  sq.child_number
```

```
SQL_ID          COUNT(*)
6txz4fyjkthc8      5
4zu3y492acc7k      4
a0y6psud2wxv3      4
26dzd78gvcms7      3
3zbqn76xdzjjx      2
4vatuf85avwgz      2
```

```
select sql_id, sid, session_serial# from v$sql_monitor
where status = 'EXECUTING' order by 1
```

```
/* Run via SQLPLUS */
SET LONG 1000000
SET LONGCHUNKSIZE 1000000
SET LINESIZE 1000
SET PAGESIZE 0
SET TRIM ON
SET TRIMSPOOL ON
SET ECHO OFF
SET FEEDBACK OFF

SPOOL c:\temp\report_sql_monitor4.htm

SELECT DBMS_SQLTUNE.report_sql_monitor(
  sql_id          => '6txz4fyjkthc8',
  session_id      => 1367,
  session_serial  => 11,
  type            => 'HTML',
  report_level    => 'ALL') AS report
FROM dual;
SPOOL OFF
```

```
type => 'HTML'
        'ACTIVE'
        'TEXT'
        'XML'
```

Global Information: EXECUTING

Instance ID	:	1
Session	:	(1367:11)
SQL ID	:	3d1f19229sk9d
SQL Execution ID	:	16777217
Execution Started	:	01/30/2017 15:18:34
First Refresh Time	:	01/30/2017 15:18:40
Last Refresh Time	:	01/30/2017 15:23:46
Duration	:	314s
Module/Action	:	JDBC Thin Client/-
Service	:	SYS$USERS
Program	:	JDBC Thin Client

Binds

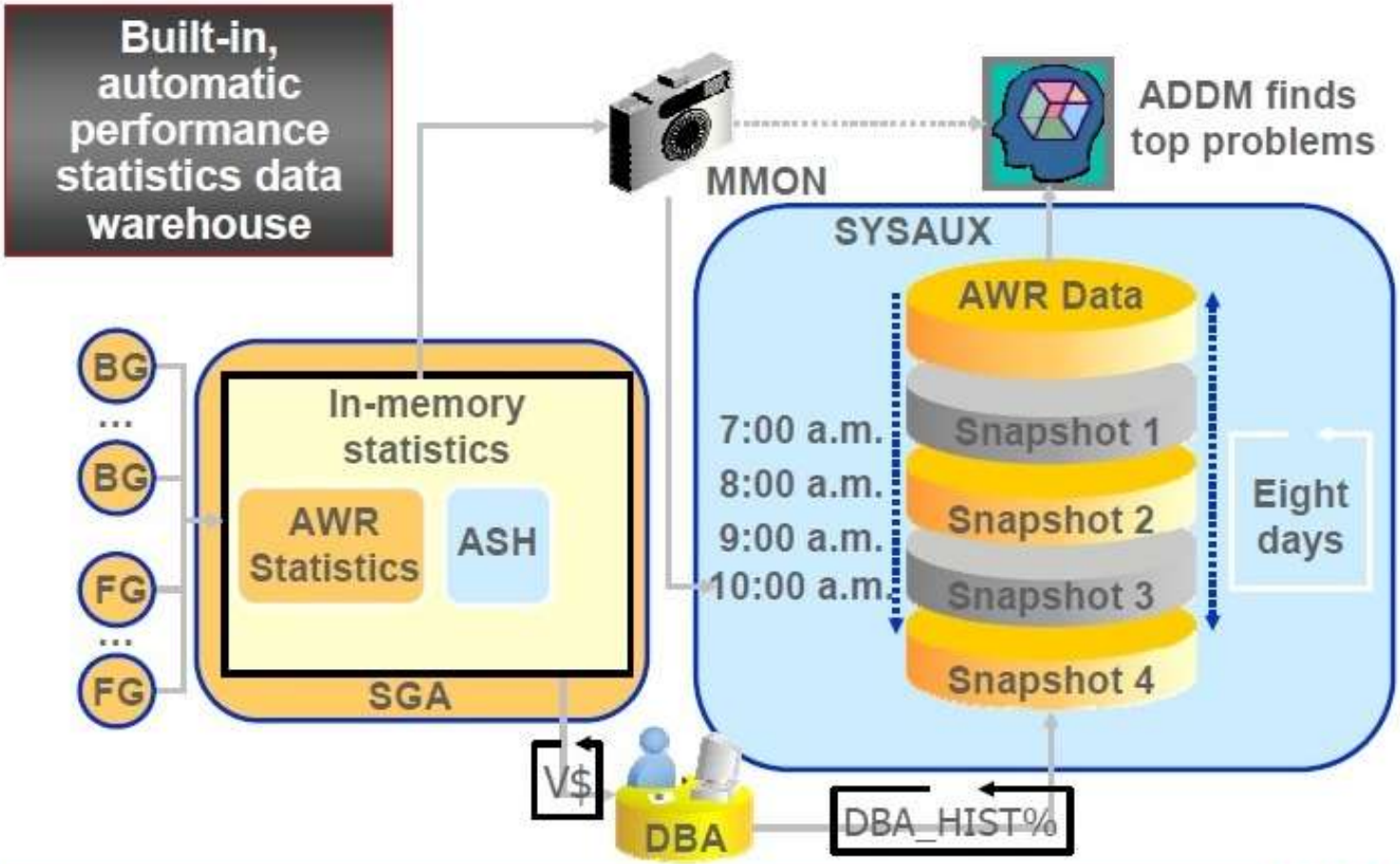
Name	Position	Type	Value
	2	DATE	01/27/2017 00:00:00
	3	DATE	01/27/2017 23:59:59
	4	DATE	01/27/2017 00:00:00
	5	DATE	01/27/2017 23:59:59

SQL Plan Monitoring Details (Plan Hash Value=800962418)

	Id	Operation
	0	SELECT STATEMENT
	1	SORT ORDER BY
	2	VIEW
	3	WINDOW SORT
	4	FILTER
	5	HASH JOIN RIGHT SEMI
	6	COLLECTION ITERATOR PICKLER FETCH
	7	NESTED LOOPS
	8	NESTED LOOPS
	9	HASH JOIN
	10	VIEW
	11	HASH JOIN
	12	HASH JOIN
	13	INDEX FAST FULL SCAN
	14	INDEX FAST FULL SCAN
	15	INDEX FAST FULL SCAN
->	16	NESTED LOOPS
	17	NESTED LOOPS
->	18	HASH JOIN
	19	TABLE ACCESS FULL
->	20	NESTED LOOPS
->	21	NESTED LOOPS
->	22	NESTED LOOPS
	23	TABLE ACCESS BY INDEX ROWID
->	24	INDEX UNIQUE SCAN
	25	TABLE ACCESS BY GLOBAL INDEX ROWID
	26	INDEX RANGE SCAN
	27	PARTITION RANGE ALL
->	28	INDEX RANGE SCAN
	29	TABLE ACCESS BY LOCAL INDEX ROWID
	30	PARTITION RANGE ITERATOR
	31	INDEX UNIQUE SCAN
	32	TABLE ACCESS BY LOCAL INDEX ROWID
->	33	INDEX UNIQUE SCAN
->	34	TABLE ACCESS BY GLOBAL INDEX ROWID

Estimated Rows	Cost	Active Period (314s)	Execs	Rows	Memory	Temp	IO Requests	CPU Activity	Wait Activity
			1						
28	452K		1						
28	452K		1						
28	452K		1						
			1						
28	452K		1	0	760.0KB				
408	30		1	1					
497	452K		1	2					
497	452K		1	7					
497	451K		1	19	1.3MB				
655	19		1	664					
			1	664					
			1	664					
655	4		1	664					
655	4		1	664					
655	13		1	664					
497	451K		1	19					
497	451K		1	144K				.34%	
497	450K		1	144K	1.6MB				
83	2		1	61					
502	450K		1	144K					
502	450K		1	144K					
502	279K		1	144K					
1	2		1	1					
1	1		1	1					
502	279K		1	144K				1.4%	
402K	1252		1	144K				.34%	
1	340		144K	144K				1.0%	
1	340		25M	144K				94%	100%
1	341		144K	144K				.34%	
1	1		144K	144K				.34%	
1	1		144K	144K				1.4%	
1	2		144K	19					
1	2		19	7			3 (60%)		
1	3		7	2			2 (40%)		

Automatic Workload Repository (AWR)



Aggregated information or session specific detailed information?

AWR reports work from between snapshots. They are “average” based they are aggregated information for the most part.

You can’t extrapolate detail from an aggregate. You cannot necessarily determine what’s wrong with an individual program by examining only the system-wide statistics for an instance.

ADDM is “AI” mined aggregated data targeted for identifying performance problems … sort of a resource profile for the database instance.

10046 is detail info resource profile makes it actionable.
RTM is detail information so actionable.

ASH is specific detailed information for sessions “doing things” if they are doing things (using CPU / waiting for something) when sampled. Then you get when data moved from memory to disk a sample of a sample.

Compare and Contrast on 10046 tracing versus AWR/ADDM/ASH

A 10046 trace works from the inside of the session being traced and writes information out to external file. It is comprehensive and catches all of the (currently instrumented) wait events.

It can be complicated and cumbersome to get the access and/or turn tracing on and off and/or get access to the trace file output.

RTM a lot like 10046 Trace but runs automagically.

A potential weakness using a 10046 trace is that while it shows “everything” that impacted your session it does not show “what impacts your session” had on other sessions.

Locking and blocking behavior may not be shown very clearly from a 10046 single trace especially across a busy OLTP system.

Compare and Contrast on 10046 tracing versus AWR/ADDM/ASH.

ADDM and AWR work automagically. They work from the inside of the database while trying to figure out whats going. Kind of like parents hosting a teenage party in the basement? Keeping an ear on the door? Watching who / what goes in and out?

ASH involves sampling of wait events. Not comprehensive but over a long enough sample set or a big problem probably may be good enough.

AWR gets a good picture over a period of time.

ADDM is the squeaky wheel the confidential informant.

ADDM perspective:

Run it first during problem investigations.

Run it periodically to help identify things that may deserve attention.

ADDM real world experience.

Use it with caution and skepticism. Often it is right on target but sometimes not worth considering the recommendations.

Most useful when you are familiar with application and the database characteristics.

AWR helps give details so many details about a time period being looked at.

Remember to concentrate on elapsed time for parts of the application important to the business to run fast.

AWR difference report can be used to compare how a system was performing between significant changes.

10046 tracing still can be useful at times but RTM and ASH and DBMS_XPLAN may be able to substitute.

```
Activity During the Analysis Period
-----------------------------------
Total database time was 30685 seconds.
The average number of active sessions was 50.72.

Summary of Findings
-------------------
   Description                                Active Sessions      Recommendations
                                              Percent of Activity
   ----------------------------------------   -------------------  ---------------
1  CPU Usage                                  39.04 | 76.98        2
2  Top SQL Statements                         32.15 | 63.38        5
3  "User I/O" wait Class                      14.03 | 27.66        0
4  Top Segments by "User I/O" and "Cluster"   2.82 | 5.56          1
5  PL/SQL Execution                           2.51 | 4.94          2


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


          Findings and Recommendations
          ----------------------------

Finding 1: CPU Usage
Impact is 38.98 active sessions, 76.98% of total activity.
----------------------------------------------------------
Host CPU was a bottleneck and the instance was consuming 100% of the host CPU.
All wait times will be inflated by wait for CPU.
Host CPU consumption was 100%.

   Recommendation 1: Host Configuration
   Estimated benefit is 39.04 active sessions, 76.98% of total activity.
   ---------------------------------------------------------------------
   Action
      Consider adding more CPUs to the host or adding instances serving the
      database on other hosts.
   Action
      Also consider using Oracle Database Resource Manager to prioritize the
      workload from various consumer groups.
```

Host CPU

CPUs	Cores	Sockets	Load Average Begin	Load Average End	%User	%System	%WIO	%Idle
24	12	4	71.90	82.69	69.9	30.1	0.0	0.0

Time Model Statistics

- Total time in database user-calls (DB Time): 26649.9s
- 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	26,315.74	98.75
DB CPU	7,110.79	26.68
PL/SQL execution elapsed time	894.67	3.36
parse time elapsed	62.35	0.23
hard parse elapsed time	40.49	0.15
connection management call elapsed time	4.12	0.02
hard parse (sharing criteria) elapsed time	3.52	0.01

```
Activity During the Analysis Period
-----------------------------------
Total database time was 1503 seconds.
The average number of active sessions was 2.09.

Summary of Findings
-------------------
   Description           Active Sessions      Recommendations
                         Percent of Activity
   ------------------    -------------------  ---------------
1  Top SQL Statements    1.24 | 59.33         5
2  PL/SQL Execution      .31 | 14.67          3


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


          Findings and Recommendations
          ----------------------------

Finding 1: Top SQL Statements
Impact is 1.24 active sessions, 59.33% of total activity.
---------------------------------------------------------
SQL statements consuming significant database time were found. These
statements offer a good opportunity for performance improvement.

   Recommendation 1: SQL Tuning
   Estimated benefit is .4 active sessions, 19.33% of total activity.
   ------------------------------------------------------------------
   Action
      Run SQL Tuning Advisor on the SELECT statement with SQL_ID
      "4uakphduw0w8k".
```

```
Recommendation 1: SQL Tuning
Estimated benefit is .4 active sessions, 19.33% of total activity.
-------------------------------------------------------------
Action
   Run SQL Tuning Advisor on the SELECT statement with SQL_ID
   "4uakphduw0w8k".
   Related Object
      SQL statement with SQL_ID 4uakphduw0w8k.
      SELECT * FROM SRE_CORP.VS_SKU WHERE IMAGE_RECNBR = :B1
Rationale
   The SQL spent 100% of its database time on CPU, I/O and Cluster waits.
   This part of database time may be improved by the SQL Tuning Advisor.
Rationale
   Database time for this SQL was divided as follows: 100% for SQL
   execution, 0% for parsing, 0% for PL/SQL execution and 0% for Java
   execution.
Rationale
   SQL statement with SQL_ID "4uakphduw0w8k" was executed 2873827 times and
   had an average elapsed time of 0.000037 seconds.
Rationale
   Top level calls to execute the PL/SQL statement with SQL_ID
   "3axj4zrja6ny2" are responsible for 75% of the database time spent on
   the SELECT statement with SQL_ID "4uakphduw0w8k".
   Related Object
      SQL statement with SQL_ID 3axj4zrja6ny2.
      BEGIN :1 := swingbench.storedprocedure3(:2 ,:3 ); END;
```

```
Recommendation 2: SQL Tuning
Estimated benefit is .22 active sessions, 10.67% of total activity.
---------------------------------------------------------------------
Action
   Investigate the PL/SQL statement with SQL_ID "39gfc6kr1xwm2" for
   possible performance improvements. You can supplement the information
   given here with an ASH report for this SQL_ID.
   Related Object
      SQL statement with SQL_ID 39gfc6kr1xwm2.
      BEGIN :1 := swingbench.storedprocedure6(:2 ,:3 ); END;
Rationale
   The SQL Tuning Advisor cannot operate on PL/SQL statements.
Rationale
   Database time for this SQL was divided as follows: 57% for SQL
   execution, 0% for parsing, 43% for PL/SQL execution and 0% for Java
   execution.
Rationale
   SQL statement with SQL_ID "39gfc6kr1xwm2" was executed 19225 times and
   had an average elapsed time of 0.016 seconds.
```

```
Recommendation 3: SQL Tuning
Estimated benefit is .45 active sessions, 7.54% of total activity.
-------------------------------------------------------------
Action
   Run SQL Tuning Advisor on the SELECT statement with SQL_ID
   "g3k6n2jy4g55c".
   Related Object
      SQL statement with SQL_ID g3k6n2jy4g55c.
      select a.REPORT_ID, a.REPORT_NAME, a.REPORT_TITLE, a.REPORT_TYPE,
      a.SHOW_IN_REPORT_LIST from                a,                  b,
                     c where a.REPORT_NAME = :1  and b.R                =
                       and a.REPORT_ID = b.REPORT_ID and UPPER(
      c.USER_LOGINNAME ) = UPPER( :2  )
Rationale
   The SQL spent 100% of its database time on CPU, I/O and Cluster waits.
   This part of database time may be improved by the SQL Tuning Advisor.
Rationale
   Database time for this SQL was divided as follows: 100% for SQL
   execution, 0% for parsing, 0% for PL/SQL execution and 0% for Java
   execution.
Rationale
   SQL statement with SQL_ID "g3k6n2jy4g55c" was executed 86 times and had
   an average elapsed time of 19 seconds.
```

An index on UPPER(USER_LOGINNAME) was added

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Startup Time	Release	RAC
[redacted]	[redacted]	[redacted]	1	22-Jan-17 02:01	11.2.0.4.0	NO

Host Name	Platform	CPUs	Cores	Sockets	Memory (GB)
[redacted]	Solaris[tm] OE (64-bit)	128	16	2	255.50

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	120506	17-Mar-17 10:10:17	876	19.1
End Snap:	120507	17-Mar-17 10:20:50	874	19.2
Elapsed:		10.55 (mins)		
DB Time:		86.39 (mins)		

Report Summary

Load Profile

	Per Second	Per Transaction
DB Time(s):	8.2	0.2
DB CPU(s):	4.0	0.1
Redo size (bytes):	596,055.7	15,339.6
Logical read (blocks):	188,901.1	4,861.4
Block changes:	3,647.3	93.9
Physical read (blocks):	15,253.4	392.6
Physical write (blocks):	682.5	17.6
Read IO requests:	7,277.4	187.3
Write IO requests:	560.3	14.4
Read IO (MB):	119.2	3.1
Write IO (MB):	5.3	0.1
User calls:	955.1	24.6
Parses (SQL):	250.0	6.4
Hard parses (SQL):	0.1	0.0
SQL Work Area (MB):	51.6	1.3
Logons:	1.2	0.0
Executes (SQL):	1,250.5	32.2
Rollbacks:	0.1	0.0
Transactions:	38.9	

Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
DB CPU		2503.2		48.3	
db file sequential read	4,029,110	1706.7	0	32.9	User I/O
read by other session	1,907,513	868	0	16.7	User I/O
latch: cache buffers chains	2,104,512	183	0	3.5	Concurrency
direct path read	37,581	181.4	5	3.5	User I/O
log file sync	22,531	45.5	2	.9	Commit
db file parallel read	12,382	31.3	3	.6	User I/O
library cache: mutex X	52,447	4.2	0	.1	Concurrency
enq: TX - index contention	561	2.8	5	.1	Concurrency
db file scattered read	748	1.3	2	.0	User I/O

Wait Classes by Total Wait Time

Wait Class	Waits	Total Wait Time (sec)	Avg Wait (ms)	% DB time	Avg Active Sessions
User I/O	5,991,958	2,790	0	53.8	4.4
DB CPU		2,503		48.3	4.0
System I/O	106,569	219	2	4.2	0.3
Concurrency	2,166,508	193	0	3.7	0.3
Commit	22,533	45	2	.9	0.1
Other	5,065	2	0	.0	0.0
Network	487,912	1	0	.0	0.0
Application	101	0	2	.0	0.0
Configuration	7	0	1	.0	0.0

AWR stuff …

It's fairly straight forward most of the time prioritizing and attacking an AWR report ..

Look for SQL using the most Elapsed Time.

Look for SQL using the most CPU time

Look for SQL doing the most logical reads

Look for SQL doing the most physical reads

For SQL that looooooks expensive

Contrast it with how many times it was executed

SQL ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100
- %Total - Elapsed Time as a percentage of Total DB time
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Captured SQL account for 96.1% of Total DB Time (s): 26,650
- Captured PL/SQL account for 21.0% of Total DB Time (s): 26,650

Elapsed Time (s)	Executions	Elapsed Time per Exec (s)	%Total	%CPU	%IO	SQL Id	SQL Module
7,468.54	6	1,244.76	28.02	27.41	26.08	dzr6sa3gcrbpk	JDBC Thin Client
4,186.65	48	87.22	15.71	23.05	45.77	fd6gddbw4nxzk	JDBC Thin Client
3,949.73	6	658.29	14.82	21.26	60.93	5634b4d50fn14	JDBC Thin Client
2,370.64	48	49.39	8.90	22.03	47.35	c4v9h4fhzhxv7	JDBC Thin Client
1,819.45	49	37.13	6.83	24.39	43.78	95xg0brm00v9p	JDBC Thin Client
1,501.44	15	100.10	5.63	31.69	0.00	b083as1jmrpmt	JDBC Thin Client
1,451.35	1	1,451.35	5.45	19.24	68.94	4fb67h7qs8kpn	JDBC Thin Client
719.75	6,716	0.11	2.70	58.67	0.01	cj9ydsx25tkq0	JDBC Thin Client
631.97	1	631.97	2.37	24.81	37.58	9nv0svfhka1a4	u (TNS V1-V3)
607.31	0		2.28	26.37	28.21	gfv7sh4gp0uv3	JDBC Thin Client
606.73	0		2.28	15.54	73.48	4a4gn6vapy90t	JDBC Thin Client
606.38	0		2.28	13.79	81.93	85vrx4zx09b3v	JDBC Thin Client
606.26	0		2.27	26.57	25.23	19h09bfyzg53x	DBMS_SCHEDULER
550.58	22	25.03	2.07	21.69	14.92	9ft1pqwnc8pns	JDBC Thin Client
479.81	3	159.94	1.80	24.56	16.35	3b1hr9jfnw48u	DBMS_SCHEDULER
372.06	186,222	0.00	1.40	23.58	15.05	g3vncnz9qd7p0	DBMS_SCHEDULER
336.22	17	19.78	1.26	35.71	0.00	8p529w0ymnfpq	(TNS V1-V3)
322.06	2	161.03	1.21	32.10	0.01	4cc6ncyvq7br7	JDBC Thin Client
315.92	2,767	0.11	1.19	55.40	0.00	cwgtargc65kqg	JDBC Thin Client
285.55	14	20.40	1.07	34.47	0.00	09scdtr66qcjd	(TNS V1-V3)

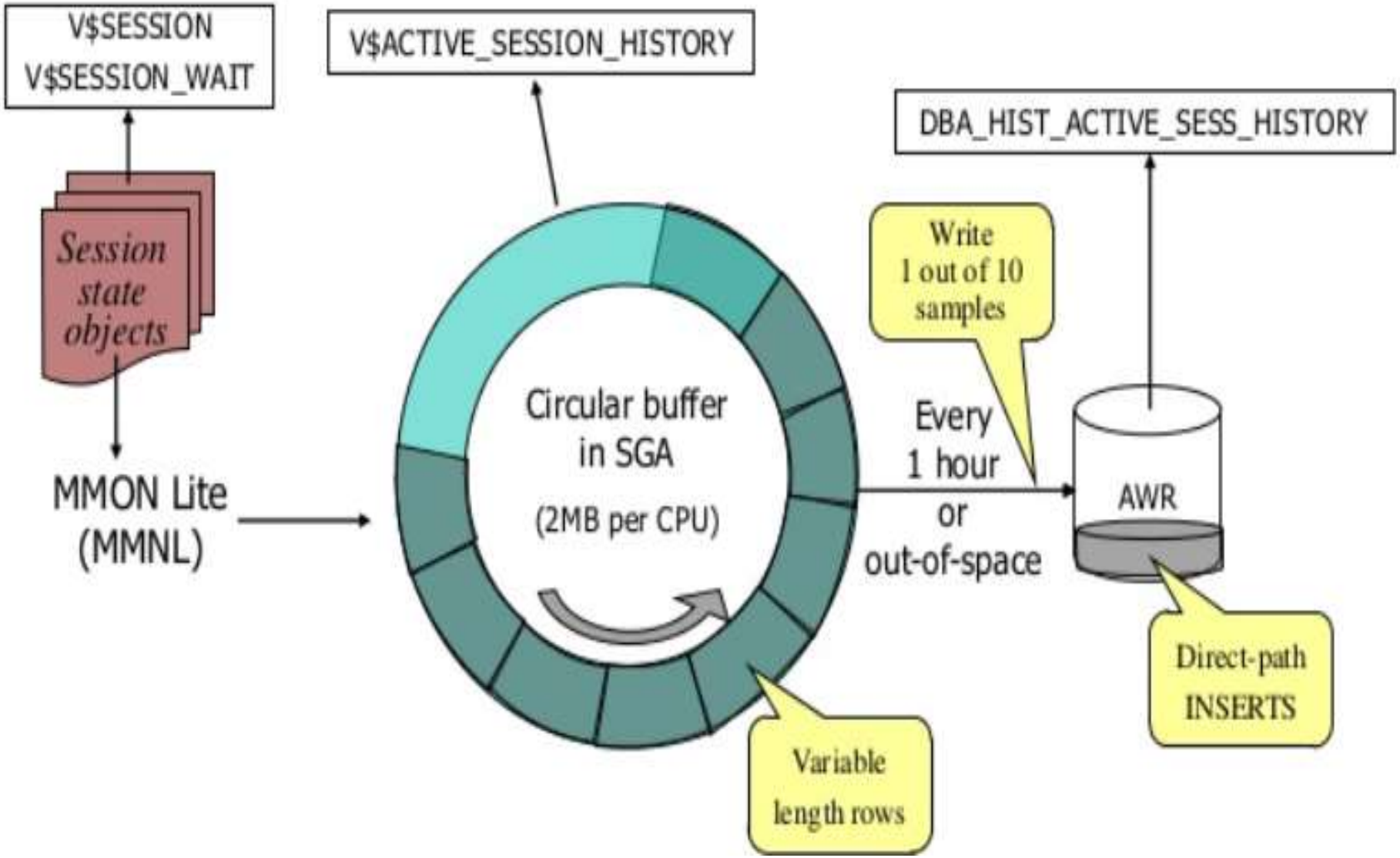
SQL ordered by Gets

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the
- Total Buffer Gets: 303,242,963
- Captured SQL account for 60.3% of Total

Buffer Gets	Executions	Gets per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id
12,109,487	3	4,036,495.67	3.99	458.46	608.56	1g67j0g8qsbc5
11,897,530	7	1,699,647.14	3.92	332.04	574.79	0rbycbsqtu46k
9,530,239	552	17,264.93	3.14	62.84	239.80	7vvgka8xgh0n8
6,250,377	1,562,533	4.00	2.06	70.61	71.43	gv7fvasgvsh03
6,198,227	1,239,572	5.00	2.04	251.42	666.86	7bjmtz515bfpn
6,119,456	10	611,945.60	2.02	963.30	1763.42	c514xam8n540a
6,078,949	45,421	133.84	2.00	11.61	11.31	fhsx4p4r8pc6p
5,640,106	264,237	21.34	1.86	32.28	88.82	528dn8ctbjwhs

Active Session History:

```
SELECT substr(sq.sql_text,1,100) what_sql,
       vs.last_call_et,
       vs.sql_id,
       vs.sid, vs.serial#,
       vs.username, vs.machine, vs.module
  FROM v$session vs,
       v$sql sq
WHERE vs.status             = 'ACTIVE'
   AND vs.type             <> 'BACKGROUND'
   AND vs.schemaname       <> 'SYS'
   AND vs.sql_id            =  sq.sql_id
   AND vs.sql_child_number =  sq.child_number
```


V$SESSION
V$SESSION_WAIT
V$ACTIVE_SESSION_HISTORY
DBA_HIST_ACTIVE_SESS_HISTORY
Session state objects
MMON Lite (MMNL)
Circular buffer in SGA
(2MB per CPU)
Write 1 out of 10 samples
Every 1 hour or out-of-space
AWR
Direct-path INSERTS
Variable length rows

There are so many GREAT ASH presentations around but some of the ones to not miss include:

http://www.ooug.org/wp-content/uploads/2016/05/Tanel_Poder_Active_Session_History_seminar.pdf

Aka Getting the Most out of Oracle's Active Session History

Tanel Poders ASH session snapper
http://tech.e2sn.com/oracle-scripts-and-tools/session-snapper

Tim Gormans RDBMS Forensics Troubleshooting Using ASH

http://evdbt.com/download/presentation-rdbms-forensics-troubleshooting-using-ash/

This presentation should make you think of Active Session History as something like queryable trace information captured and stored within the database.

Plus of course anything by Ric Van Dyke!

I find the ASH script I tend to use the most just looks for blockers and waiters:

```
SELECT ash_data.*, substr(sqlinfo.sql_text,1,70) FROM
(SELECT to_char(ash.sample_time,'MM/DD/YYYY HH24:MI:SS') when_time,  count(*)
        sessions_blocked, ash.event, ash.blocking_session,
        ash.blocking_session_serial#, ash.sql_id, ash.sql_opname
   FROM DBA_HIST_ACTIVE_SESS_HISTORY ash
  WHERE ash.SAMPLE_TIME >= to_date('11/28/2017 08:00','MM/DD/YYYY HH24:MI')
    and ash.sample_time <= to_date('11/29/2017 11:30','MM/DD/YYYY HH24:MI')
    -- and ash.event not like 'read by other session%'
    and blocking_session is not null
  GROUP BY to_char(ash.sample_time,'MM/DD/YYYY HH24:MI:SS'), ash.event,
           ash.sql_id,
           ash.sql_opname, ash.blocking_session, ash.blocking_session_serial#
  ORDER BY 1) ash_data, v$sqlarea sqlinfo
WHERE ash_data.sql_id = sqlinfo.sql_id
  AND sessions_blocked >= 1
 ORDER BY when_time desc
```

WHAT_TIME	BLOCKED	EVENT	SID	SERIAL#	SQL_ID	SQL_OPNAME	First 70 character
03/26/2017 10:25:04	2	enq: TM - contention	2152	3493	07s2d0hkktx07	INSERT	INSERT INTO CA_
03/26/2017 10:25:04	5	enq: TM - contention	2152	3493	55wvs4ynbp29y	UPDATE	UPDATE CA_PC_P
03/26/2017 10:25:04	2	enq: TM - contention	2152	3493	3s30h88gyqn09	INSERT	insert into CA_PC
03/26/2017 10:25:04	1	enq: TM - contention	1160	48891	5fka9649ahhnp	LOCK TABLE	LOCK TABLE "PAY
03/26/2017 10:25:14	6	enq: TM - contention	2152	3493	07s2d0hkktx07	INSERT	INSERT INTO CA_
03/26/2017 10:25:14	1	enq: TM - contention	1160	48891	5fka9649ahhnp	LOCK TABLE	LOCK TABLE "PAY
03/26/2017 10:25:14	10	enq: TM - contention	2152	3493	55wvs4ynbp29y	UPDATE	UPDATE CA_PC_P
03/26/2017 10:25:14	7	enq: TM - contention	2152	3493	3s30h88gyqn09	INSERT	insert into CA_PC
03/26/2017 10:25:26	10	enq: TM - contention	2152	3493	55wvs4ynbp29y	UPDATE	UPDATE CA_PC_P
03/26/2017 10:25:26	9	enq: TM - contention	2152	3493	3s30h88gyqn09	INSERT	insert into CA_PC
03/26/2017 10:25:26	1	enq: TM - contention	1160	48891	5fka9649ahhnp	LOCK TABLE	LOCK TABLE "PAY
03/26/2017 10:25:26	11	enq: TM - contention	2152	3493	07s2d0hkktx07	INSERT	INSERT INTO CA_
03/26/2017 10:25:37	11	enq: TM - contention	2152	3493	3s30h88gyqn09	INSERT	insert into CA_PC
03/26/2017 10:25:37	10	enq: TM - contention	2152	3493	55wvs4ynbp29y	UPDATE	UPDATE CA_PC_P
03/26/2017 10:25:37	1	enq: TM - contention	1203	10071	d4ns3cttc71tz	LOCK TABLE	LOCK TABLE "PAY
03/26/2017 10:25:37	13	enq: TM - contention	2152	3493	07s2d0hkktx07	INSERT	INSERT INTO CA_
03/26/2017 10:25:47	4	enq: TM - contention	2152	3493	55wvs4ynbp29y	UPDATE	UPDATE CA_PC_P
03/26/2017 10:25:47	1	enq: TM - contention	1160	48891	5fka9649ahhnp	LOCK TABLE	LOCK TABLE "PAY
03/26/2017 10:25:57	4	enq: TM - contention	2152	3493	07s2d0hkktx07	INSERT	INSERT INTO CA_

ASH … based on sampling ..

It is not comprehensive EVERY EVENT for EVERY SESSION … just what is caught doing something at the time of sampling.

Over time though the same guilty parties probably will show up on a typical system. The more badly behaved a problem is the more often evidence pointing to the guilty parties will get collected from the sampling.

You can build a resource profile of a SQL statement execution (if it runs long enough) and see what events are impacting it etc.

Over time and given enough executions of it … a sampled ASH resource profile is probably good enough to emulate many parts of what you would have seen from a 10046 trace.

Not enough time but people have built "10046 trace resource profiler" tools using ASH data.