What ASH is used for in Oracle database?

ORACLE®

Mikhail Bazgutdinov

Agenda

- What is ASH? Why ASH?
- Interpreting ASH data using ASH reports and raw ASH data
- ASH analytics in Cloud Control
- Some issues with ASH data (avoiding mistakes)
- 12c enhancements to ASH

What is ASH?
Why ASH?

ORACLE®

Graham Wood

Here is Graham Wood, the "father" of AWR and ASH.

Works for Oracle, in the "Real-world performance group“. The center of expertise focused on solving DB performance issues and publishing "Best practices“ videos and articles.

Real-world performance Learning library

When do we need ASH?

Manager asks … WHY ?!

- **Response is slow**
- **Database is hung**
- **Batch job behind schedule**

Need answers fast !

DB Time Concepts

FOREGROUND ONLY

- 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

Why ASH?

Oracle RDBMS has rich performance diag toolbox: **GV$, AWR, SQL Monitoring.**

- **GV$:** either current state or cumulative values of metrics since instance start.
  - Exception: gv$sysmetic_history – 15/60 second averages for some perf counters
    - 47 metrics for 15-sec intervals; 161 metrics for 60-sec intervals
- **AWR**: Timed statistics and performance metrics summed/averaged over the snapshot intervals which are 10-60 minutes.
- **Active Session History:** 1-sec sampling. Misses a lot of short operations but reflects well the workload every second.
- **SQL Monitor:** Very detailed and visual info for long-running SQLs only (current and historical). Includes real bind values.
  - Relies on ASH for some metrics

Questions answered by ASH

-Some questions are harder (if possible) to answer using **AWR/GV$**

- -How DB time was spread over clock time?
- -How long was the user experience for SQL response time (GV$SQL includes only time spend in DB calls)?
- -Which app component is the source of the issue (machine/module/action)?
  - -We can isolate the source of the issue using many other dimension
- -Which was the blocking session?
- -Find out distinct SQLs with the same exec plan.
- -Which session consumed most of IO/PGA/TEMP/Interconnect bandwidth?
- -What was the activity prior to instance crashed (after the last AWR snapshot was collected)?

How much does it cost?

ASH is licensed as part of the Diagnostic pack over Enterprise Edition.

Not available for Standard Edition.

Note: in SE the only AWR tables which are populated are dba_hist_sysmetric_history.

no ASH in SE.

How are ASH data being collected?

ASH is sampled by MMNL (MMON lightweighted) and MMON itself.

Every 1 seconds to memory, every 10 seconds - to disk.

GV$ACTIVE_SESSION_HISTORY / DBA_HIST_ACTIVE_SESS_HISTORY

ASH columns may be unknown at sampling time
SQL_PLAN_HASH_VALUE: session is still optimizing SQL

Long Events
Events that are longer than 1 sec is always sampled
Event may be sampled multiple times
TIME_WAITED fixed up in first sample after event completes
Only final ("fixed-up") ASH row for long events has TIME_WAITED > 0.
All others stay 0.
Querying current ASH may return un-fixed rows (should not be a problem generally)

Fix-up TIME_WAITED

```
select session_id,to_char(sample_time,'hh24:mi:ss') "Time", event,seq#,
time_waited/1000/1000 "Waited, sec"
from v$active_session_history where session_id=1409 -- and event='log file parallel write'
and sample_time>sysdate-1
```

Output | Query Result

SQL | Fetched 150 rows in 0,094 seconds

SESSION_ID	Time	EVENT	SEQ#	Waited, sec
1409	01:49:02	log file parallel write	44453	0.000484
1409	01:58:46	log file parallel write	46272	0.001449
1409	02:03:46	log file parallel write	47466	0.01032
1409	02:04:32	log file parallel write	47593	0.001909
1409	02:05:23	log file parallel write	47652	0.001294
1409	02:05:28	log file parallel write	47690	0.001325
1409	02:05:49	log file parallel write	48046	0.00045
1409	02:09:36	log file parallel write	49617	0
1409	02:09:37	log file parallel write	49617	2.211248
1409	02:14:12	log file parallel write	49930	0.000489
1409	02:17:31	log file parallel write	50105	0.00042
1409	02:19:13	log file parallel write	51047	0.000502
1409	02:20:50	log file parallel write	51179	0.001603
1409	02:24:02	log file parallel write	51478	0.000562
1409	02:28:06	log file parallel write	52214	0.000594
1409	02:30:23	log file parallel write	52566	0.000854

- Событие с порядковым номером 49617 длилось более 2 сек. Попало в 2 сэмпла. 1-й – с нулем сек, последний – с актуальной длительностью.

ASH In-memory

- "Active" sessions sampled every 1 second
  - Direct session state access and callbacks
  - Active = on CPU or in non-idle Wait
  - Sampler itself (MMNL session) is not visible in ASH. It's the only one who hides from ASH.
- Circular memory buffer
  - Fixed footprint => variable time range
  - Design goals: one hour activity and <5% of shared pool, Absolute max is 256 Mb.
- Non-latching queries
  - Readers/writer go in opposite directions
- Wait times are "fixed up"
  - But don't be tempted to use them incorrectly!

ASH On-disk

- 1-in-10 samples persisted with AWR snapshots
- DBA_HIST_ACTIVE_SESS_HISTORY
  - Partitioned by DBID, SNAP_ID
  - Emergency flush under buffer pressure (ASH memory buffer is 2/3 full)
- AWR retention 7 days by default
- Query snapshot ranges from DBA_HIST_SNAPSHOT
  - For partition elimination always filter on 3 columns:
    SNAP_ID between 100 and 200 AND instance_number=1 AND DBID=(select dbid from v$database)

Where ASH data are stored?

SELECT inst_id, oldest_sample_time, sysdate - oldest_sample_time ash_in_memory

FROM **gv$ash_info**;

```
   INST_ID OLDEST_SAMPLE_TIME                  ASH_IN_MEMORY
---------- ---------------------------------- --------------------
         1 24-JUL-17 10.00.01.886000000 AM  +000000001 05:32:14
```

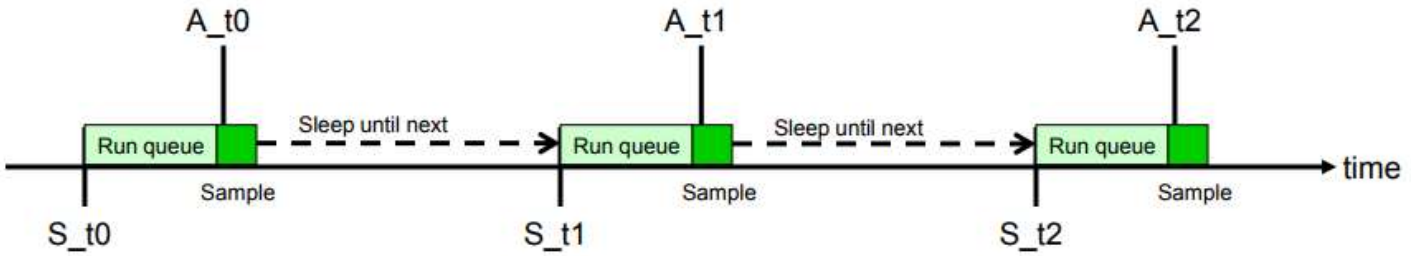
```
select bytes/(1024*1024) MB from v$sgastat where name like
'ASH buffers';
MB
----------
29
```

Session waits stored in V$-views and table

Current	**V$session_wait** Current session waits
Recent	**V$session_wait_history** Last 10 waits per session
30 Minutes .. few hours	**V$active_session_history** Polling at 1 second
Last 7 days (or custom retention) **in AWR**	**wrh$_active_session_history** **dba_hist_active_sess_history** (1 in 10 values from v$active_session_history)

Why does ASH work when the server is CPU bound?

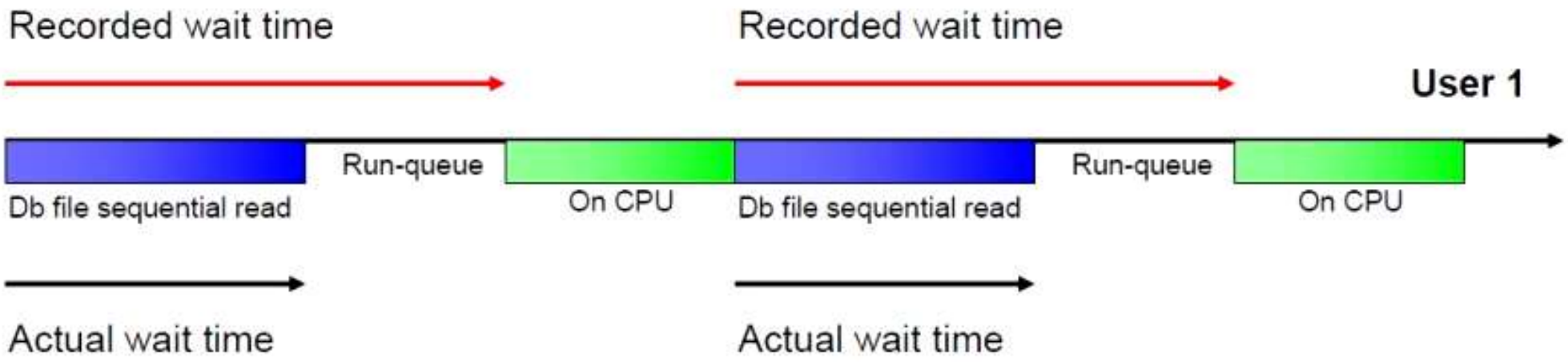
1. ASH sampler is very efficient, and does not lock. Therefore, in almost all cases it takes a single CPU slice to finish a full sample.
2. After a sample is done, the sampler computes next scheduled sample time and sleeps until then
3. Upon scheduled wake-up, it waits for CPU (runq) and samples again
4. Thus, CPU bound samples are shifted by one runq but stay about 1 per second



If run queue times are consistent sampling interval will be preserved but sample times shifted

CPU Run-queue and wait latencies

1. Event wait time is increased by the wait time in CPU runqueue.
2. Unfortunately it is not possible to track the runqueue itself as a separate wait because process is stuck while it is in runqueue and is not able to stop the timer which measures previous waitevent duration.



How to configure ASH data collection?

_ash_enable	TRUE	To enable or disable Active Session sampling and flushing
_ash_size	1048618	To set the size of the in-memory Active Session History buffers
_ash_compression_enable	TRUE	To enable or disable string compression in ASH
_ash_disk_filter_ratio	10	Ratio of the number of in-memory samples to the number of samples actually written to disk
_ash_disk_write_enable	TRUE	To enable or disable Active Session History flushing
_ash_eflush_trigger	66	The percentage above which if the in-memory ASH is full the emergency flusher will be triggered
_ash_min_mmnl_dump	90	Minimum Time interval passed to consider MMNL Dump
_ash_sample_all	FALSE	To enable or disable sampling every connected session including ones waiting for idle waits **Customer case for 18c**: this helped to include some missing sessions into ASH
_ash_sampling_interval	1000	Time interval between two successive Active Session samples in millisecs

- Short recommendation: do not change these parameters.

ASH Pros and Cons

PROS

- Supports the DB Time method of performance analysis
  - Historically for large or regularly occured problems
  - Recently or “now” (V$SESSION still available) for emergency
- Always available (licensing restrictions)
- Minimal cost to server performance

CONS

- We query a sample and not the full data set
- Difficult to form semantically meaningful queries
- Probability distribution of a query result matters
- short retention time for 1-seconds samples

  Hint: create table ASH as SELECT * FROM V$ACTIVE_SESSION_HISTORY;

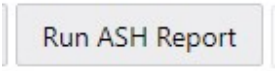
Interpreting ASH data

ORACLE®

How to build ASH report?

What is you favorite tool to get an ASH report?

OEM -> Performance -> Performace Home -> "Run ASH Report".



Putty -> $ORACLE_HOME/rdbms/admin/ashrpt.sql (ashrpti.sql for RAC)

SQL Developer -> DBA tab -> choose connection -> Performance - > ASH report Viewer

Some open-source and commercial 3rd party tools named like "ASH Viewer"
https://sourceforge.net/projects/ashv/

OEM. You can filter data for the report. (Filter by SQL_ID as an example).

Other filters are: SID, Wait class, Service, Module, Action, Client.

How to read ASH report? - Essentials in the caption

- Data Source - V$ or DBA_HIST
- Elapsed time - clock time
- Average Active Session - some kind of db time
- % of total db activity
- Many "Top …" sections

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
MORPHEUS	4127582969	morpheus	1	11.2.0.4.0	NO	anna.testdomain.ru

CPUs	SGA Size	Buffer Cache	Shared Pool	ASH Buffer Size
2	597M (100%)	396M (66.3%)	176M (29.5%)	4.0M (0.7%)

	Sample Time	Data Source
Analysis Begin Time:	28-Nov-17 12:09:53	V$ACTIVE_SESSION_HISTORY
Analysis End Time:	28-Nov-17 12:18:53	V$ACTIVE_SESSION_HISTORY
Elapsed Time:	9.0 (mins)	
Sample Count:	643	
Average Active Sessions:	1.19	
Avg. Active Session per CPU:	0.60	
Report Target:	SQL_ID like 'dskp9f0r23dj3'	98.2% of total database activity

How to read ASH report? - Different "Top's"

- As an example - Top sessions

Top Sessions

- '# Samples Active' shows the number of ASH samples in which the session was found waiting for that particular event. The
- 'XIDs' shows the number of distinct transaction IDs sampled in ASH when the session was waiting for that particular event
- For sessions running Parallel Queries, this section will NOT aggregate the PQ slave activity into the session issuing the PQ

Sid, Serial#	% Activity	Event	% Event	User	Program	# Samples Active	XIDs
156, 115	83.83	CPU + Wait for CPU	83.83	ASH	sqlplus@anna.t...u (TNS V1-V3)	539/540 [100%]	0
149, 771	9.02	CPU + Wait for CPU	9.02	ASH	sqlplus@anna.t...u (TNS V1-V3)	58/540 [11%]	0
36, 23	7.15	CPU + Wait for CPU	7.15	ASH	sqlplus@anna.t...u (TNS V1-V3)	46/540 [9%]	0

How to read ASH report? - Wait events

- As an example - Top waitevents.

Top User Events

Event	Event Class	% Event	Avg Active Sessions
row cache lock	Concurrency	54.55	352.53
library cache pin	Concurrency	31.38	202.80
library cache lock	Concurrency	9.48	61.27
enq: TX - index contention	Concurrency	3.88	25.07

- NB: there is no information about wait total duration and average wait time.
- Compare to AWR:

Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
library cache pin	663,203	1573.8K	2373	34.9	Concurrency
DB CPU		730.2K		16.2	
library cache lock	90,292	430.1K	4764	9.5	Concurrency
row cache lock	150,950	289.5K	1918	6.4	Concurrency
enq: SQ - contention	7,348	172.4K	23465	3.8	Configuration

How to read ASH report? - Wait events - Parameters

Top Event P1/P2/P3 Values

Event	% Event	P1 Value, P2 Value, P3 Value	% Activity	Parameter 1	Parameter 2	Parameter 3
row cache lock	54.55	"13","0","5"	54.54	cache id	mode	request
library cache pin	31.52	"5256317615496","5241691673728","3681830649856002"	0.02	handle address	pin address	100*mode+namespace
library cache lock	9.51	"5564666640808","5231897287152","3211267"	0.03	handle address	lock address	100*mode+namespace
enq: TX - index contention	3.88	"1415053316","197328909","1027724"	1.42	name|mode	usn<<16 | slot	sequence

% of this event in the ASH report

% of these P1/P2/P3 set in the ASH report

- Row cache lock occurred with the same parameters ("hot" cache_id - 13).
- Library cache pin and library cache lock occurred with different parameters (top parameters gives only 0.02% of total), so no single "hot" object.
- This information is missing in AWR report.
- Translation of P1/P2/P3 to "hot" object name depends on Event.

How to read ASH report? - Events parameters to objects.

- Translation of P1/P2/P3 to "hot" object name depends on Event.

You may use some SQL queries published on MOS or some OUG blogs.

Remember: script may depend on Oracle version and often requires **X$** views.

Example: library cache: mutex X - many concurrent sessions executes the same trigger (on massive INSERTs) and each session pins it in the library cache.

Solution:

```
select
    OWNER, name, LOCKED_TOTAL "Locked", PINNED_TOTAL "Pinned", FULL_HASH_VALUE
from v$db_object_cache
where HASH_VALUE={P1 value};
```

```
alter system set "_kgl_debug"="hash='c4982a06e69512c0348c4331433f51de' debug=33554432";
alter system set _kgl_hot_object_copies=128;
```

Длинный хеш для _kgl_debug - поле FULL_HASH_VALUE. 33554432 - магическое число.

How to read ASH report? - Top SQLs'

- SQL commands related to top waitevents

Top SQL with Top Events

SQL ID	Planhash	Sampled # of Executions	% Activity	Event	% Event	Top Row Source	% RwSrc	SQL Text
3xcgx55t55juc	379421932	1	64.66	CPU + Wait for CPU	40.00	TABLE ACCESS - FULL	40.00	select /*+ LEADING(A) USE_NL(B...
				direct path read	24.66	TABLE ACCESS - FULL	24.66	
7wwa8agka0xg8	2139464690	2	32.33	CPU + Wait for CPU	21.92	TABLE ACCESS - FULL	21.92	select /*+ LEADING(A) USE_NL(B...
				direct path read	10.41	TABLE ACCESS - FULL	10.41	

Count of distinct SQL executions

Top SQL PLAN operation and it's part in the total activity.

- This information is missing in AWR report
  - for heavy queries AWR report can contain "0" Executions
- ASH report does not have "elapsed time", "number of execs" for SQL. See AWR.
- ASH shows the top SQL PLAN step, but it only displays the operation name, although the v$ASH contains also SQL_PLAN_LINE_ID column which refers to the current line number in the execution plan.

How to read ASH report? - Top Objects, Top Latches

- Top objects accessed (available for some waitevents only)

Top DB Objects

- With respect to Application, Cluster, User I/O and buffer busy waits only.

Object ID	% Activity	Event	% Event	Object Name (Type)	Tablespace
1641756	21.47	buffer busy waits	21.46	ODDO.SYS_LOB0000099610C00026$$.SYS_LOB_P72695 (LOB PARTITION)	ODDO_TABLE
341299	1.02	enq: TX - row lock contention	1.02	ODDO.ODDO_WORTH_REMAINS.TB_38 (TABLE PARTITION)	ODDO_TABLE

- This information and even more can be found in AWR report too

Top Latches

Latch	% Latch	Blocking Sid(Inst)	% Activity	Max Sampled Wait secs
latch: enqueue hash chains	28.43	Held Shared	18.55	0.12

# Waits Sampled	# Sampled Wts < 10ms	# Sampled Wts 10ms - 100ms	# Sampled Wts 100ms - 1s	# Sampled Wts > 1s
50,189	2,979	47,098	112	0

How to read ASH report? - Timeline

- Last table of the report is "Activity Over Time", a kind of timeline.
- We can see that the minutes 12:09 and 12:10 were the most loaded one (24 samples for 7 seconds at 12:09 time slot and 147 samples for 1 minutes at 12:10 time slot)

Activity Over Time

- Analysis period is divided into smaller time slots
- Top 3 events are reported in each of those slots
- 'Slot Count' shows the number of ASH samples in that slot
- 'Event Count' shows the number of ASH samples waiting for that event in that slot
- '% Event' is 'Event Count' over all ASH samples in the analysis period

Slot Time (Duration)	Slot Count	Event	Event Count	% Event
12:09:53 (7 secs)	24	CPU + Wait for CPU	24	3.73
12:10:00 (1.0 min)	147	CPU + Wait for CPU	147	22.86
12:11:00 (1.0 min)	61	CPU + Wait for CPU	61	9.49
12:12:00 (1.0 min)	59	CPU + Wait for CPU	59	9.18
12:13:00 (1.0 min)	61	CPU + Wait for CPU	61	9.49
12:14:00 (1.0 min)	59	CPU + Wait for CPU	59	9.18
12:15:00 (1.0 min)	61	CPU + Wait for CPU	61	9.49
12:16:00 (1.0 min)	59	CPU + Wait for CPU	59	9.18
12:17:00 (1.0 min)	59	CPU + Wait for CPU	59	9.18
12:18:00 (53 secs)	53	CPU + Wait for CPU	53	8.24

Performance analysis using ASH reports

- Choose the appropriate time bounds for ASH report. Include the time when problem started.
- Check the report target (time bounds and filters) in the caption. Compare to the problem timing reported by user or monitoring tools.
- Check for Top waitevents. Some of them could be the consequences only.
- Check for Top parameters P1/P2/P3. Do we have some "hot" element?
- Check for Top SQL. Unusual SQL's in the TOP could be the cause of the problem OR the consequences of the problem.
- Have a look at the AWR report too. It contains elapsed time and execution count for Top SQLs'. It also contains instance metrics like "Logons per sec", "Redo per sec" which are essential for db performance.

gv$active_session_history - How many columns?

- Oracle 11.2 - 96 columns
- Oracle 12.2 - 113 columns

v$active_session_history columns review - 1

Name	Null?	Type
SAMPLE_ID		NUMBER
SAMPLE_TIME		TIMESTAMP(3)
IS_AWR_SAMPLE		VARCHAR2(1)
SESSION_ID		NUMBER
SESSION_SERIAL#		NUMBER
SESSION_TYPE		VARCHAR2(10)
FLAGS		NUMBER
USER_ID		NUMBER
SQL_ID		VARCHAR2(13)
IS_SQLID_CURRENT		VARCHAR2(1)
SQL_CHILD_NUMBER		NUMBER
SQL_OPCODE		NUMBER
SQL_OPNAME		VARCHAR2(64)
FORCE_MATCHING_SIGNATURE		NUMBER
TOP_LEVEL_SQL_ID		VARCHAR2(13)
TOP_LEVEL_SQL_OPCODE		NUMBER
SQL_PLAN_HASH_VALUE		NUMBER
SQL_PLAN_LINE_ID		NUMBER
SQL_PLAN_OPERATION		VARCHAR2(30)
SQL_PLAN_OPTIONS		VARCHAR2(30)
SQL_EXEC_ID		NUMBER
SQL_EXEC_START		DATE

ASH sample metadata

Session info

SQL statement info

SQL execution *plan* info

Individual SQL *execution* info

v$active_session_history columns review - 2

```
...
PLSQL_ENTRY_OBJECT_ID            NUMBER
PLSQL_ENTRY_SUBPROGRAM_ID        NUMBER
PLSQL_OBJECT_ID                  NUMBER
PLSQL_SUBPROGRAM_ID              NUMBER
QC_INSTANCE_ID                   NUMBER
QC_SESSION_ID                    NUMBER
QC_SESSION_SERIAL#               NUMBER
PX_FLAGS                         NUMBER
EVENT                            VARCHAR2(64)
EVENT_ID                         NUMBER
EVENT#                           NUMBER
SEQ#                             NUMBER
P1TEXT                           VARCHAR2(64)
P1                               NUMBER
P2TEXT                           VARCHAR2(64)
P2                               NUMBER
P3TEXT                           VARCHAR2(64)
P3                               NUMBER
WAIT_CLASS                       VARCHAR2(64)
WAIT_CLASS_ID                    NUMBER
WAIT_TIME                        NUMBER
SESSION_STATE                    VARCHAR2(7)
TIME_WAITED                      NUMBER
```

PL/SQL object info, join to dba_procedures / @procid.sql

Parallel execution info

Wait event info

Wait event parameters (extra info)

Remember, you should *not* sum any wait columns, use COUNT(*) to estimate DB Time

v$active_session_history columns review - 3

BLOCKING_SESSION_STATUS	VARCHAR2(11)
BLOCKING_SESSION	NUMBER
BLOCKING_SESSION_SERIAL#	NUMBER
BLOCKING_INST_ID	NUMBER
BLOCKING_HANGCHAIN_INFO	VARCHAR2(1)
CURRENT_OBJ#	NUMBER
CURRENT_FILE#	NUMBER
CURRENT_BLOCK#	NUMBER
CURRENT_ROW#	NUMBER
TOP_LEVEL_CALL#	NUMBER
TOP_LEVEL_CALL_NAME	VARCHAR2(64)
CONSUMER_GROUP_ID	NUMBER
XID	RAW(8)
REMOTE_INSTANCE#	NUMBER
TIME_MODEL	NUMBER
IN_CONNECTION_MGMT	VARCHAR2(1)
IN_PARSE	VARCHAR2(1)
IN_HARD_PARSE	VARCHAR2(1)
IN_SQL_EXECUTION	VARCHAR2(1)
IN_PLSQL_EXECUTION	VARCHAR2(1)
IN_PLSQL_RPC	VARCHAR2(1)
IN_PLSQL_COMPILATION	VARCHAR2(1)
IN_JAVA_EXECUTION	VARCHAR2(1)
IN_BIND, IN_CURSOR_CLOSE, IN_SEQUENCE_LOAD ...	

Blocking session info

DB object involved in a wait (not populated for all waits, not always cleaned up properly)

Database call (OPI call) info

Current transaction info

Time model phase info.

These Y / N flags tell in which phase (SQL parse, SQL execute, login, PL/SQL, login) the session happened to be when sampled

v$active_session_history columns review - 4

CAPTURE_OVERHEAD	VARCHAR2(1)
REPLAY_OVERHEAD	VARCHAR2(1)
IS_CAPTURED	VARCHAR2(1)
IS_REPLAYED	VARCHAR2(1)
DBREPLAY_FILE_ID	NUMBER
DBREPLAY_CALL_COUNTER	NUMBER
SERVICE_HASH	NUMBER
PROGRAM	VARCHAR2(48)
MODULE	VARCHAR2(64)
ACTION	VARCHAR2(64)
CLIENT_ID	VARCHAR2(64)
MACHINE	VARCHAR2(64)
PORT	NUMBER
ECID	VARCHAR2(64)
TM_DELTA_TIME	NUMBER
TM_DELTA_CPU_TIME	NUMBER
TM_DELTA_DB_TIME	NUMBER
DELTA_TIME	NUMBER
DELTA_READ_IO_REQUESTS	NUMBER
DELTA_WRITE_IO_REQUESTS	NUMBER
DELTA_READ_IO_BYTES	NUMBER
DELTA_WRITE_IO_BYTES	NUMBER
DELTA_INTERCONNECT_IO_BYTES	NUMBER
PGA_ALLOCATED	NUMBER
TEMP_SPACE_ALLOCATED	NUMBER

DB Replay & workload capture

Client application info

Execution context identifier (end-to-end request ID)

More precise measurement of DB/CPU time between samples

I/O *counters*. These can be **summed** over multiple samples

Session memory usage when sampled (use MAX or AVG*)

TM_DELTA_TIME / DELTA_TIME are like a duration between previous and current ASH sample of the same session. If session was idle for, say, 10 seconds, these columns contain something about 10 mlns (microsecs).

Cue Fields of ASH

SAMPLE_TIME	When
SESSION_STATE	On waitevent or on CPU (WAITING/ON CPU)
SESSION_ID, SESSION_SERIAL#, SESSION_TYPE	Session
SQL_ID	Current SQL
EVENT	Wait event (NULL for CPU)
TIME_WAITED	Wait duration (0 for CPU)
TM_DELTA_CPU_TIME	CPU time (may be underestimated by 20-40%, I do not know why)

SESSION_STATE = 'ON CPU' is a derived status.
If session is in database call and it is not in a wait, ASH assumes it is on CPU.
Un-instumented waits are also accounted as "ON CPU" (mostly in 10.1).
Session in Run queue: WAITING or "ON CPU"?
Run queue and SMT4 on AIX ==> Unaccounted time

TIME_WAITED vs WAIT_TIME vs TM_DELTA_DB_TIME

All times/durations in ASH are in microseconds (1 millionth of sec)

Duration of current waitevent

0 for waits. >0 for CPU. (is the last completed wait and represents a full sample.). I have no idea why this column is needed.

Duration between previous ASH sample for the same session

DB time accumulated by the session during **TM_DELTA_TIME** interval

TM_DELTA_TIME vs DELTA_TIME

Both fields contains clock time (in microseconds) between current and previous ASH sample. But first one is used for measuring CPU time and DB time, while the second one - for WRITE/READ IO requests and MB.

I guess samples which contains no CPU time have only DELTA_TIME filled in.

Waitevents

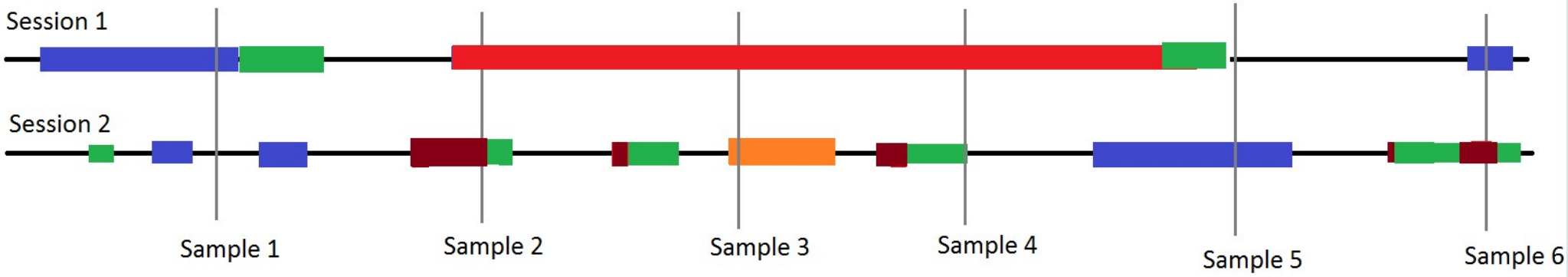
`EVENT`	Waitevent name session is waiting on
`SEQ#`	Same value in several rows for events longer than 1 seconds. Different values for different events (to distinguish series of short events from single long event)
`SESSION_STATE`	WAITING or ON CPU
`P1/P2/P3 , P1Text/P2Text/P3Text`	Event's parameter's values and parameters' names.
`TIME_WAITED`	Duration of current waitevent. Oracle says it's wrong to SUM or AVG it over multiply samples.

TIME_WAITED

TIME_WAITED

Duration of **current** (i.e. incomplete) waitevent.
Oracle says it's wrong to SUM or AVG it over multiply samples.

Value in the TIME_WAITED mostly does not represent the full wait time. If the sampling happens mid-wait, TIME_WAITED will be ½ of the actual length of the wait. If it happens after only 25% of the total wait time, TIME_WAITED will show only ¼ of the total wait.



SQL - related fields

SQL_ID	Current SQL which is being executed (except for triggers and recursive server SQL)
SQL_OPNAME	SELECT / INSERT / UPDATE/ DELETE / PL-SQL
TOP_LEVEL_SQL_ID	SQL_ID of parent SQL
FORCE_MATCHING_SIGNATURE	Useful for similar SQLs with literals
SQL_PLAN_HASH_VALUE	Also can be used for grouping similar SQLs'
SQL_PLAN_LINE_ID	To find the most heavy operation in the execution plan. Also helps to identify the hot object (usually index).
SQL_EXEC_ID, SQL_EXEC_START	To distinguish executions of the same SQL
PLSQL_ENTRY_OBJECT_ID/PLSQL_ENTRY_SUBPROGRAM_ID PLSQL_OBJECT_ID/PLSQL_SUBPROGRAM_ID	Package and subprogram within package which is being executed. ENTRY - for the Top-level (if package function was called from another package).

SQL - related fields

`TOP_LEVEL_SQL_ID` — Current SQL which is being executed (except for triggers and recursive server SQL)

В Top'е запрос, выполнился 47 млн раз за 30 минут и на 95% сидит на I/O. Получено 6 млн строк.

Откуда этот запрос (SQL Module у него пустой)?

Executions	Rows Processed	Rows per Exec	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
46,790,948	6,424,230	0.14	65,332.44	5.7	95.4	2q1pc1y8uf0nr		SELECT CASE WHEN :B2 ='CC' THE...
46,712,382	6	0.00	9,964.75	96.8	0	6uhxvn601tfp6		WITH CM_ALL AS(SELECT /*+ lea...
3,097,802	17,617,817	5.69	43,160.71	18.3	85.9	1mw02zv6yszwu	SBRF CC Contact FA Cards View	WITH MAIN_ASSET AS (SELECT /*+...

Ответ: в ASH видим, что у этого быстрого запроса везде одно и то же значение в TOP_LEVEL_SQL_ID.

В том же ASH видим, что этот Top-Level SQL выполняется уже 10 часов и ни разу не выполнился. В SQL Monitoring видим, что в нем есть функция, и она попала на шаг плана, выполняющийся несколько миллионов раз.

Вывод: немасштабируемый прикладной код.

CURRENT_OBJ# / FILE# / BLOCK# / ROW#

CURRENT_OBJ#	Object ID of the object that the session is referencing. V$SESSION.ROW_WAIT_OBJ#.
CURRENT_FILE#	File number of the file containing the block that the session is referencing. V$SESSION.ROW_WAIT_FILE#.
CURRENT_BLOCK#	File number of the file containing the block that the session is referencing. V$SESSION.ROW_WAIT_FILE#.
CURRENT_ROW#	Row identifier that the session is referencing. V$SESSION.ROW_WAIT_ROW#.

CURRENT_OBJ# is only correct for sessions waiting on application (row lock, table lock), cluster (on data blocks), concurrency (buffer busy), and user I/O on data blocks. Maps to DBA_OBJECTS.DATA_OBJECT_ID.

For CPU and other wait samples this fields sometimes contains wrong (stale) information (this columns are not cleared).

For some Cluster and User I/O correct information for file# and block# could be found in P1/P2.

Sometimes (for ITL-waits) CURRENT_OBJ# is **0** or **-1** while correct for tx - row lock contention.

As a workaround , check the current SQL for possible candidates for contention.

So it is better not to rely on current_obj# in ash..

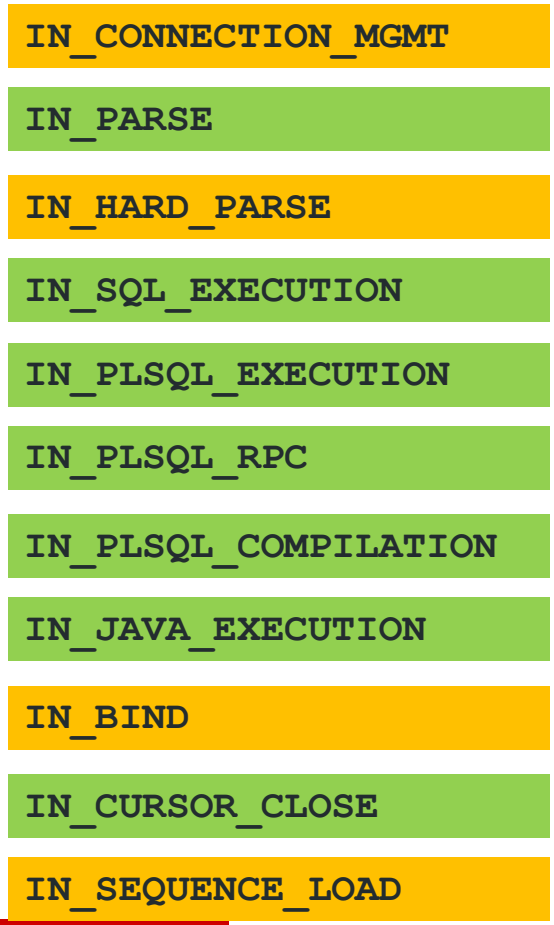
<-- I found these recommendation in one of the bugs

CURRENT_OBJ# - when it is correct?

Customer case For **enq: tx - index contention** (Concurrency waitclass) current_obj# can be NULL or -1, but it's correctly populated in the row for the blocking session. It requires us to join v$ASH to itself and to dba_objects to find the name of hot index.

```
select     decode(a2.current_obj#,-1,a1.current_obj#,null,a1.current_obj#,a2.current_obj#) current_obj#
         , a1.sample_time
         , count(*) cnt_samples
         , avg(decode(a1.time_waited,0,null,a1.time_waited)) avg_not_zero_time_waited
       from v$active_session_history a1
       left join v$active_session_history a2 on a1.blocking_session=a2.session_id and
a1.blocking_session_serial#=a2.session_serial# and decode(a1.time_waited,0,a1.sample_id-1, a1.sample_id)
=a2.sample_id
      where a1.sample_time > sysdate-1/24/60*5 ----!!!! last 5 minutes
       and a1.event = 'enq: TX - index contention'
      group by decode(a2.current_obj#,-1,a1.current_obj#,null,a1.current_obj#,a2.current_obj#),a1.sample_time
```

Bit Vector



Some operations are designed to be too short to measure their duration in the time model.

They are being kept in the session bit vector.

Oracle just set a bit before entering the operation's code and reset it after the code is finished.

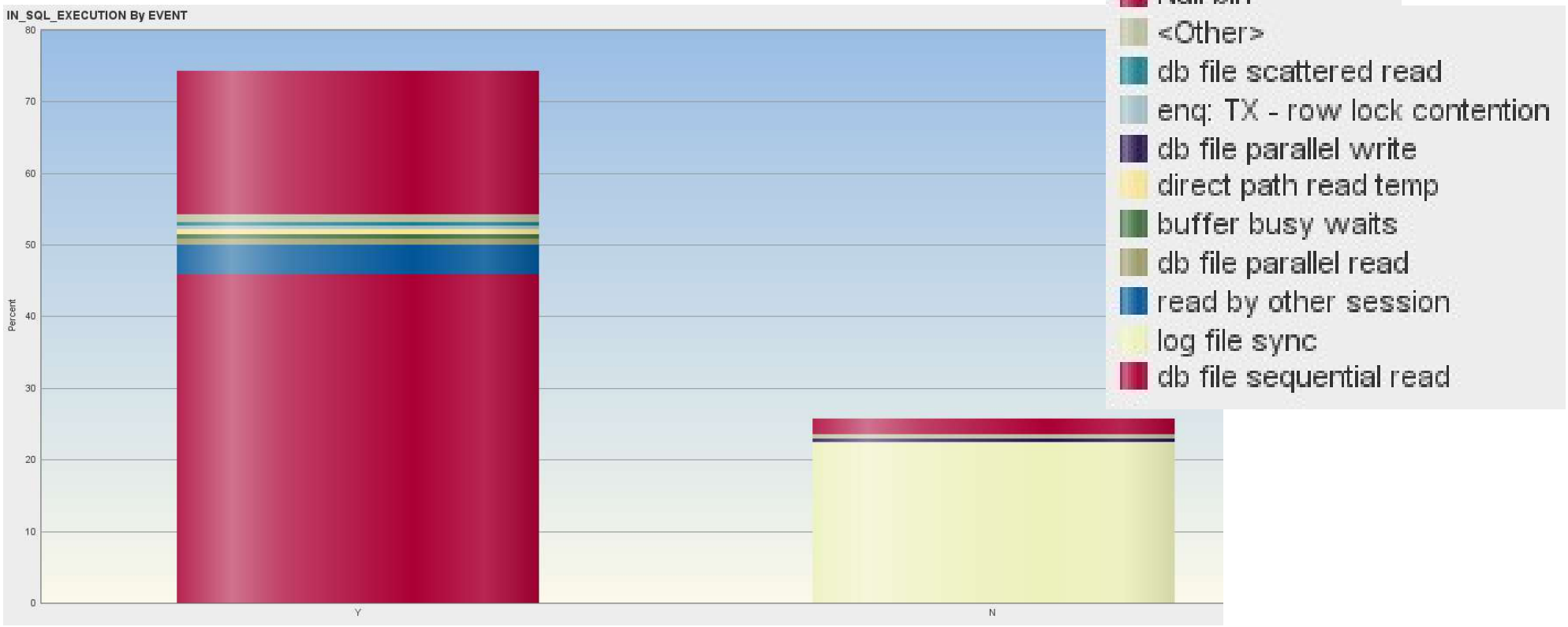
ASH samples this bit vector.

Only really many operations in this state could become a problem.

Use the regular COUNT(*) to estimate how much time was spent on this untimed operations.

IN_SQL_EXECUTION - by event (chalna_dbdpc)

IN_SQL_EXECUTION



When my SQL started?

AWR

```
select  st.snap_id,
        cast(sn.begin_interval_time as date) begin_time,
        cast (sn.end_interval_time as date) end_time,
        st.executions_delta exec_delta,
        st.executions_total exec_total,
        st.end_of_fetch_count_delta eof_delta,
        trunc(st.elapsed_time_delta/1e6) elapsed_delta_sec,
        trunc(st.elapsed_time_total/1e6) elapsed_total_sec
from    dba_hist_sqlstat st, dba_hist_snapshot sn
where   sql_id = 'dskp9f0r23dj3'
          and st.snap_id=sn.snap_id
order by        st.snap_id;
```

SNAP_ID	BEGIN_TIME	END_TIME	EXEC_DELTA	EXEC_TOTAL	EOF_DELTA	ELAPSED_DELTA_SEC	ELAPSED_TOTAL_SEC
1361	28.11.2017 12:00:29	28.11.2017 12:10:29	7	7	3	177	177
1362	28.11.2017 12:10:29	28.11.2017 12:20:30	0	7	3	533	712

When my SQL started?

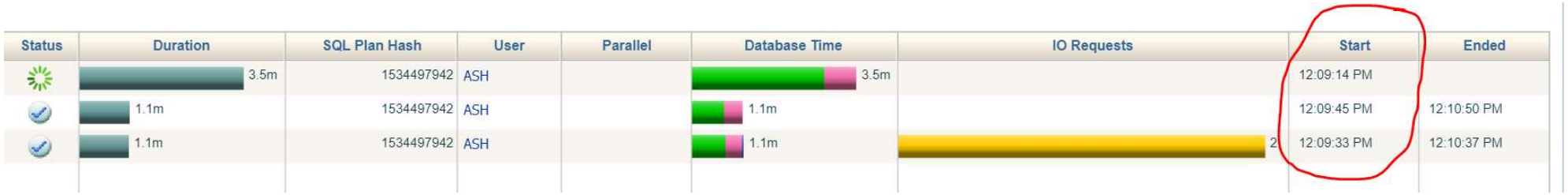
ASH

```
select  a.sql_exec_id,   a.sql_exec_start,
   a.module,    a.session_id,
   min(a.sample_time) "Started",
   max(a.sample_time) "Ended",
   Round((cast(max(a.sample_time) as date) - cast(min(a.sample_time) as
date))*24*60*60) "Elapsed time, sec"
from    v$active_session_history a
where  sql_id = 'dskp9f0r23dj3'
group by   a.sql_exec_id,   a.sql_exec_start,   a.module,   a.session_id
order by min(a.sample_time);
```

SQL_EXEC_ID	SQL_EXEC_START	MODULE	SESSION_ID	Started	Ended	Elapsed time, sec
16777219	28.11.2017 12:09:14	SQL*Plus	156	28.11.2017 12:09:15	28.11.2017 12:18:53	578
16777220	28.11.2017 12:09:22	SQL*Plus	36	28.11.2017 12:09:23	28.11.2017 12:09:25	2
16777221	28.11.2017 12:09:33	SQL*Plus	36	28.11.2017 12:09:34	28.11.2017 12:10:38	64
16777222	28.11.2017 12:09:45	SQL*Plus	149	28.11.2017 12:09:46	28.11.2017 12:10:50	64

When my SQL started?

SQL Monitoring



Status	Duration	SQL Plan Hash	User	Parallel	Database Time	IO Requests	Start	Ended
	3.5m	1534497942	ASH		3.5m		12:09:14 PM	
	1.1m	1534497942	ASH		1.1m		12:09:45 PM	12:10:50 PM
	1.1m	1534497942	ASH		1.1m	2	12:09:33 PM	12:10:37 PM

SQL Monitoring contains very valuable metrics like CPU consumption and IO requests tracked for SQL executions.

Some of these metrics are missing even in ASH.

What my SQL spent the execution time for?

```
-- CPU and waits for every SQL execution
select
    a.sql_exec_start,
    NVL(a.event,'ON CPU') "Wait event",
    COUNT(*) "ASH samples cnt",
    Round(SUM(TM_DELTA_TIME)/1e6) TM_DELTA_TIME,
    Round(SUM(DELTA_TIME)/1e6) DELTA_TIME,
    Round((cast(max(a.sample_time) as date) -
           cast(min(a.sample_time) as date))*24*60*60) "Elapsed time, sec"
from
    v$active_session_history a
where
    sql_id = 'dskp9f0r23dj3'
group by    a.sql_exec_start, NVL(a.event,'ON CPU')
order by min(a.sample_time);
```

	SQL_EXEC_START	Wait event	ASH samples cnt	TM_DELTA_TIME	DELTA_TIME	Elapsed time, sec
1	12:09:14	ON CPU	578	578	580	578
2	12:09:22	ON CPU	3	3	3	2
3	12:09:33	ON CPU	65	73	73	64
4	12:09:45	ON CPU	65	65	65	64

Was parallel execution used or not?

```
select
    a.sql_exec_id, NVL(a.QC_SESSION_ID, a.session_id) session_id,
    ROUND(PX_FLAGS/2097152) "Degree",
    a.sql_exec_start,
    NVL(a.event,'ON CPU') "Wait event",
    COUNT(*) "ASH samples cnt",
    Round(SUM(TM_DELTA_CPU_TIME)/1e6) CPU_TIME,
    Round(SUM(TM_DELTA_DB_TIME)/1e6) DB_TIME
from
    v$active_session_history a
where
    sql_id in ('7wwa8agka0xg8','3xcgx55t55juc')
group by   NVL(a.QC_SESSION_ID, a.session_id), a.PX_FLAGS, a.sql_exec_id,
           a.sql_exec_start, NVL(a.event,'ON CPU')
order by min(a.sample_time);
```

QC_INSTANCE_ID

QC_SESSION_ID

QC_SESSION_SERIAL#

QC_FLAGS

QC stands for "Query coordinator"

Program column can also be an indicator of PX execution: **oracle@anna.testdomain.ru (P001)**

	SQL_EXEC_ID	SESSION_ID	Degree	SQL_EXEC_START	Wait event	ASH samples cnt	CPU_TIME	DB_TIME	Elapsed time, sec
1	16777216	35	4	14:23:38	ON CPU	4247	1272	4255	1626
2	16777216	35	4	14:23:38	direct path read	2233	674	2253	1625
3	16777216	36	(null)	14:23:53	ON CPU	1050	324	1062	1611
4	16777216	36	(null)	14:23:53	direct path read	555	167	550	1605
5	16777217	149	(null)	14:24:07	ON CPU	1035	296	1012	1598
6	16777217	149	(null)	14:24:07	direct path read	557	172	587	1594

Blocking session in ASH - fields

`BLOCKING_SESSION/BLOCKING_SESSION_SERIAL#`	SID/SERIAL# of blocking session
`BLOCKING_INST_ID`	Instance num of blocking sess (new 11g)

Documentation is misleading (both for 11.2 and 19c):

`BLOCKING_SESSION:` Populated only if the blocker is on the same instance and the session was waiting for enqueues or a "buffer busy" wait. Maps to V$SESSION.BLOCKING_SESSION.

If you are trying to use this field to build a blocking tree, you need to join ASH to itself on **A.sample_id=B.sample_id** and **A.BLOCKING_SESSION=B.SESSION_ID** and **A.session_serial# = B.blocking_session_serial#**.

You need several joins to itself if you have long chain of locks.

NB1: V$-views DO NOT support read consistency, so we recommend to perform CTAS and join the table.

NB2: the sample time is only consistent on a by-instance basis in a RAC. NOT across the full RAC. This is one more reason why blocking session can be wrong.

Graham Wood: Hang information in ASH is reliable for 3-sec lock in single instance, 10 sec for RAC.

Blocking session in ASH - scripts

Oracle's script (PL/SQL over dba_hist_active_sess_history): **find_ASH_hang_chains.sql**

Tanel Poder's script (pure SQL, unlimited nesting) : **ash_wait_chains_Poder.sql, dash_wait_chains_Poder.sql**

My script (simple SQL, single level):

```
 with waiters as
(
    select blocking_Session,blocking_session_Serial#,sample_id,count(*) cnt
    from gv$active_session_history
    where blocking_session is not null
    and event in ('library cache lock','cursor: pin S wait on X')
    group by inst_id,blocking_Session,blocking_session_Serial#,sample_id
)
select /*+ NO_MERGE(waiters) */ waiters.cnt "blocked sessions cnt",  ash.*
from gv$active_session_history ash join waiters on ash.session_id = waiters.blocking_Session
    and ash.session_serial#=waiters.blocking_session_Serial# and ash.sample_id=waiters.sample_id
order by waiters.cnt desc,sample_time;
```

The main concern: if blocking session is IDLE, you never find it in the same sample of ASH. You need to go back in time to find the moment when it was active. And it may happen that operation which put the lock is not in the ASH at all (missed to be sampled).

Blocking session in ASH - Wrong?

	Waiter	Blocker
SAMPLE_TIME	11.16.15.841 AM	11.16.15.841 AM
SESSION_ID	475	20683
SESSION_SERIAL#	10983	5821
SQL_OPNAME	INSERT	SELECT
SQL_PLAN_OPERATION	INSERT STATEMENT	TABLE ACCESS
SQL_PLAN_OPTIONS		BY LOCAL INDEX ROWID
SQL_EXEC_START		22.11.2017 11:16:11
EVENT	library cache: mutex X	db file sequential read
WAIT_CLASS	Concurrency	User I/O
SESSION_STATE	WAITING	WAITING
TIME_WAITED	10193	6361
BLOCKING_SESSION_STATUS	VALID	NO HOLDER
BLOCKING_SESSION	20683	
BLOCKING_SESSION_SERIAL#	5821	

We were trying for find the blocking session for one INSERT'ing and waiting on "library cache: mutex X".
We suspect the concurrency either on INSERT's cursor or on BEFORE INSERT trigger.
The root cause of this concurrency is VERY frequent execution of INSERT and implicit execution of the trigger.

According to ASH, blocking session is performing SELECT query for 4 seconds and waiting on db file sequential read (reading index).

How SELECT which is reading from disk can cause concurrency on library cache? My only suggestion is ASH "BLOCKING_SESSION" column is wrong.

Performance analysis using ASH vs DB_TIME

- ASH Math: COUNT(*) = DB Time (seconds)
  - GROUP BY dimensions of interest
  - Multiply by 10 for on-disk queries
- Avoid to use MIN, AVG, MAX
  - Sampling is biased to longer events
- AWR/ASH difference regarding BACKGROUND processes
  - DB time in AWR does not include BG process activitity, while ASH does

ASH Top - scripts - Tanel Poder - example 1

Tanel Poder script: **ashtop_Poder.sql**

Parameters: **GROUP BY, FILTER, FROM, TO**

Example1: @ashtop_Poder.sql **event2** session_type='FOREGROUND' sysdate-10/1440 sysdate

Samples	AAS	%This	EVENT2	FIRST_SE	LAST_SEE	Distinct Execs Seen
709	2.4	65%	ON CPU	09:43:41	09:47:33	4
309	1.0	28%	direct path read	09:43:40	09:47:02	3
64	0.2	6%	direct path write temp	09:44:00	09:47:01	1
7	0.0	1%	direct path read temp	09:47:10	09:47:26	1

AAS - Average active sessions for the whole timeframe between FROM and TO

Event2 - a calculated column, extended EVENT column

ASH Top - scripts - Tanel Poder's - example 2

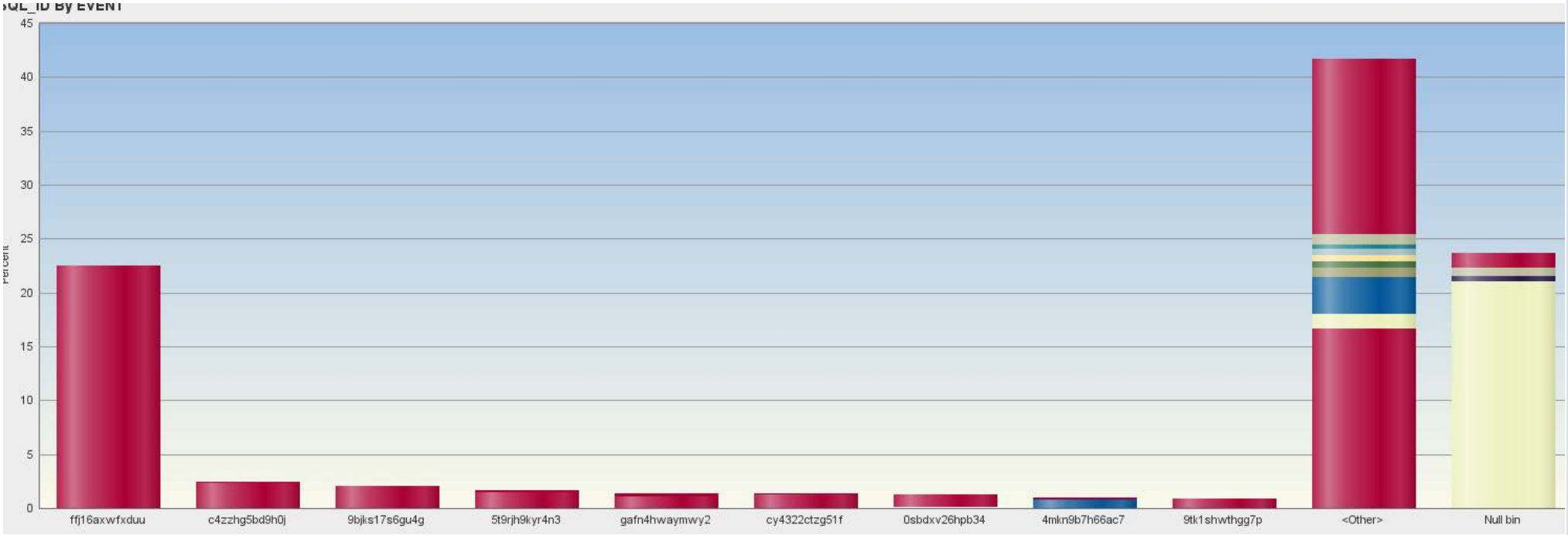
Example2: @ashtop_Poder.sql **program2** "wait_class='User I/O'" "to_timestamp(to_char(sysdate,'ddmmyyyy ')||'09:43:00','ddmmyyyy hh24:mi:ss')" "to_timestamp(to_char(sysdate,'ddmmyyyy ')||'09:48:00','ddmmyyyy hh24:mi:ss')"

Samples	AAS	%This	PROGRAM2	First Seen	Last Seen	Distinct Execs Seen
311	1.0	82%	(Pnnn)	09:43:40	09:47:26	1
69	0.2	18%	(sqlplus)	09:44:00	09:45:35	2

Program2 - a "cleared" PROGRAM column

```
CASE WHEN a.session_type = 'BACKGROUND' OR REGEXP_LIKE(a.program, '.*\([PJ]\d+\)') THEN
      REGEXP_REPLACE(SUBSTR(a.program,INSTR(a.program,'(')), '\d', 'n') -- меняем цифры на "n" в параллельных процессах и джобах
  ELSE
   '('||REGEXP_REPLACE(REGEXP_REPLACE(a.program, '(.*)@(.*)(\(.*\))', '\1'), '\d', 'n')||')' -- убираем все, что после @
  END || ' ' program2
```

ASH analytics - Data mining in SQL Developer



ASH Top - scripts - Mikhail Bazgutdinov

ASH_Top2_events.sql <-- From memory
ASH_Top2_events_AWR.sql <-- From AWR, requires to specify filter by snap_id, dbid, instance_number, sample_time (WRH$_ACTIVE_SESSION_HISTORY partitioned on DBID and SNAP_ID)

Time	Total se...	Top1 event	Top1 count sess	Top1 event avg wait,msec	Top2 event	Top2 count sess	Top2 event av...	Written by ...	LGWR wait ...
11:43:57	6	direct path read	4	(null)	On CPU	2	(null)	(null)	(null)
11:43:56	6	On CPU	3	(null)	direct path read	3	10,4	(null)	(null)
11:43:55	6	On CPU	3	(null)	direct path read	3	8,4	(null)	(null)
11:43:54	6	On CPU	3	(null)	direct path read	2	10,2	(null)	(null)
11:43:53	6	On CPU	4	(null)	direct path read	2	10,5	(null)	(null)
11:43:52	6	On CPU	3	(null)	direct path read	3	2	(null)	(null)
11:43:51	6	On CPU	5	(null)	direct path read	1	1,7	(null)	(null)
11:43:50	7	On CPU	5	(null)	direct path read	2	10,8	(null)	(null)

"Bonus": it also shows LGWR activity (amount redo written, waitevent, wait_time).
You can easily change it to DBWR.

ASH Top - scripts - Randolf Geist's **XPLAN_ASH**

Very sophisticated script - Latest version on Github

```
Real-Time SQL Monitoring Execution Summary
------------------------------------------

| STATUS | USERNAME | PX IS CROSS INST | PX MIN DOP | PX MAX DOP | PX INSTANCES | PX SERVERS REQUESTED | PX SERVERS ALLOCATED | DURATION AND DATABASE TIME GRAPH | DURATION | DATABASE TIME | CPU TIME |
|---|---|---|---|---|---|---|---|---|---|---|---|
| DONE (ALL ROWS) | CBO_TEST | Y | 3 | 3 | 2 | 9 | 9 | ####### @@@@@@@@@@**######## | +0 00:00:57 | +0 00:03:02 | +0 0 |


SQL statement execution ASH Summary
-----------------------------------

| INSTANCE_COUNT | SESSION_ID | INSTANCE_ID | USER_ID | TOP_LEVEL_SQL_ID | FIRST_SAMPLE | LAST_SAMPLE | DURATION SECS TOTAL | DURATION TOTAL | DURATION SECS ACTIVE | DURATION ACTIVE | STA |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 157 | 1 | 90 | 4qguzswbbpva3 | 2014-12-31-20:02:01 | 2014-12-31-20:02:56 | 58 | +0 00:00:58 | 56 | +0 00:00:56 | INA |
```

Blog: https://oracle-randolf.blogspot.com/2015/12/new-version-of-xplanash-utility.html
Youtube: https://www.youtube.com/watch?v=2UYZVU68g4c

Developers not Using Bind Variables

Which ASH column we can use to discover this kind of problem?

SELECT COUNT (DISTINCT SQL_ID), SQL_PLAN_HASH_VALUE …

SELECT COUNT (DISTINCT SQL_ID), FORCE_MATCHING_SIGNATURE …

Difference between SQL_PLAN_HASH_VALUE and FORCE_MATCHING_SIGNATURE is different SQL's can use the same plan (e.g. accessing single table by the same index) while FORCE_MATCHING_SIGNATURE is strongly for SQL differs by literal values only.

Which step is the most time consuming in complex SQL?

If your SQL query is complex enough, you may want to know which step of the plan is most expensive

```
select count(*), SQL_PLAN_LINE_ID
From v$active_session_history
where sql_id=''
And IS_SQLID_CURRENT='Y'
Group by SQL_PLAN_LINE_ID
order by count(*) desc;
```

Why do we need not current SQL_ID in the ASH? Who knows?

Василий

- Как сосчитать количество выполнений SQL-запроса по изменению значений sql_exec_id
- select max(sql_exec_id) - min(sql_exec_id) cnt, min(sql_exec_id), max(sql_exec_id), sample_time
  from mbazgutdinov.CHALNA_ASH110920 t where event = 'Disk file operations I/O' and sql_id = 'dxj8fuhckzhj7'
  group by sample_time
-
- SQL_EXEC_ID - When the SQL_ID is specified, the SQL_EXEC_ID indicates the individual execution of interest. When NULL (the default) the most recent execution of the statement targeted by the SQL_ID is assumed.

What is missing in ASH

- Reliable information about blocking session. We need hanganalyze.
- Current call stack. We need hanganalyze or pstack.
- Bind variables' values.

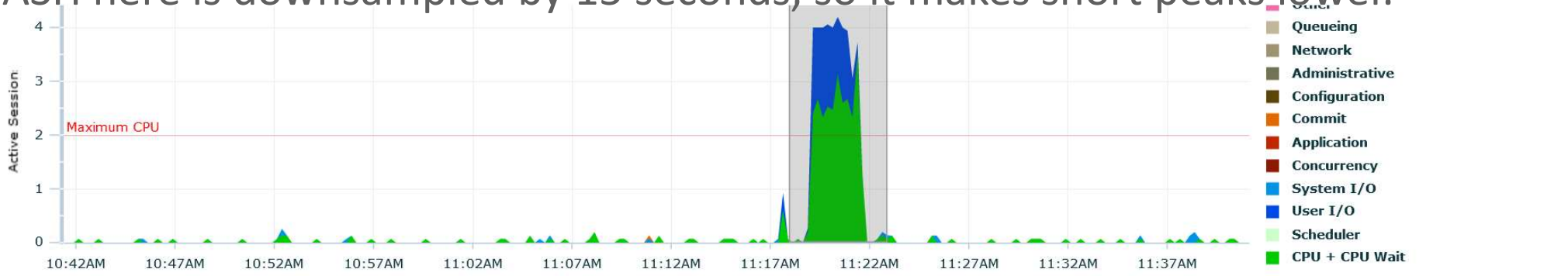
- All these flaws are nothing comparing to the main advantage of ASH: ASH is always ON.

ASH analytics in Cloud Control

ORACLE®

Performance -> Top Activity

Performance -> Top Activity - Default view. This screen is based on ASH data. ASH here is downsampled by 15 seconds, so it makes short peaks lower.



Detail for Selected 5 Minute Interval

Start Time Dec 4, 2017 11:17:56 AM

Run ASH Report

Top SQL

Actions Schedule SQL Tuning Advisor Go

Select All | Select None

Select	Activity (%)	SQL ID	SQL Type
☐	99.50	1qat0gc1g20ad	SELECT
☐	.17	6ajkhukk78nsr	PL/SQL EXECUTE
☐	.17	6403c5znjqh5b	INSERT
☐	.17	691p2a1hswk0m	INSERT

Top Sessions

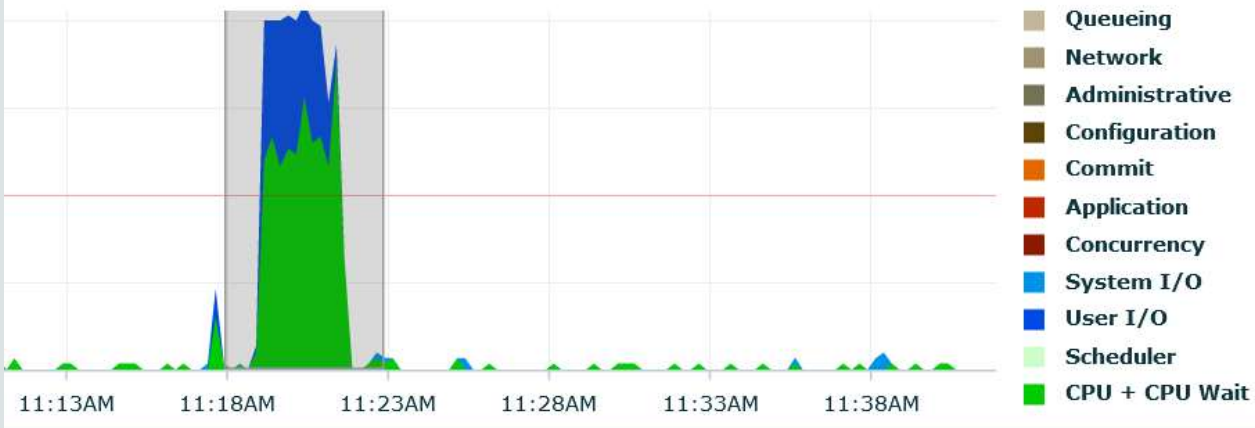
View Top Sessions

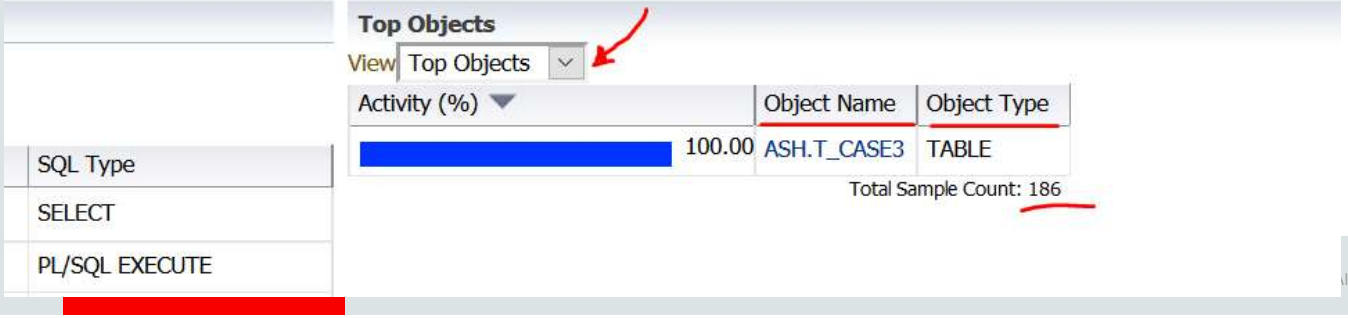
Activity (%)	Session ID	QC Session ID	User Name	Program
24.80	24	42	ASH	oracle@anna.testdomain.ru (P001)
24.80	30	42	ASH	oracle@anna.testdomain.ru (P003)
24.15	152	42	ASH	oracle@anna.testdomain.ru (P000)
23.18	156	42	ASH	oracle@anna.testdomain.ru (P002)
.81	5		SYS	oracle@anna.testdomain.ru (DBW0)

Color codes are consistent across EM releases and different EM screens.

Performance -> Top Activity: different Top dimensions

Performance -> Top Activity - Choose "Top Objects" to display.
More useful views: Top Services/Modules/Files/PL-SQL





Pro:

- Color scheme reflects Waitclass and it is consistent among screens.
- You can "drill down" into the Waitclass.

Con:

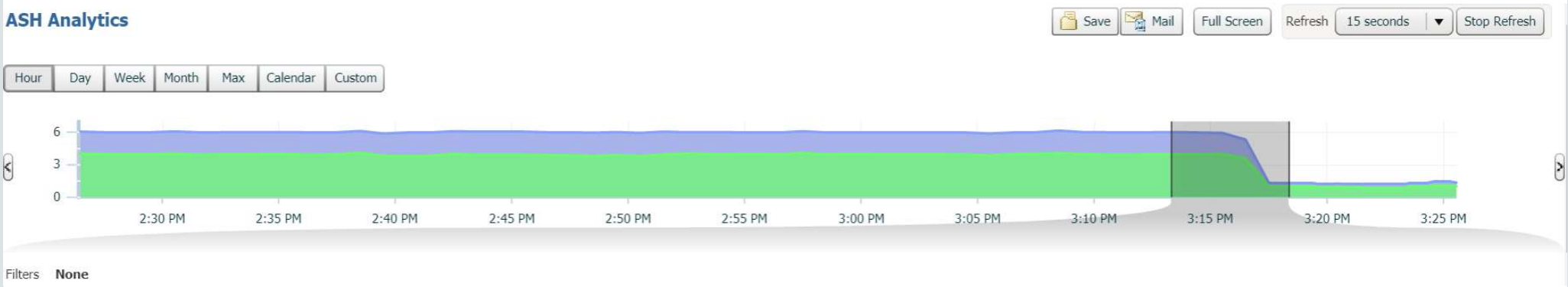
- You cannot chose the timeframe length to analyze.
- You cannot filter data by any attribute except WaitClass.
- You cannot drill down deeper than Waitclass.
- Response is slow if db overloaded.

Performance -> Performance home - CPU Wait

Performance -> Performance home. This screen is based on v$sysmetric_history data (captured once per minute). It contains derived component "CPU Wait"

Timeline to pickup the timeframe for analysis

Performance -> ASH analytics



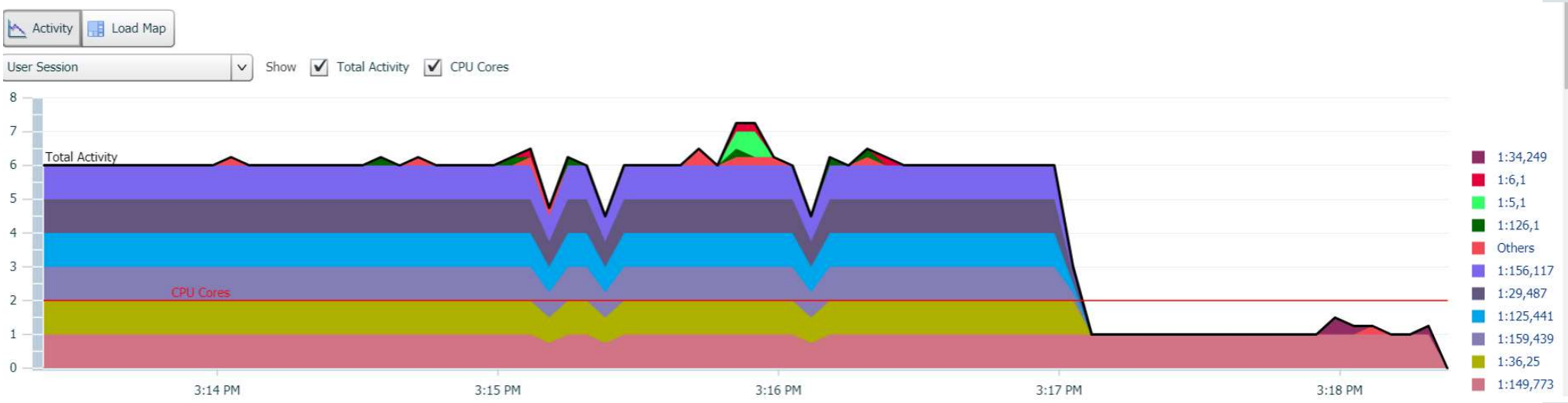
You can choose the timeframe of analyzing data (change the width of "gray rectangle" and amount of data displayed on the timeline.

You can save the current view or even email it

You cannot use it to analyze saved into table ASH contents.

Timeline activity

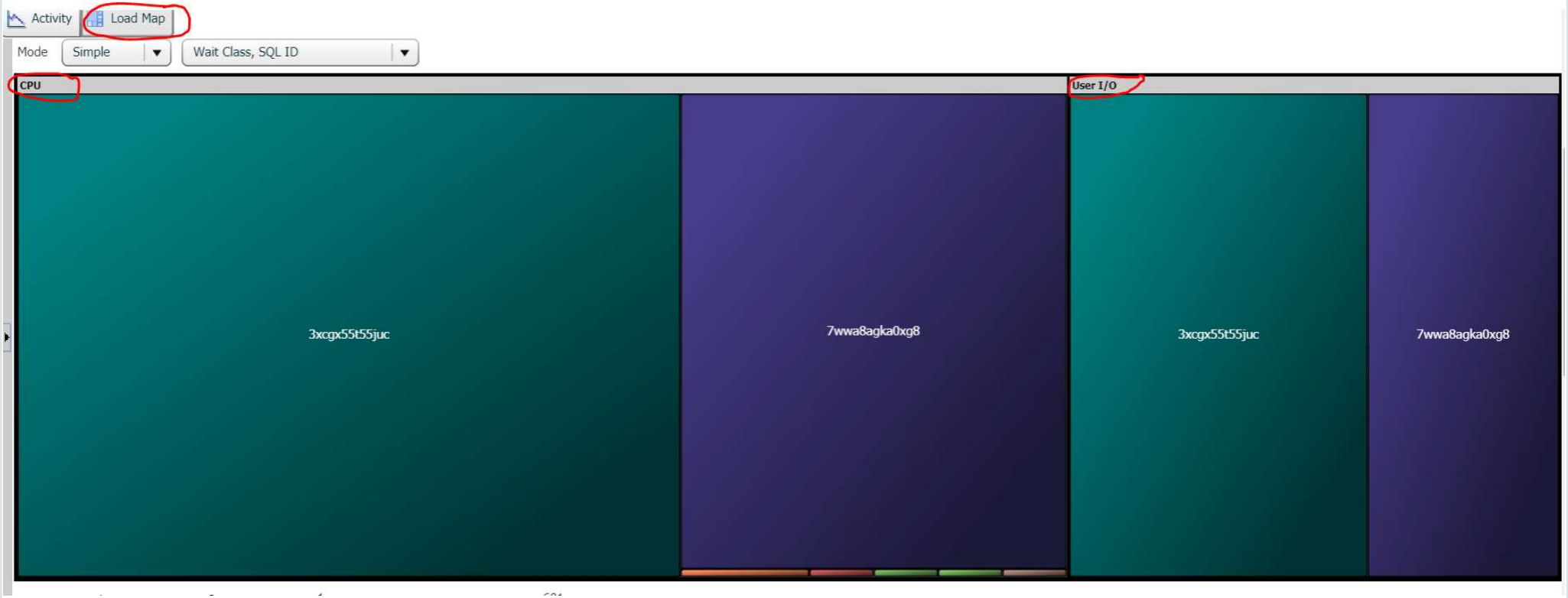
- Active sessions.



NB: each parallel thread is shown as a separate session. No possibility to group by QC.

Load map

- Wait class and SQL_ID



Load map

- Top level SQL -> SQL (library cache: mutex X)

12c enhancements in ASH

ORACLE®

16 new columns in V$ASH in 12.2 vs 11.2

Container database identifications: CON_DBID, CON_ID

Contain new features identifications: Come you aware of?

You name it and I will try to find the corresponding column in ASH.

5 columns for IN Memory: IN_INMEMORY_QUERY
IN_INMEMORY_POPULATE/PREPOPULATE/REPOPULATE/TREPOPULATE

Adaptive plans: SQL_ADAPTIVE_PLAN_RESOLVED, SQL_FULL_PLAN_HASH_VALUE

- DBOP_NAME, DBOP_EXEC_ID Database operation name and exec id. Used for Real-Time **Database** Monitoring by calls to **DBMS_SQL_MONITOR** .BEGIN_OPERATION and END_OPERATION.
- IN_TABLESPACE_ENCRYPTION (Y) Encryption or decryption of a tablespace occurred (Y) or not (N)
- DELTA_READ_MEM_BYTES - Number of read bytes through the buffer cache
  - missed in dba_hist_active_sess_history (enhancement request exists since 2016 to add it, not ready yet).

Undocumented: USECS_PER_ROW, SAMPLE_TIME_UTC

Multitenant support in ASH

CON_ID, CON_DBID columns in the DBA_HIST_ACTIVE_SESS_HISTORY

CDB_HIST_ACTIVE_SESS_HISTORY is a select from DBA_HIST_ACTIVE_SESS_HISTORY forcing the container

Summary - 1

Oracle diagnostics is very comprehensive and explanatory (when used properly).

ASH provides MANY points of view into SQL execution.

Visualizing ASH info makes it easier (CC ASH analytics or any other tool).

SQL Monitoring fills some of the gaps.

ASH data is only correct statistically, i.e. it is not 100% correct, and we should not make legal conclusion based on the single line from ASH. We should always do COUNT(*) and if COUNT(*) is few, double check this information from more reliable source (e.g. tracing, errostacks, hanganalyze, SQL Monitoring).

Never use **`WAIT_TIME`**

Summary - 2

Remember that SUM(TIME_WAITED) and AVG(TIME_WAITED) is ALWAYS wrong, how false it is, depends on wait duration, sample moment, wait duration dispersion.

DBA_HIST_ACTIVE_SESS_HISTORY is flushed onto disk every 10 seconds, so even failed AWR snapshots can have ASH data (AWR report is missing but ASH is in place).