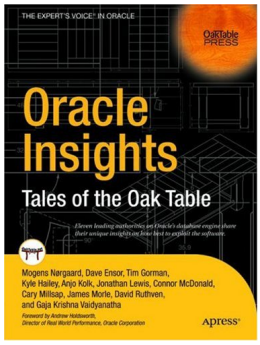
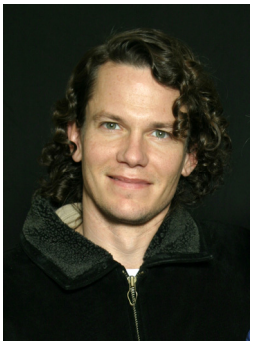


A-Sampling We Will Go

by Kyle Hailey



Oracle wait events took years to become widely known for their power in performance tuning. With this article you'll be able to beat the curve harnessing the power the new Oracle 10*g* concept of Active Session History (ASH). ASH represents a revolution in performance tuning as radical as the introduction of wait events.

Wait events revolutionized tuning way back in Oracle 7 by identifying performance bottlenecks in the database and explicitly quantifying their impact. Still lacking in the wait events interface was the ability to identify the causes of the problems. The solution has come with ASH. ASH clearly identifies the SQL and Sessions at the root of wait bottlenecks.

ASH ironically has be technically feasible by a savvy DBA since Oracle 7 but it took Oracle 10*g* for anyone to put this cutting edge technology to work, and in no small way, but by actually implanting it in the Oracle kernel and basing Oracle 10*g* performance tools on ASH.

ASH is founded upon a data gathering technique called "Sampling." Sampling is a method of collecting data by taking snapshots every second. The snapshots record the state of every active session in the database. For every active session, Oracle records the SQL being executed as well as the state of the session. A session state can be running on the CPU or waiting on any of several hundred wait events. With this simple amount of information, the session + state + sql , we can quickly identify SQL that are CPU intensive, IO intensive or involve in anyone of a number of the possible resource contention issues that can arise in an Oracle database.

Looking at the situation more concretely, when there is a slow down on the database we look at our trusted Statspack report for the period of the slowdown. The first step in analyzing the statspack report is to look at the Top 5 Timed Events

The top 5 timed wait events will tell us if any wait event has crept up to cause a bottleneck. If we do find a wait event bottleneck we will need to know who or what is causing the problem in order to solve it. For example if there is a CPU bottleneck, we need to know what SQL statement is hogging the CPU. If there is an IO bottleneck we need to know what SQL statement is stuck on IO and needs tuning. If there is a complex situation like a buffer busy waits or latch contention we need to know which sessions were involved, what the wait event arguments were and what SQL they were executing. Statspack fails to give us the necessary detailed information but ASH does.

Real Life Example

Let's look at a real life example on a 9*i* database. This example is probably the hardest wait bottleneck to solve, so if you can follow this one then it's downhill from there. This example is one of the more complex tuning problems and thus the analysis requires a deep understanding of Oracle wait events.

Imagine a situation where users call into the help desk complaining that the application has slowed down. The help desk determines it's not the application and calls me, the DBA, to fix it. To solve the situation I run the statspack reports over the last hour and look at "Top 5 Timed Events"

```
Top 5 Timed Events
~~~~~~~~~~~~~~~~~~                                     % Total
Event                 Waits    Time (s) Call Time
--------------------------- ----------- ---------
buffer busy waits     2,748         250     78.72
CPU time                            32     10.16
free buffer waits     1,588          15      4.63
write complete waits     10           8      2.51
log buffer space        306           5      1.51
```

Sure enough there is a bottleneck and the bottleneck is on "buffer busy waits", but now what do I do to solve it? There are some guesses I can make, and I can even use some data in statspack to help guess, but there is no way to conclusively analyze this wait bottleneck or most wait events with statspack. To solve most wait event bottlenecks we need more detailed information than is available in statspack. What information do we need? We mainly need to know what SQL, Sessions, Objects and wait details involved. How do we get that information? We get than information by sampling v$session_wait, i.e. via ASH. We also need to understand how to solve each wait event and specifically in this case how to solve "buffer busy waits." A complete explanation of buffer busy waits is beyond this presentation but a little information on them will suffice. A buffer busy wait has two central causes. The first cause is contention on reads off disk and is equivalent to an IO wait. The second cause is sessions bottlenecking on writing to a particular block. In order to solve the problem I need to know first the kind of buffer busy wait bottleneck, read or write, and second I need to know the objects and queries involved. With this information I can take steps to alleviate the problem.

How do we find the objects and SQL involved in the buffer busy waits? The information we need can be found in a table v$session_wait when the problem exists but v$session_wait only represents the current state of the system and if our problem is intermittent or passed then the information we need from v$session_wait will be lost. V$session_wait's lack of history is where new Oracle 10*g* concept Active Session History (ASH) comes in. ASH at its

Oracle Performance Training Day with Kyle Hailey

The Training You *Need* at a Price You Can *Afford!*

Is there a *single* metric that summarizes database performance in a nutshell? If you knew what it was, you would watch it like a hawk, hour after hour and day after day, and you would use it to measure the effect of hardware upgrades, software upgrades, and other configuration changes. All your tuning efforts would be directed toward improving the number, and you would want to determine the effect of each user, each program, each application server, each table, and each SQL statement on the number.

If your database suddenly becomes sluggish, can you tap out a few commands and immediately pinpoint the culprit user, program, application server, table, or SQL statement? Can you do the same thing if the database was sluggish earlier in the day when you were not present?

If you want to learn how to quickly solve Oracle performance problems, come take advantage of a full day of training by Kyle Hailey on August 17 at the Carr America Conference Center in Pleasanton for only $250—register by July 15 to receive an early bird price of $195. The price includes continental breakfast, lunch, and a free copy of Oracle Wait Interface: A Practical Guide to Performance Diagnostics & Tuning*, by Richmond Shee et al. It's the training you need, at a price you can afford!*

Kyle Hailey is a regular speaker at NoCOUG conferences and needs very little introduction to NoCOUG members. He is a founding member of the Oak Table and co-author of *Oracle Insights: Tales of the Oak Table*, together with other NoCOUG regulars like Jonathan Lewis, Cary Millsap, and Gaja Krishna Vaidyanatha. He had a long and distinguished career at Oracle and designed the performance tuning module in Oracle Enterprise Manager 10*g*. Take a look at the topics he will cover on August 17.

1. **Proving the database is running smoothly**—The power of proving that the database is running smoothly is often underestimated in performance problem situations. If the database is running smoothly, then the problem is elsewhere, perhaps in the application code. Kyle will explain how to determine if there is a bottleneck in the database, or if the database is running smoothly, in three easy steps.
2. **Performance data collection tools**—How to collect the necessary data to analyze the performance of an Oracle database. An overview of performance data collections tools, including both Statspack and AWR, and practical examples of their use.
3. **Oracle wait events**—Oracle wait events are central to any good performance tuning methodology but are often misunderstood and undocumented. Kyle will explain the causes, show how to analyze, and provide solutions for the top wait events that represent 99% of the wait time in most databases.
4. **Top five wait events**—A step-by-step analysis and solution for the top five wait events. From a Statspack report to the final solution, see each step of analyzing and solving these wait events using real-life examples.
5. **Easy wait events**—One-step solutions when the performance data identifies these wait events as the problem.
6. **Difficult wait events**—Methods for analyzing and solving wait events when Statspack or AWR data are not enough.
7. **Sampling techniques**—The only way to solve the majority of wait bottlenecks is with a technique called sampling. Sampling can be done on any version of Oracle from version 7 onward, but is automated in Oracle 10*g* with the Active Sessions History feature (ASH).
8. **Active Session History**—Active Session History, often abbreviated as ASH, revolutionizes performance tuning by providing an unprecedented depth of analysis. See how the power of ASH can improve your life.
9. **ASH report**—The performance tool that is probably the least used, but is the most powerful one available.
10. **Oracle OEM 10*g***—How the Oracle OEM 10*g* performance interface radically simplifies performance tuning and unifies all of the above techniques in one simple interface.

Register at **www.nocoug.org**. *Seating is limited, so register early.*

simplest is just reading v$session_wait every second and storing the data. Oracle does this automatically in 10*g* but it has been possible in every version since Oracle 7 by running the following query:

```
column sid format 999
column username format a10
column serial format 99999
column hash_value format 9999999999
column event format a15
column p1 format 9999999999
column p2 format 99999999
column p3 format 9999

select
              nvl(s.username,s.program) username,
              s.sid sid,
              s.serial# serial,
              s.sql_hash_value hash_value,
              substr(decode(w.wait_time,
                     0, w.event,
                    'ON CPU'),1,15) event ,
              w.p1  p1,
              w.p2  p2,
              w.p3  p3
from           v$session        s,
               v$session_wait   w
where     w.sid=s.sid
        and s.status='ACTIVE'
        and s.type='USER'
```

I joined v$session_wait with v$session to get the sql hash value as well. ASH can be extended with similar joins to build up a rich source of sampled data.

From the above query we get output like this.

```
USERNAME   SID   SERIAL   HASH_VALUE EVENT
P1       P2       P3
---------- ----- -------- ---------- --------------------
-------- -------- ----
SYS        64    8717     4116021597 PL/SQL lock timer
300      0        0
SYS        58    19467     961168820 ON CPU
16508152 1        0
STARGUS    71    6251     1311875676 direct path write
201      2155902  127
(CJQ0)     9     1                 0 rdbms ipc message
500      0        0
```

In the case of the buffer busy wait, I've been running the above query and storing it in a table that I call v$ash. I can now read v$ash and look for the detail information I need to solve the buffer busy wait problem. First I want to know what kind of buffer busy waits were happening, read or write. Read or write can be determined from the P3 column which tells us whether the buffer busy wait was a read or write bottleneck. (The P1, P2 and P3 columns have a different meaning for each wait event.) If P3 is 200 or greater the problem is write contention, if P2 is less than 200 it is a simple IO wait.

```
SQL> Select count(*), p3
     from v$ash
     where event = 'buffer busy waits'
     group by p3;

  COUNT(*) P3
---------- ----
      3423 220
```

We see that all the P3 values are 220 thus this is a write problem (below 200 is IO, above 200 is write contention). Write problems can happen for all sorts of reasons depending on the block type the writing is occurring on, so in order to solve this issue we need to know what kind of block the problem is occurring on. We can find this out by looking at p1 and p2 for the buffer busy waits which represents the file and the block:

```
select count(*),  p1 filen, p2 blockn , hash_value
from v$ash
where event='buffer busy waits'
group by p1, p2, hash_value;

  COUNT(*) FILEN    BLOCKN
---------- -------- --------
         1   11       90644
         2   11       90651
         3   11       98233
         1   11       104767
         3   11       113291
         1   11       119842
         1   11       119856
         3   11       121632
         1   11       126334
```

I'll just pick the last block and find out what object it is coming from:

```
column segment_name format a30
select owner,
segment_name,
segment_type,
block_id, blocks+block_id
from dba_extents
where file_id = 1
and 126334 between block_id AND block_id + blocks-1;

OWNER      SEGMENT_NAME       SEGMENT_TY   BLOCK_ID
BLOCKS+BLOCK_ID
---------- ------------------ ---------- ---------- -----
----------
SYSTEM     TOTO1               TABLE          125201
127249
```

From this result I know it's a data block of a table and not a header block (because the header block would be the first block in the object). Now the question is what are the sessions doing? I can look at the sql and see what they are executing:

```
SQL> select count(*),  p1 filen, p2 blockn , hash_value
  2  from v$ash
  3  where event='buffer busy waits'
  4  group by p1, p2, hash_value;

  COUNT(*) FILEN    BLOCKN   HASH_VALUE
---------- -------- -------- ----------
         3 1        94609     558666863
         2 11       81163     558666863
         2 11       87123     558666863
         1 11       90644     558666863
         2 11       90651     558666863
         3 11       98233     558666863
         1 11       104767    558666863
         3 11       113291    558666863
         1 11       119842    558666863
         1 11       119856    558666863
         3 11       121632    558666863
         1 11       126334    558666863
```

I can see that the same SQL is causing all the "buffer busy waits" because the hash value is the same. Now I can find out what that sql statement is by:

```
select sql_text
from v$sqltext
where hash_value=558666863;

SQL_TEXT
----------------------------------------------------------
INSERT into toto1 values (:b1,lpad('a',1000,'a'))
```

With this final piece of information it looks like the segment is probably missing free lists or not in an ASSM tablespace. Let's check these possibilities.

```
select  FREELISTS, TABLESPACE_NAME
from dba_tables
where table_name='TOTO1' and owner='SCOTT';

 FREELISTS TABLESPACE_NAME
---------- ------------------------------
         1 USERS
```

```
select tablespace_name, SEGMENT_SPACE_MANAGEMENT
from dba_tablespaces;

TABLESPACE_NAME                SEGMEN
------------------------------ ------
SYSTEM                         MANUAL
UNDOTBS1                       MANUAL
TEMP                           MANUAL
USERS                          MANUAL
TOOLS                          AUTO
```

We see that that the table is in a table space, USERS, that does not have ASSM active.

In summary, the table TOTO1 has wait contention on writes to blocks and multiple sessions are concurrently inserting into it and it has no free lists nor is it in an Automatic Segment Space Managed (ASSM) table space. This causes all sessions inserting to insert into the same block, the last free one on the one free list and thus fight over exclusive write access to that last block. The solutions is to either put the segment into a ASSM table space and let Oracle handle the contention or put free lists on the segment which is more precise. If I choose free lists, then how many do I need? I need as many as there are concurrent users inserting into the table which I can get with

```
select count(*), sid, serial  from v$ash where
event='buffer busy waits'
group by sid, serial;

  COUNT(*) SID   SERIAL
---------- ----- --------
         7 12    79
         4 14    99
         4 19    770
         8 20    176
```

There were 4 sessions inserting during the period of ASH data I looked at, thus in my case I need 4 free lists.

Extended Power of ASH

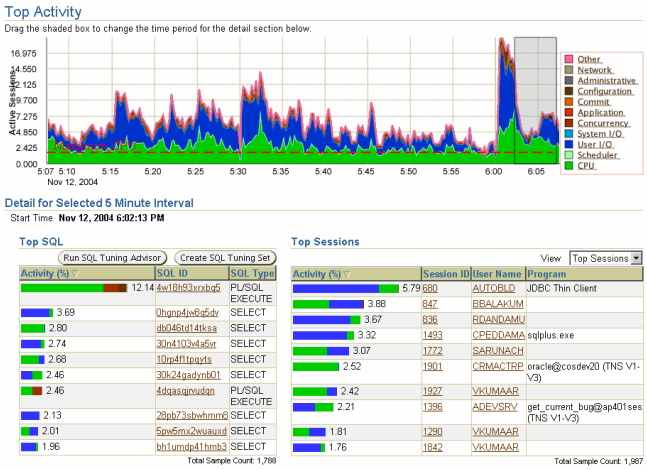
The above example is simplified. Two other important pieces of data missing in the above example that also need to be tracked are sql_text and sample_time. Sample_time is needed to group data by performance bottleneck time periods and sql_text should be collected on an ongoing basis in case the SQL found in v$session_wait gets kicked out of the shared pool. At the end of the day collecting ASH becomes quite complicated with the need to partition and purge data, send data to a different repository to reduce overhead on the target and the need to collect other bits of information like maps of user# to username and object# to object name. To see a package that does much of this, refer to **www.perfvision.com/ash.php**.

Even with a good collection running, it takes some non-trivial sql to mine the richness of the data.

Even with a good set of scripts to analyze the data it takes a number of well written sql statements to drill down sufficiently into the problem.

Luckily 10*g* supplies all of this functionality. Oracle 10*g* automatically collects all of the data, analyzes all the wait events, reports on bottlenecks, and supplies solutions.

On top of this, the data is displayed in a graphically in OEM 10*g* in a way that condenses dozens of sql statements that would take time and effort to run and comprehend into visual that can be immediately understood:



Even in this reduced graphic we can quickly and easily see that there is a performance problem on the machine. In this case there is clearly a CPU bottleneck, because the Top Activity page reports that CPU (in the top chart, the green) is above the maximum CPU on the machine (the dotted red line). This means that the machine is at 100% CPU and there are processes in the run queue. We can also see the chart at the bottom right that one SQL statement is using much more CPU than any other sql statement, so this is the statement to tune. It only takes a second to scan the graphic and see where the problem is and it's all based on ASH.

Summary

ASH is a leap in performance tuning technology. It took a clear vision of the future and a bit of a leap of faith to let go of the compulsive need to have exact measurements and instead embrace sampling a technology that statistically approximates the same value with little loss of accuracy but at the same time brings a great wealth of information necessary for analyzing the performance of a database.

ASH can be simulated on any version of Oracle since v7 with the simple act of repeatedly sampling and storing the values from v$session_wait into a historic table. Although the idea is easy, the implementation turns out to be fairly complex. Luckily Oracle implements not only ASH in version 10*g* but more importantly Oracle implements a diagnostic package that reads the data, analyzes the data and report the findings and solutions automatically so you don't even need to know all the esoteric wait analysis. The best part of all is OEM 10*g* exposes all the data and analysis graphically provide an interface that communicates quickly and clearly the date of the system. ▲

Kyle Hailey is a regular speaker at NoCOUG conferences and needs very little introduction to NoCOUG members. He is a founding member of the Oak Table and co-author of Oracle Insights: Tales of the Oak Table, *together with other NoCOUG regulars like Jonathan Lewis, Cary Millsap, and Gaja Krishna Vaidyanatha. He had a long and distinguished career at Oracle and designed the performance tuning module in Oracle Enterprise Manager 10g.*