Advanced Performance Diagnostics: What the GUI (Does and) Doesn’t Show You

Nicholas J. Donatone
Principal Grid Sales Consultant
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Agenda

- Review of Performance Methodology
- Review - AWR versus ASH
- Interesting Reports
- Mining your data
Why Oracle Enterprise Manager?
• Leader in the complete enterprise application stack
• Built-in manageability in every tier
• Integrated manageability across the entire stack
Oracle Enterprise Manager

*Increases Business Efficiency*

- Manage applications top-down, from the business perspective by understanding user experiences and business impact of IT issues.

- Manage entire application lifecycle to increase business agility with comprehensive application quality management and compliance solutions.

- Reduce operational costs through intelligent diagnostics and automated IT processes.
Oracle’s Performance Methodology

- Methodology has evolved with each release
  - Oracle 7
    - Wait events instrumentation
    - BSTAT, ESTAT
  - Oracle 8
    - STATSPACK
  - Oracle 10g and 11g
    - Enhanced Time-Wait Model
    - “Database Time (DB)” Based Methodology
Oracle’s Performance Methodology

- How to tune your system for a given workload?
  - Identify operations consuming most DB Time
  - Identify resource/capacity related bottlenecks
  - Reduce “DB Time” consumed for the workload

- EM embodies Methodology + Best Practice
  - Workflows based on Methodology
  - Problem determination is few mouse clicks away
EM Performance Page

- How do you tune an Oracle database using EM’s Performance Page?
  - Simplest Answer: “Follow ADDM Recommendations”
  - Simple Answer: “Click on the biggest block of color”
AWR versus ASH
Automatic Workload Repository (AWR)

- **Built-in, automatic performance statistics data warehouse**
- **In-memory statistics**
  - AWR Statistics
  - ASH

**SYSAUX**
- AWR Data
- Snapshots 1 to 4
- 7:00 a.m., 8:00 a.m., 9:00 a.m., 10:00 a.m.

**ADDM finds top problems**

**Eight days**

**DBA**
- Eight days
- DBA_%

**V$**
AWR

- Built-in workload and performance statistics repository in the database
- Automatically Captures Workload Data
- Stores different classes of data:

<table>
<thead>
<tr>
<th></th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter Statistics</td>
<td>Number of Executions</td>
</tr>
<tr>
<td>Time Statistics</td>
<td>DB Time</td>
</tr>
<tr>
<td>Metrics / Rates</td>
<td>Physical Reads / Second</td>
</tr>
<tr>
<td>SQL Statistics</td>
<td>Disk Reads (Per SQL statement)</td>
</tr>
<tr>
<td>Sampled Data</td>
<td>Session Waits</td>
</tr>
</tbody>
</table>
AWR data

- During snapshots, flushed from V$ views to DBA_HIST_* tables
- Interesting Performance tables:
  - DBA_HIST_SNAPSHOT
    - Snapshots in the AWR
    - Join to other tables to constrain the time frame
  - DBA_HIST_SYSTEM_EVENT
    - Information on total waits and times for an event
  - DBA_HIST_SYS_TIME_MODEL
    - System Time Model statistics
  - DBA_HIST_SQLSTAT
    - SQL statistics over time
Active Session History (ASH)

• ASH is session level data

• Active sessions sampled and persisted in-memory
  • Sampling interval = 1 second
  • V$ACTIVE_SESSION_HISTORY
  • Foreground and background sessions are sampled

• On-disk persistence
  • DBA_HIST_ACTIVE_SESS_HISTORY

• ASH is a many-dimensional FACT table
  • Dimensions are V$SESSION columns
  • Fact is that DB time was accumulating over these dimensions

• ASH is a system-wide trace of what happened
# Active Session History (ASH)

<table>
<thead>
<tr>
<th>Time</th>
<th>SID</th>
<th>Module</th>
<th>SQL ID</th>
<th>State</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:38:26</td>
<td>213</td>
<td>Book by author</td>
<td>qa324jffritcf</td>
<td>WAITING</td>
<td>db file sequential read</td>
</tr>
<tr>
<td>7:42:35</td>
<td>213</td>
<td>Get review id</td>
<td>aferv5desfzs5</td>
<td>CPU</td>
<td></td>
</tr>
<tr>
<td>7:50:59</td>
<td>213</td>
<td>Add to cart</td>
<td>hk32pekfcbdfcr</td>
<td>WAITING</td>
<td>buffer busy wait</td>
</tr>
<tr>
<td>7:52:33</td>
<td>213</td>
<td>One click</td>
<td>abngldf95f4de</td>
<td>WAITING</td>
<td>log file sync</td>
</tr>
</tbody>
</table>
ASH

- Can be used for
  - Transient performance problems

- Targeted performance analysis by various dimensions
  - SQL_ID
  - session
  - module
  - service
  - wait_class
## AWR versus ASH Summary

<table>
<thead>
<tr>
<th>Feature</th>
<th>AWR</th>
<th>ASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance Wide data</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Time Based data</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Counts/occurrence data</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Analyze any time period</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Detailed session level data</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Individual wait event data</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Sampled data</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Time based analysis</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Resources in
$ORACLE_HOME/rdbms/admin

• Available report scripts
  • Common reports
    • awrrpt.sql
    • ashrpt.sql
    • addmrpt.sql

• Less Well Known reports
  • ashrpti.sql
  • awrddrpt.sql
  • awrsqrpt.sql
  • spawrrac.sql
ashrpti.sql

- ASH report for dimensions in addition to time
  - SQL_ID
  - session
  - service
  - wait_class
  - client_id
awrddrpt.sql

- AWR Compare Periods Report
  - Good for finding out ‘what changed’ in the instance
- Use Case
  - Overall system performance resulting from SQL tuning
  - Two snapshots - before and after SQL tuning

### System Configuration Comparison

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>Diff</th>
<th>%Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGA Target:</td>
<td></td>
<td></td>
<td>0M</td>
<td>0.00</td>
</tr>
<tr>
<td>Buffer Cache:</td>
<td>240M</td>
<td>240M</td>
<td>0M</td>
<td>0.00</td>
</tr>
<tr>
<td>Shared Pool Size:</td>
<td>336M</td>
<td>336M</td>
<td>0M</td>
<td>0.00</td>
</tr>
<tr>
<td>Large Pool Size:</td>
<td>4M</td>
<td>4M</td>
<td>0M</td>
<td>0.00</td>
</tr>
<tr>
<td>Java Pool Size:</td>
<td>12M</td>
<td>12M</td>
<td>0M</td>
<td>0.00</td>
</tr>
<tr>
<td>Streams Pool Size:</td>
<td>0M</td>
<td>0M</td>
<td>0M</td>
<td>0.00</td>
</tr>
<tr>
<td>Log Buffer:</td>
<td>4,848K</td>
<td>4,848K</td>
<td>0K</td>
<td>0.00</td>
</tr>
<tr>
<td>PGA Aggregate Target:</td>
<td>M</td>
<td>M</td>
<td>0M</td>
<td>0.00</td>
</tr>
<tr>
<td>Undo Management:</td>
<td>AUTO</td>
<td>AUTO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- System wide ‘Logical Reads per TXN’ significantly reduced

<table>
<thead>
<tr>
<th>Load Profile</th>
<th>1st per sec</th>
<th>2nd per sec</th>
<th>%Diff</th>
<th>1st per txn</th>
<th>2nd per txn</th>
<th>%Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB time:</td>
<td>4.54</td>
<td>0.20</td>
<td>-95.59</td>
<td>14.14</td>
<td>0.59</td>
<td>-95.83</td>
</tr>
<tr>
<td>CPU time:</td>
<td>4.53</td>
<td>0.20</td>
<td>-95.58</td>
<td>14.09</td>
<td>0.58</td>
<td>-95.88</td>
</tr>
<tr>
<td>Redo size:</td>
<td>5,351.08</td>
<td>5,069.74</td>
<td>-5.26</td>
<td>16,651.18</td>
<td>14,855.46</td>
<td>-10.78</td>
</tr>
<tr>
<td>Logical reads:</td>
<td>1,212,747.47</td>
<td>10,212.59</td>
<td>-99.16</td>
<td>3,773,757.58</td>
<td>29,925.17</td>
<td>-99.21</td>
</tr>
</tbody>
</table>
AWR Report for a particular SQL Statement
- Useful for researching individual SQL statement plan changes over time
- Use Case
  - Single SQL statement, before and after tuning
  - Buffer gets substantially decreased

**Plan Statistics**

**Before tuning**

<table>
<thead>
<tr>
<th>Stat Name</th>
<th>Statement Total</th>
<th>Per Execution</th>
<th>% Snap Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed Time (ms)</td>
<td>571,421</td>
<td>2,747.22</td>
<td>41.67</td>
</tr>
<tr>
<td>CPU Time (ms)</td>
<td>569,652</td>
<td>2,739.72</td>
<td>41.71</td>
</tr>
<tr>
<td>Executions</td>
<td>208</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffer Gets</td>
<td>145,778,328</td>
<td>700,857.35</td>
<td>39.82</td>
</tr>
</tbody>
</table>

**After tuning**

<table>
<thead>
<tr>
<th>Stat Name</th>
<th>Statement Total</th>
<th>Per Execution</th>
<th>% Snap Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed Time (ms)</td>
<td>33,905</td>
<td>69.48</td>
<td>55.37</td>
</tr>
<tr>
<td>CPU Time (ms)</td>
<td>33,920</td>
<td>69.51</td>
<td>56.34</td>
</tr>
<tr>
<td>Executions</td>
<td>488</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffer Gets</td>
<td>646,144</td>
<td>1,736.00</td>
<td>27.52</td>
</tr>
</tbody>
</table>
spawrrac.sql

- Generates global AWR report for all nodes on a cluster
- In 11g
- Supplements Global ADDM in 11g
- Has limitations
  - Text only
spawrrac.sql

- Use Cases
  - How localized are my buffer accesses?
  - How evenly is my workload distributed?
  - What is my cluster-wide physical I/O?

<table>
<thead>
<tr>
<th>I#</th>
<th>Buffer Access</th>
<th>Logical Reads</th>
<th>Physical Reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>92.71% Local</td>
<td>134,798,497</td>
<td>5,969,938</td>
</tr>
<tr>
<td>2</td>
<td>95.45% Local</td>
<td>140,324,093</td>
<td>3,371,883</td>
</tr>
<tr>
<td>3</td>
<td>97.19% Local</td>
<td>39,300,537</td>
<td>477,181</td>
</tr>
<tr>
<td>4</td>
<td>96.51% Local</td>
<td>58,850,603</td>
<td>1,227,469</td>
</tr>
</tbody>
</table>

avg  | 93,318,433 | 2,761,618
sum  | 373,273,730 | 11,046,471

Global Cache Efficiency Percentages

<table>
<thead>
<tr>
<th>I#</th>
<th>Local %</th>
<th>Remote %</th>
<th>Disk %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>92.71</td>
<td>2.86</td>
<td>4.43</td>
</tr>
<tr>
<td>2</td>
<td>95.45</td>
<td>2.14</td>
<td>2.40</td>
</tr>
<tr>
<td>3</td>
<td>97.19</td>
<td>1.60</td>
<td>1.21</td>
</tr>
<tr>
<td>4</td>
<td>96.51</td>
<td>1.41</td>
<td>2.08</td>
</tr>
</tbody>
</table>
spawrrac.sql

- Significant enhancements planned
  - HTML
  - Subset of Instances
  - Global Diff Report
Additional AWR Scripts

- Moving AWR Data
  - Use Cases
    - To offload analysis from production database
    - To preserve data longer than the default on the production system

- awrextr.sql
  - extract data from awr
- awrload.sql
  - load data from awrextr dump file
Using AWR Data For Trending

• Common use cases of AWR data are already presented in EM
• Data in DBA_HIST_* tables can be mined to produce data for targeted questions for your company

• Following are some examples to get you started
• These examples were produced using charting capability of SQL Developer
• SQL for these reports are in the appendix
Average Active Sessions

- Average Active Sessions = DBtime / Elapsed Time
  - DBtime
    - Time foreground processes using CPU or non-idle wait events
    - From `dba_hist_sys_time_model`
  - Elapsed Time
    - Calculated from begin / end interval from `dba_hist_snapshot`

- Use Case
  - Longer term trending of RAC cluster
  - Can choose different time ranges
  - Includes data from multiple RAC instances
  - Not broken down by wait events
Average Active Sessions
define num_days=1
select to_char(end_interval_time,'mm-dd hh24') snap_time,
                           instance_number,
                           avg(v_ps)     pSec
from (select end_interval_time,
              instance_number,
              v/ela            v_ps
     from (select trunc(s.end_interval_time,'hh24') end_interval_time,
                      instance_number,
                      (case when s.begin_interval_time = s.startup_time
                      then value
                      else value - lag(value,1) over (partition by sy.stat_id,
                                                        sy.dbid,
                                                        sy.instance_number,
                                                        s.startup_time
                      order by sy.snap_id)
                      end)/1000000  v
                      , (cast(s.end_interval_time as date) - cast(s.begin_interval_time as date))*24*3600 ela
     from dba_hist_snapshot s,
         dba_hist_sys_time_model sy
     where s.dbid  = sy.dbid
       and s.instance_number = sy.instance_number
       and s.snap_id     = sy.snap_id
       and sy.stat_name  = 'DB time'
       and s.end_interval_time > trunc(sysdate) - &num_days)
group by to_char(end_interval_time,'mm-dd hh24'), instance_number
order by to_char(end_interval_time,'mm-dd hh24'), instance_number
/)
Average Active Sessions by Wait Class

• Use Case
  • Longer term trending of RAC cluster
  • Can choose different time ranges
  • Broken down by wait events
  • Includes data from multiple RAC instances
  • Could focus on one class of wait events

• Average Active Sessions = DBtime / Elapsed Time
  • Data comes from
    • `dba_hist_sys_time_model`
    • `dba_hist_snapshot`
Average Active Sessions by Wait Class
CPU Load

• Data is from `dba_hist_osstat`

• Includes data from two RAC instances

• Data captured during every snapshot, averaged over snapshot time period

• Doesn’t show short term fluctuations
CPU Load
Real Time SQL Monitoring

• Explain Plan Shows Progress During SQL Execution
• In 11.1.0.7 DBControl
Real Time SQL Monitoring

- In 11.1.0.6
  - DBMS_SQLTUNE.REPORT_SQL_MONITOR

- Views
  - v$sql_monitor
  - v$sql_plan_monitor
ORACLE IS THE INFORMATION COMPANY
Appendix
define num_days=1
select to_char(end_interval_time,'mm-dd hh24') snap_time,
     , instance_number
     , avg(v_ps) pSec
from (select end_interval_time
     , instance_number
     , v/ela v_ps
from (select trunc(s.end_interval_time,'hh24') end_interval_time
     , s.instance_number
     , (case when s.begin_interval_time = s.startup_time
     then value
     else value - lag(value,1) over (partition by sy.stat_id,
     , sy.dbid
     , sy.instance_number
     , s.startup_time
     order by sy.snap_id)
     end)/1000000 v
     , (cast(s.end_interval_time as date) - cast(s.begin_interval_time as date))*24*3600 ela
from dba_hist_snapshot s
     , dba_hist_sys_time_model sy
where s.dbid = sy.dbid
     and s.instance_number = sy.instance_number
     and s.snap_id = sy.snap_id
     and sy.stat_name = 'DB time'
     and s.end_interval_time > trunc(sysdate) - &num_days)
group by to_char(end_interval_time,'mm-dd hh24'), instance_number
order by to_char(end_interval_time,'mm-dd hh24'), instance_number
/
define num_days = 1
select to_char(end_time,'mm-dd hh24') snap_time,
       wait_class,
       sum(pSec) avg_sess
from
    (select end_time,
           wait_class,
           p_tmfg/1000000/ela pSec
    from
      (select end_time,
              wait_class,
              p_tmfg/1000000/ela pSec
           from
             (select trunc(s.end_interval_time,'hh24') end_time,
                    (cast(s.end_interval_time as date) - cast(s.begin_interval_time as date)) * 24 * 3600 ela
                     s.snap_id,
                     wait_class,
                     e.event_name,
                     case when s.begin_interval_time = s.startup_time
                          then e.time_waited_micro_fg
                          else e.time_waited_micro_fg
                          else e.time_waited_micro_fg
                          - lag(time_waited_micro_fg) over (partition by event_id,
                                                          e.dbid,
                                                          e.instance_number,
                                                          s.snap_id
                                                          order by e.snap_id)
                          end    p_tmfg
               from dba_hist_snapshot s
                    , dba_hist_system_event e
               where s.dbid = e.dbid
                   and s.instance_number = e.instance_number
                   and s.snap_id = e.snap_id
                   and s.end_interval_time > trunc(sysdate) - &num_days
                   and e.wait_class != 'Idle'
           union all
           /* Continued on next slide */
Active Sessions Per Wait Class SQL

/* Continued from previous slide */
select trunc(s.end_interval_time,'hh24') end_time,
  (cast(s.end_interval_time as date) - cast(s.begin_interval_time as date)) * 24 * 3600 ela,
  s.snap_id,
  t.stat_name wait_class,
  t.stat_name event_name,
  case when s.begin_interval_time = s.startup_time
    then t.value
    else t.value
    - lag(t.value) over (partition by stat_id,
      t.dbid,
      t.instance_number,
      s.startup_time
    order by t.snap_id)
  end p_tmfg
from dba_hist_snapshot s,
  dba_hist_sys_time_model t
where s.dbid = t.dbid
  and s.instance_number = t.instance_number
  and s.snap_id = t.snap_id
  and s.end_interval_time > trunc(sysdate) - &num_days
  and t.stat_name = 'DB CPU')
group by to_char(end_time,'mm-dd hh24'), wait_class
order by to_char(end_time,'mm-dd hh24'), wait_class
/
OS CPU Busy SQL

define num_days = 1
select to_char(trunc(end_interval_time,'hh24'),'mm-dd hh24') snap_time,
       instance_number,
       busy/decode(busy+idle,0,null,busy+idle)*100 pct_busy
from (select s.snap_id,
         s.instance_number,
         s.dbid,
         s.end_interval_time,
         os.stat_name,
         case when s.begin_interval_time = s.startup_time
              then os.value
              else os.value - lag(os.value,1) over (partition by os.stat_name,
                                                       s.instance_number,
                                                       s.dbid,
                                                       s.startup_time
                                                       order by os.snap_id)
         end delta_v
      from dba_hist_snapshot s,
           dba_hist_osstat os
      where s.snap_id = os.snap_id
            and s.instance_number = os.instance_number
            and s.dbid = os.dbid
            and s.end_interval_time > trunc(sysdate) - &num_days
            and os.stat_name in ('BUSY_TIME','IDLE_TIME'))
pivot (sum(delta_v)
       for stat_name in ('BUSY_TIME' busy
                           ,'IDLE_TIME' idle))
order by to_char(trunc(end_interval_time,'hh24'),'mm-dd hh24'), instance_number
/