Historical Performance Analysis Using AWR

April 2009

ORACLE

INTRODUCTION

The Automatic Workload Repository (AWR) in Oracle Database 10g and 11g stores a wealth of data regarding database performance. This data is a key component of the Diagnostics and Tuning pack and is licensed as part of the Diagnostics Pack. Oracle Enterprise Manager provides a graphical user interface to this data, allowing you to display and diagnose the performance of your database. The standard AWR report offers an alternate view into these statistics, by providing detailed information for a specified time interval.

There are times when viewing longer term historical information would be useful. Viewing this longer term historical information could help pinpoint when a performance problem may have started. Was the onset sudden, or did the workload gradually increase over a few weeks?

AWR contains a wealth of data that can be used for historical performance analysis.

The use of AWR views are licensed with the Oracle Diagnostic Pack.

Viewing the historical performance of your workload can be helpful in identifying peak hours and peak days. Similarly, certain historical characteristics of the workload, such as I/O requests per second, or user calls per second, may be useful to look at to see if the workload remains constant, is increasing or decreasing.

A trend of a high load SQL statement may be useful to determine whether the characteristics of a SQL statement is changing – is it using more CPU, is it taking more elapsed time per execution, is it retrieving more data per execution, or is it simply getting executed more often?

Longer term charts of performance data can also help in capacity planning. By identifying growing usage of key resources, you can identify possible future resource constraints.

All of the above information is available in the AWR, but may not be easily visible in the standard AWR report. Oracle Enterprise Manager does display the performance information in graphs along several dimensions. However, the data that is present in the AWR can be analyzed in many more ways than are possible to present in a limited number of graphs. This paper gives several examples of graphs you can create, to illustrate the kind of information you can mine.

By default, the AWR retention period is eight days. To do longer term historical analysis you will need to either increase the default retention, or move the data to another database. Details on doing so are discussed later in this paper.

With a few queries, and any charting utility the information can be extracted from AWR in order to aid performance diagnosis and analysis. In the following examples, we use Oracle SQL Developer to chart the data.

FIRST THINGS FIRST

The examples in this paper are meant to provide a guideline that you can use to query your own AWR data. The queries can be easily modified to display the specific data of interest that you want to analyze.

The query results have been structured to match the format used by SQL Developer for charting (group, series, value) [Harper]. If you are using a different charting utility, you may need to pivot the results to match the requirements of your charting tools.

Some of the queries assume a default snapshot interval of 1 hour. If you are using a different snapshot interval that is not a multiple of 1 hour, those queries may need to be modified accordingly.

Query Basics

Delta values for trending are computed using the lag() function.

Most of the AWR tables store statistic values from instance startup. For trending purposes, several queries in this paper use the lag() function in order to compute statistic values for each time interval. The most common format for computing these delta values for each interval is:

```
case when s.begin_interval_time = s.startup_time
     then value
     else value - lag(value,1) over (partition by stat_id
                                                 , instance_number
                                                 , dbid
                                                 , s.startup_time
                                           order by snap_id)
 end  delta_v
```

`s.begin_interval_time = s.startup_time` checks for the first snapshot after an instance startup. In this case, the current value is returned as-is.

The `lag()` function returns the value for the statistic for the previous snapshot. [SQLREF 008]. The difference between the current value and the previous value is the delta that was incurred during the snapshot interval.

For the first snapshot in the range, the `lag()` function will return NULL. This will not be used in the graph, as we will not have the delta value for that first snapshot in the range.

AVERAGE ACTIVE SESSIONS

DB time is a key concept in the Oracle Performance Method and is reflected in the statistics stored in the AWR. DB time is defined as the amount of time foreground processes are spending in the database, either using CPU or in non-idle wait events. The number of average active sessions during an interval is calculated as:

- *AverageActiveSessions = DBtime / ElapsedTime*

Average Active Sessions can be calculated as DB time / Elapsed Time

The value for DB time is stored in the view `DBA_HIST_SYS_TIME_MODEL`, while the elapsed time can be calculated from the begin/end interval times in `DBA_HIST_SNAPSHOT`.

Average Active Sessions is a quick indication of database performance and health. If the number of average active sessions is low, that is an indication that the database is idle. If the number of active sessions increases drastically, that is usually an indication of a performance problem – the sessions are either waiting or using CPU.

Figure 1 shows the average active sessions for a RAC database with 2 nodes over the past 7 days. The graph shows a peak of 38 average active sessions, and minimal to no activity over a span of 2 days.

The SQL query that produced the graph in Figure 1 can be found in Appendix A.

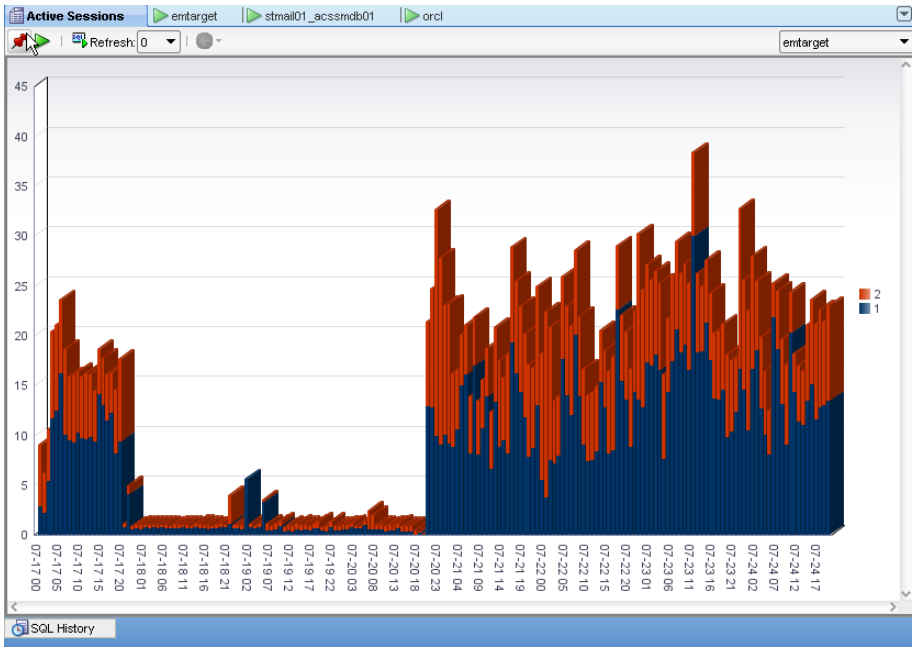

Figure 1: Average Active Sessions on Instance 1 and 2

AVERAGE ACTIVE SESSIONS BY WAIT CLASS

Another interesting graph is a view of average active sessions but broken down by wait class, rather than by instance. This graph can be a good indication of the efficiency of your database. It can also highlight scalability issues in your application or database.

The graph in Figure 2 shows the same information that is displayed by the historical view on the Performance Page in Enterprise Manager. The additional value is that you can extend the graph for longer time periods when you are mining the AWR data directly.

You can query the view `DBA_HIST_EVENT_NAME` to see a mapping of individual wait events to their wait class.

Figure 2 shows the average active sessions for a RAC database with four nodes over a seven day period, starting with Sunday. The graph shows a peak of roughly 80 average active sessions, with less activity on Saturday and Sunday . We can see that 'User I/O' is the dominant wait class and 'DB CPU' is the second major component of the workload.

The SQL query that produced the graph in Figure 2 can be found in Appendix B.

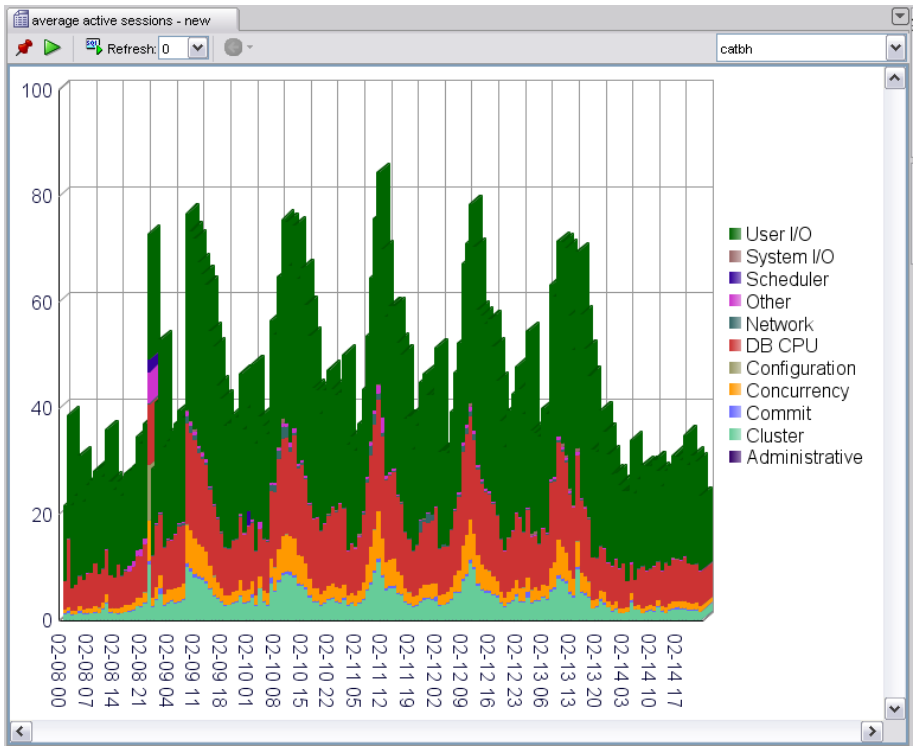

Figure 2: Average Active Sessions by Wait Class

CPU INFORMATION

CPU Information is available in the view `DBA_HIST_OSSTAT`. This data is only captured every snapshot interval, by default one hour. The data displayed will be averaged over the snapshot interval and may not accurately reflect the peaks that occur in the system.

In particular, the CPU load is only meaningful for the time the snapshot was taken. If the load on the system is constant, then this value may be useful for trending. However, if the load patterns fluctuate from minute to minute, this data will likely be less useful.

This graph in Figure 3 displays the CPU Usage for a two node RAC system for a seven day period, broken down by node. The graph runs from Sunday through Saturday. It is easy to see that the CPU usage peaks around noon on business days.

The SQL query that produced the graph in Figure 3 can be found in Appendix C.

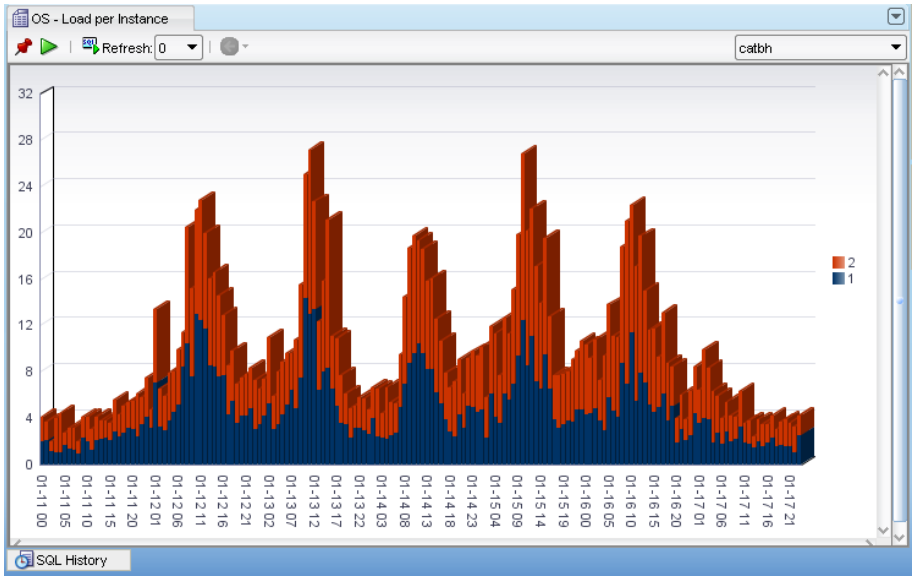

Figure 3: CPU Load

SQL ELAPSED TIME PER EXECUTION

Historical SQL statistics are stored in the view `DBA_HIST_SQLSTAT`. There is a wide variety of information in that view about SQL statement execution. Some important information in that view include data on SQL statement I/O, CPU time and elapsed execution time

Graphing the elapsed execution time of key SQL can alert you to things such as unexpected plan changes, I/O issues, or other types of database contention. In an OLTP environment, there are generally key SQL that are easily identified and that are critical to the workload functioning properly. Reviewing the performance history of these SQL alerts you to possible issues in your environment.

The graph in Figure 4 is a graph of the elapsed time of a single SQL statement on a single node over a week. While there is some variation in execution time, the variation was acceptable for this statement in this OLTP system.

The SQL query that produced the graph in Figure 4 can be found in Appendix D.

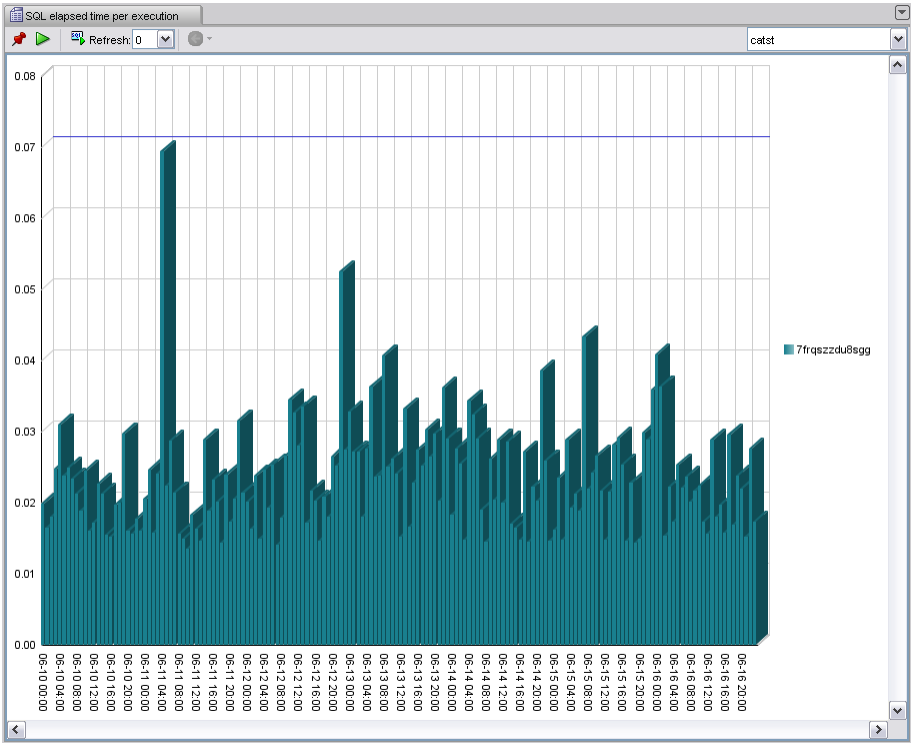

Figure 4: SQL Elapsed Time per Execution

SYSSTAT

The Oracle database keeps track of a large number of statistics about the system activity in the `V$SYSSTAT` view. The database reference guide lists the statistics and their meaning. The 'Instance Activity Stats' section of an AWR report also lists this information. There are system wide statistics about many aspects of database operation. That data is persisted to the AWR in the `DBA_HIST_SYSSTAT` view.

Graphing this information over time can give you insights into various aspects of your system performance.

The graph in Figure 5 is a graph of 'session logical reads' over a week for a four node RAC system. The value of the 'session logical reads' statistic is the number of database blocks that have been read from memory. More technically, it is the sum of 'db block gets' and 'consistent gets'. The graph runs from Sunday through Saturday. As with some of the previous graphs, the system shows expected peaks during the business days.

The SQL query that produced the graph in Figure 5 can be found in Appendix E.

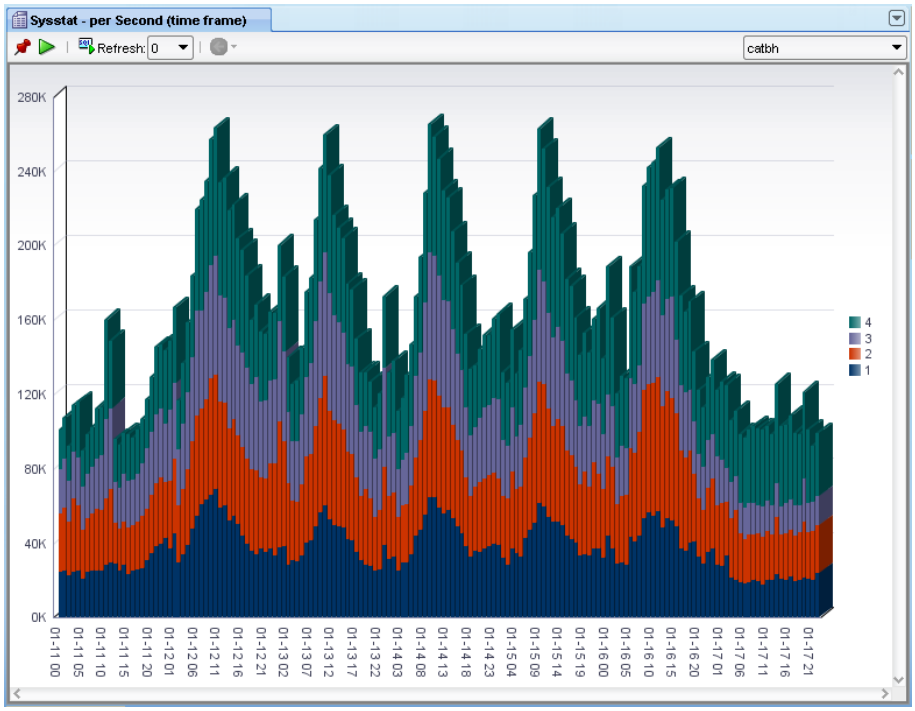

Figure 5: SYSSTAT - Logical Reads

DATABASE DISK SPACE USAGE

Every DBA wants to track the amount of disk space the database is using and how that usage changes over time.

The view `DBA_HIST_TBSPC_SPACE_USAGE` stores information about tablespace usage at each snapshot. It is possible to graph the data at that level of granularity. However, because we want to graph the usage over weeks or months, one data point a day will meet our needs. For the graph below, we've arbitrarily chosen to use the data from the first snapshot of each day. We could have chosen to pick the average size over the day, or we could have chosen the maximum size for each day.

An additional issue with the `TABLESPACE_SIZE` column in the `DBA_HIST_TBSPC_SPACE_USAGE` table is that it stores the size as the number of database blocks. We need to multiply each tablespace size with the blocksize as obtained from the `DBA_HIST_DATAFILE` view.

The graph in Figure 6 is the daily total database size for an eleven terabyte database graphed over a month. It is clear that the database is growing. If we would graph the size for a larger time period, it would be fairly easy to do some linear trending.

The SQL query that produced the graph in Figure 6 can be found in Appendix F.

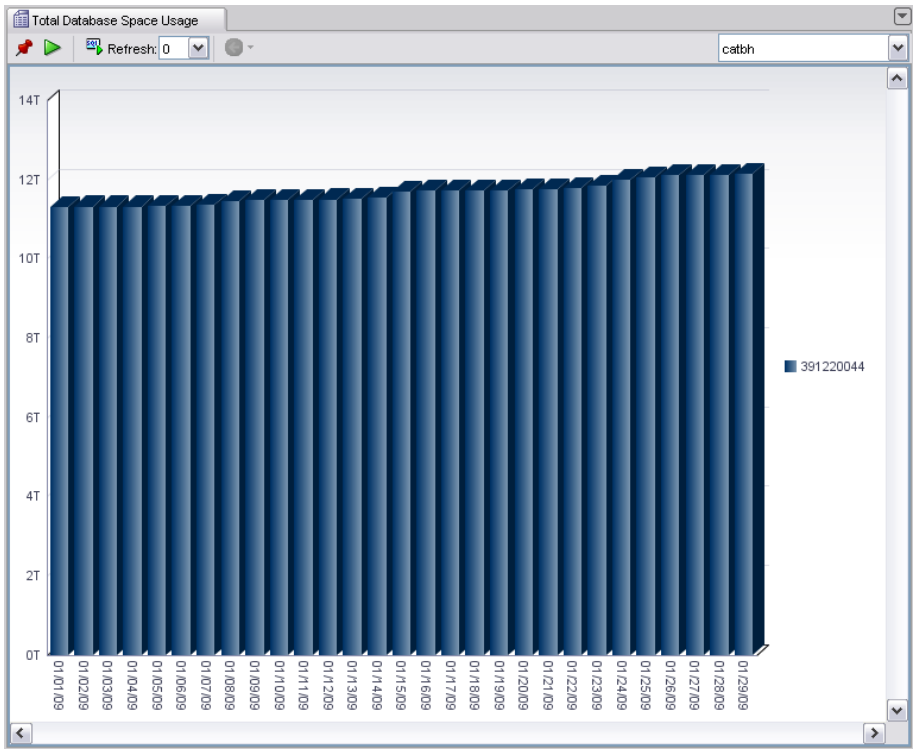

Figure 6: Total Database Size

IOSTAT BY FILETYPE

I/O statistics are stored in the AWR in several ways. The view `DBA_HIST_IOTYPE_FILE` breaks down I/O by type of file, such as data file or log file, and by type of I/O such as single block reads in megabytes or multiblock reads in megabytes. The view `DBA_HIST_IOSTAT_FUNCTION` breaks I/O down by the type of function served by the I/O such as DBWR or LGWR. These views are available in Oracle Database 11g.

Of course, there are many different ways to sum and view the data.

The graph in Figure 7 shows reads for data files for a four node cluster over a week. The bars in the graph represent the number of gigabytes read per hour, summed over the four nodes. There is not as clear a usage pattern here as in some earlier graphs. There is a large spike late on Friday night that might bear further investigation.

The SQL query that produced the graph in Figure 7 can be found in Appendix G.

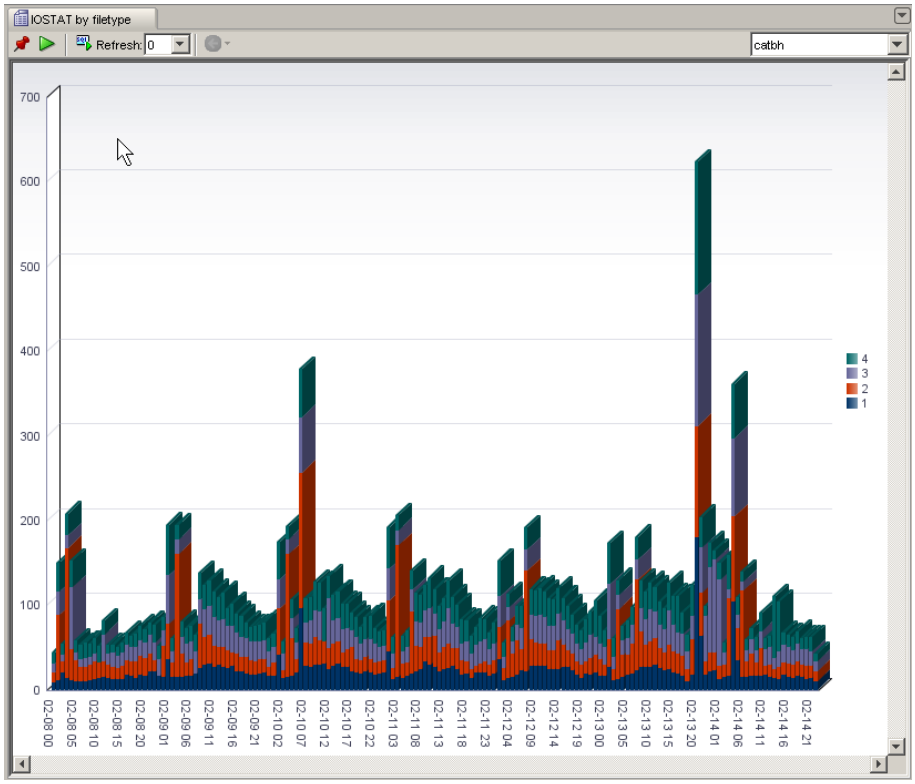

Figure 7 IOSTAT by filetype

PGA USAGE

With the implementation of automatic memory management features in the Oracle database, managing memory and the many parameters that control it has become much simpler in recent versions of Oracle. Automatic memory management of the PGA was introduced in Oracle 9i with the `PGA_AGGREGATE_TARGET` parameter.

There is a section of the AWR report which gives advice on how to size your PGA, the 'PGA MEMORY ADVISORY'. If you want to trend the changes of usage over time in your PGA, you can use the data in the `DBA_HIST_PGASTAT` view.

The graph in Figure 8 charts the statistic 'total PGA allocated' in megabytes over a week for a four node RAC system. The value recorded for that statistic is the value at the time the snapshot was taken and may vary between snapshots. For OLTP systems the value would probably be relatively stable over the snapshot period, while it could vary widely in a DSS system. The 'total PGA allocated' gradually increases over most of the week, which suggests a graph over a longer time period could be interesting. The longer term graph is not produced here for the sake of legibility in the limited space.

The SQL query that produced the graph in Figure 8 can be found in Appendix H.

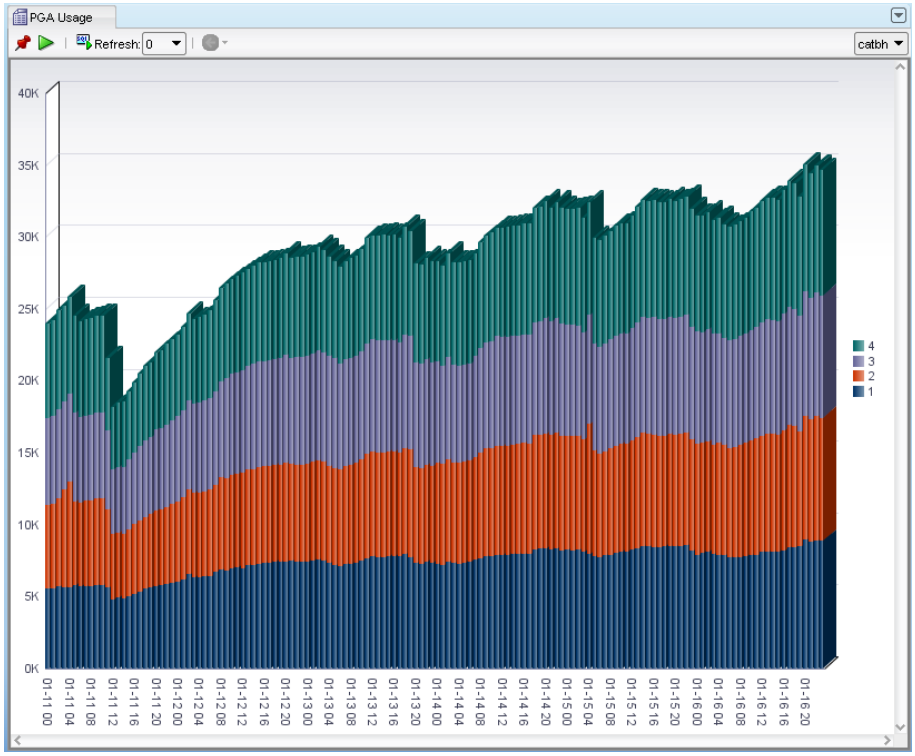

Figure 8: Total PGA In Use

DATA RETENTION

To get the data to do longer term historical analysis, you have several options. By default, the data is retained in the AWR repository for eight days. You can easily increase the retention period.

However, if you are doing intensive analysis, or wish to retain the data for extended periods of time, you might wish to move the data to a non-production database with sufficient space and processing power for analysis.

One simple solution is to export specific AWR tables and import them into a data warehouse database.

Another possibility is to use the two scripts supplied with Oracle database 10.2.0.4 and above. These scripts, awrextr.sql and awrload.sql, can be find in the $ORACLE_HOME/rdbms/admin directory. '`awrextr.sql`' uses Data Pump to extract data for a given snapshot range from your AWR. '`awrload.sql`' can then be used on target database to load the extracted Data Pump file.

CONCLUSION

There is a wealth of data available in the Oracle database AWR. This data can be used for historical performance analysis. This paper has provided a number example SQL and graphs to help you get started mining all the information that is available to you.

LICENSE INFORMATION

The dictionary views referenced in these queries are licensed with Oracle Diagnostic Pack.
(http://download.oracle.com/docs/cd/B28359_01/license.111/b28287/options.htm#CIHIHDDJ)

REFERENCES

[Harper 2007] Harper, Sue 2007. Now Reporting, Oracle Magazine May/Jun 2007. (http://www.oracle.com/technology/oramag/oracle/07-may/o37sql.html)

[SQLREF 008] Oracle Database SQL Language Reference 11g Release 1 (11.1) (http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/functions072.htm#i1327527)

[SQLREF 009] Oracle® Database Reference 11g Release 1 (11.1) (http://download.oracle.com/docs/cd/B28359_01/server.111/b28320/toc.htm)

APPENDIX A: ACTIVE SESSIONS

```
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 round(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
                                                                , s.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 > to_date(:start_time,'MMDDYYYY')
         and s.end_interval_time < to_date(:end_time,'MMDDYYYY') ))
 group by to_char(end_interval_time,'mm-dd hh24'), instance_number
 order by to_char(end_interval_time,'mm-dd hh24'), instance_number
```

APPENDIX B: ACTIVE SESSONS PER WAIT CLASS

```
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 round(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
                          - lag(time_waited_micro_fg) over (partition by event_id
                                                                       , e.dbid
                                                                       , e.instance_number
                                                                       , s.startup_time
                                                                  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 > to_date(:start_date,'MMDDYYYY')
          and s.end_interval_time < to_date(:end_date,'MMDDYYYY')
          and e.wait_class != 'Idle'
        union all
       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(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 > to_date(:start_date,'MMDDYYYY')
          and s.end_interval_time < to_date(:end_date,'MMDDYYYY')
          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
```

APPENDIX C: CPU LOAD PER INSTANCE

```
select to_char(round(s.end_interval_time,'hh24'),'mm-dd hh24') snap_time
     , os.instance_number
     , os.value
  from dba_hist_snapshot s
     , dba_hist_osstat os
 where s.dbid = os.dbid
   and s.instance_number = os.instance_number
   and s.snap_id         = os.snap_id
   and os.stat_name = 'LOAD'
   and s.end_interval_time > to_date(:start_date,'MMDDYYYY')
   and s.end_interval_time < to_date(:end_date,'MMDDYYYY')
order by to_char(trunc(s.end_interval_time,'hh24'),'mm-dd hh24'), os.instance_number
```

APPENDIX D: SQL ELAPSED TIME PER EXECUTION

```
select to_char(round(end_interval_time,'hh24'),'mm-dd hh24') snap_time
     , sql_id
     , sum(elapsed_time_delta/1000000)/decode(sum(executions_delta),0,null,sum(executions_delta))
avg_elapsed
  from dba_hist_sqlstat sq
     , dba_hist_snapshot s
 where sq.sql_id (+) = :sql_id
   and sq.dbid   (+) = s.dbid
   and sq.instance_number (+) = s.instance_number
   and sq.snap_id         (+) = s.snap_id
   and s.end_interval_time > to_date(:start_date,'MMDDYYYY')
   and s.end_interval_time < to_date(:end_date,'MMDDYYYY')
 group by to_char(round(end_interval_time,'hh24'),'mm-dd hh24'), sql_id
 order by to_char(round(end_interval_time,'hh24'),'mm-dd hh24')
```

APPENDIX E: SYSSTAT - LOGICAL READS

```
select to_char(round(end_interval_time,'hh24'),'mm-dd hh24') snap_time
     , instance_number
     , avg(pSec)        perSec
  from ( select end_interval_time
              , instance_number
              , greatest(v/ela,0)   pSec
         from (
            select /*+ leading(s,sn,sy) */ s.snap_id
                 , s.instance_number
                 , s.dbid
                 , s.end_interval_time
                 , case when s.begin_interval_time = s.startup_time
                        then sy.value
                        else sy.value - lag(sy.value,1) over (partition by sy.stat_id
                                                                         , sy.instance_number
                                                                         , sy.dbid
                                                                         , s.startup_time
                                                              order by sy.snap_id)
                   end  v
                 , (cast(end_interval_time as date) - cast(begin_interval_time as date))*24*3600 ela
             from dba_hist_snapshot s
                , dba_hist_sysstat sy
                , dba_hist_stat_name sn
            where s.dbid            = sy.dbid
              and s.instance_number = sy.instance_number
              and s.snap_id         = sy.snap_id
              and s.dbid           = sn.dbid
              and sy.stat_id        = sn.stat_id
              and end_interval_time between to_timestamp(:start_date,'MMDDYYYY')
                                        and to_timestamp(:end_date,'MMDDYYYY')
              and sn.stat_name = 'session logical reads'
         )
      )
group by to_char(round(end_interval_time,'hh24'),'mm-dd hh24'), instance_number
order by to_char(round(end_interval_time,'hh24'),'mm-dd hh24'), instance_number
```

APPENDIX F: DISK SPACE USAGE

```
--  Get the datbase block size from dba_hist_datafile
WITH ts_info as (
   select dbid, ts#, tsname, max(block_size) block_size
   from dba_hist_datafile
   group by dbid, ts#, tsname),
-- Get the maximum snaphsot id for each day from dba_hist_snapshot
snap_info as (
   select dbid,to_char(trunc(end_interval_time,'DD'),'MM/DD/YY') dd, max(s.snap_id) snap_id
   from dba_hist_snapshot s
   where s.end_interval_time > to_date(:start_time,'MMDDYYYY')
     and s.end_interval_time < to_date(:end_time,'MMDDYYYY')
   group by dbid,trunc(end_interval_time,'DD'))
-- Sum up the sizes of all the tablespaces for the last snapshot of each day
select  s.dd, s.dbid, sum(tablespace_size*f.block_size)
   from dba_hist_tbspc_space_usage sp,
        ts_info                    f,
        snap_info                  s
   where  s.dbid           = sp.dbid
      and s.snap_id        = sp.snap_id
      and sp.dbid          = f.dbid
      and sp.tablespace_id = f.ts#
   group by s.dd, s.dbid
   order by s.dd
```

APPENDIX G: IOSTAT BY FILETYPE

```
select to_char(round(end_interval_time,'hh24'),'mm-dd hh24') snap_time
     , instance_number
     , sum(megabytes) / 1024 Gigabytes
from
(
     select end_interval_time
              , instance_number
              , megabytes
     from
     (
           select  s.snap_id
                 , s.instance_number
                 , s.dbid
                 , s.end_interval_time
                 , case when s.begin_interval_time = s.startup_time
                        then nvl(ft.small_read_megabytes+large_read_megabytes,0)
                        else nvl(ft.small_read_megabytes+large_read_megabytes,0) -
                             lag(nvl(ft.small_read_megabytes+large_read_megabytes,0),1)
                                                  over (partition by ft.filetype_id
                                                                   , ft.instance_number
                                                                   , ft.dbid
                                                                   , s.startup_time
                                                              order by ft.snap_id)
                          end  megabytes
              from dba_hist_snapshot s
                 , dba_hist_iostat_filetype ft
                 , dba_hist_iostat_filetype_name fn
             where s.dbid            = ft.dbid
               and s.instance_number = ft.instance_number
               and s.snap_id         = ft.snap_id
               and s.dbid            = fn.dbid
               and ft.filetype_id        = fn.filetype_id
               and end_interval_time between to_timestamp(:start_date,'MMDDYYYY')
                                         and to_timestamp(:end_date,'MMDDYYYY')
                and fn.filetype_name = 'Data File'
     )
)
group by to_char(round(end_interval_time,'hh24'),'mm-dd hh24'), instance_number
order by to_char(round(end_interval_time,'hh24'),'mm-dd hh24'), instance_number
/
```

APPENDIX H: TOTAL PGA IN USE

```
select to_char(round(s.end_interval_time,'hh24'),'mm-dd hh24') snap_time
       , g.instance_number
       , g.value/1048576 mbytes
  from dba_hist_snapshot s
       , dba_hist_pgastat g
 where s.snap_id = g.snap_id
     and s.instance_number = g.instance_number
     and s.dbid = g.dbid
     and g.name = 'total PGA allocated'
     and s.end_interval_time > to_date(:start_date,'MMDDYYYY')
     and s.end_interval_time < to_date(:end_date,'MMDDYYYY')
 order by to_char(s.end_interval_time,'mm-dd hh24:mi')
        , g.instance_number
/
```

Hisorical Analysis of Performance Using AWR
April 2009
Authors: Kurt Engeleiter, Cecilia Gervasio Grant
Contributing Authors:

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com