



# Using the Self-tuning Features of Oracle



By Don Burleson

**T**he dynamic memory allocation features that were first introduced in Oracle9i make it possible to create a self-tuning database. This article investigates how STATSPACK extension table can be used to monitor the Oracle database server and adjust the memory regions for *sort\_area\_size*, *large\_pool\_size*, *pga\_aggregate\_target*, *sga\_max\_size* and *db\_cache\_size* according to the processing demands on the server and within the database. We will also look at how STATSPACK can monitor the usage within the memory regions, and how an intelligent mechanism can be created to automatically reconfigure Oracle according to its current processing needs.



In Oracle Database 10g we see important self-tuning features such as Oracle's new Automatic Memory Management. The AMM tool detects shortages and adjusts the main memory regions (*shared\_pool*, *db\_cache\_size*, *pga\_aggregate\_target*) as it notices changing demands on the Oracle environment.

## Moving Toward a Self-tuning Oracle Database

With these new dynamic SGA features in Oracle, we are moving toward an architecture where the Oracle DBA can monitor UNIX RAM memory usage and reconfigure the SGA and PGA regions according to existing usage patterns.

Oracle offers a degree of self-tuning capability with the new *pga\_aggregate\_target* parameter. By allowing Oracle to manage RAM memory demands according to the demands of each task, Oracle has been able to use sophisticated algorithms to improve the speed of RAM intensive tasks such as hash joins and large sorts.

However, the Oracle DBA is now able to dynamically de-allocate RAM memory from one area and re-allocate the RAM to another area of the SGA.

## Changing RAM Configuration with UNIX Scripts

In a UNIX environment it is very easy to schedule a task to change the RAM memory configuration when the processing needs change. For example, many Oracle databases operate in OLTP mode during normal work hours, while at night the database services memory-intensive batch reports.

As we have noted, an OLTP database should have a large value for *db\_cache\_size* while memory-intensive batch tasks require additional RAM in the *pga\_aggregate\_target*.

The UNIX scripts below can be used to reconfigure the SGA between OLTP and DSS without stopping the instance. In this example, we assume that we have an isolated Oracle server with eight gigabytes of RAM. We also assume that we reserve 20% of RAM for UNIX overhead, leaving a total of six gigabytes for Oracle and Oracle connections. These scripts are for HP/UX or Solaris, and accept the \$ORACLE\_SID as an argument.

The *dss\_config.ksh* script will be run at 6:00 each evening to reconfigure Oracle for the memory-intensive batch tasks that run each night.

```
dss_config.ksh

#!/bin/ksh

# First, we must set the environment . . . .
ORACLE_SID=$1
export ORACLE_SID
ORACLE_HOME=`cat /etc/oratab|grep ^$ORACLE_SID:|cut -f2 -d':'`
#ORACLE_HOME=`cat /var/opt/Oracle/oratab|grep ^$ORACLE_SID:|cut -f2 -d':'`
export ORACLE_HOME
PATH=$ORACLE_HOME/bin:$PATH
export PATH

$ORACLE_HOME/bin/sqlplus -s /nologin<<|
connect system/manager as sysdba;
alter system set db_cache_size=1500m;
alter system set shared_pool_size=500m;
alter system set pga_aggregate_target=4000m;
exit
!
```

The *oltp\_config.ksh* script will be run at 6:00 each morning to reconfigure Oracle for the OLTP usage during the day.

```
oltp_config.ksh

#!/bin/ksh

# First, we must set the environment . . . .
ORACLE_SID=$1
export ORACLE_SID
ORACLE_HOME=`cat /etc/oratab|grep ^$ORACLE_SID:|cut -f2 -d':'`
#ORACLE_HOME=`cat /var/opt/Oracle/oratab|grep ^$ORACLE_SID:|cut -f2 -d':'`
export ORACLE_HOME
PATH=$ORACLE_HOME/bin:$PATH
export PATH

$ORACLE_HOME/bin/sqlplus -s /nologin<<|
connect system/manager as sysdba;
alter system set db_cache_size=4000m;
alter system set shared_pool_size=500m;
alter system set pga_aggregate_target=1500m;
exit
!
```

Note: You can also use the `dbms_job` or `dbms_scheduler` packages to schedule these types of reconfiguration events.

Now that we have seen a generic way to change the Oracle configuration, it follows that we can develop a mechanism to constantly monitor the processing demands on Oracle and issue the alter system commands according to existing database demands.

### Trend Identification Within ORACLE Database 10g AWR

We are ready to move on to look at trend identification with the AWR views. The Oracle professional knows that aggregating important Oracle performance metrics over time (day-of-the-week and hour-of-the-day) allows them to see hidden “signatures,” trends that can be used to fix a repeating Oracle problem before it cripples the database. These signatures are extremely important for proactive tuning because they show regularly occurring changes in processing demands. This knowledge allows the DBA to anticipate upcoming changes and reconfigure Oracle just-in-time to meet the changes.

Let's take a simple example. Here is a report called `rpt_sysstat_hr_10g.sql` that will show the “signature” for any Oracle system statistic, averaged by hour of the day.

```
rpt_sysstat_hr_10g.sql

prompt
prompt This will query the dba_hist_sysstat view to
prompt display average values by hour of the day
prompt

set pages 999
break on snap_time skip 2
accept stat_name char prompt 'Enter Statistics Name: ';

col snap_time format a19
col avg_value format 999,999,999

select
  to_char(begin_interval_time,'hh24') snap_time,
  avg(value) avg_value
from
  dba_hist_sysstat
  natural join
  dba_hist_snapshot
where
  stat_name = '&stat_name'
group by
  to_char(begin_interval_time,'hh24')
order by
  to_char(begin_interval_time,'hh24')
;
```

In the output we see an average for every hour of the day. This information can then be easily pasted into an MS-Excel spreadsheet and plotted with the chart wizard or the WISE tool:

```
SQL> @rpt_sysstat_hr

This will query the dba_hist_sysstat view to
display average values by hour of the day

Enter Statistics Name: physical reads

SNAP_TIME          AVG_VALUE
-----
00                 120,861
01                 132,492
02                 134,136
03                 137,460
04                 138,944
05                 140,496
06                 141,937
07                 143,191
08                 145,313
09                 135,881
10                 137,031
11                 138,331
12                 139,388
13                 140,753
14                 128,621
15                 101,683
16                 116,985
17                 118,386
18                 119,463
19                 120,868
20                 121,976
21                 112,906
22                 114,708
23                 116,340
```

Here is the data after pasting into an MS Excel spreadsheet and plotting it with the Excel chart wizard (Figure 1):

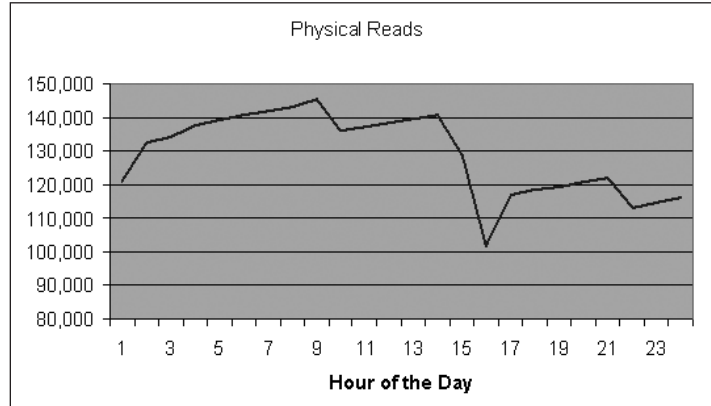


Figure 1

For details on the procedure for plotting Oracle performance data, see the OTN article title “Perfect Pitch.” You can also use open source products such as RRDtool and WISE to automate the plotting of data from the AWR and ASH.

The Oracle Workload Interface Statistical Engine (WISE) is a great way to quickly plot Oracle time series data and gather signatures for Oracle metrics. It is also able to plot performance data on daily or monthly average basis. (See [www.wise-Oracle.com](http://www.wise-Oracle.com) for details).

*continued on page 34*

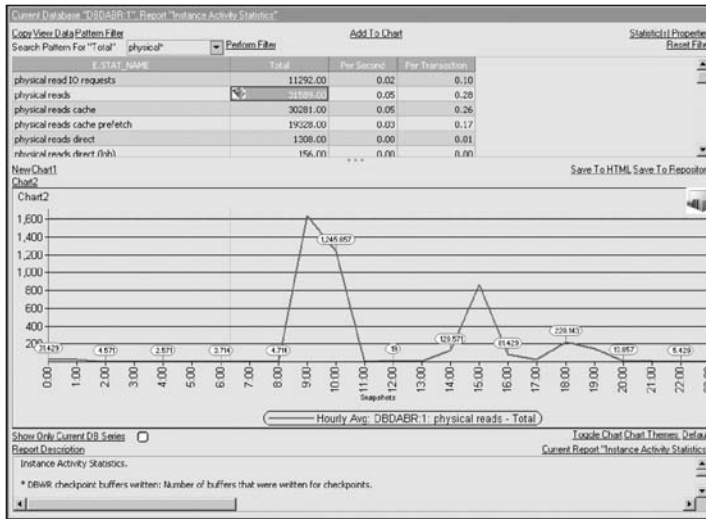


Figure 2

We can create the same types of reports aggregated by day-of-the week to see ongoing daily trends. Over long periods of time, almost all Oracle databases will develop distinct signatures that reflect the regular daily processing patterns of the end-user community.

Here we have *rpt\_sysstat\_dy\_10g.sql* that will accept any of the values from *dba\_hist\_sysstat* and plot the average values by hour-of-the-day.

```
rpt_sysstat_dy_10g.sql

prompt
prompt This will query the dba_hist_sysstat view to display
prompt average values by day-of-the-week
prompt

set pages 999

accept stat_name char prompt 'Enter Statistic Name: ';

col snap_time format a19
col avg_value format 999,999,999

select
  to_char(begin_interval_time,'day') snap_time,
  avg(value) avg_value
from
  dba_hist_sysstat
natural join
  dba_hist_snapshot
where
  stat_name = '&stat_name'
group by
  to_char(begin_interval_time,'day')
order by
  decode(
    to_char(begin_interval_time,'day'),
    'sunday',1,
    'monday',2,
    'tuesday',3,
    'wednesday',4,
    'thursday',5,
    'friday',6,
    'saturday',7
  );
```

Here we see the output: from this script:

```
SQL> @rpt_sysstat_dy
```

This will query the *dba\_hist\_sysstat* view to display average values by day-of-the-week

Enter Statistics Name: physical reads

SNAP_TIME	AVG_VALUE
sunday	190,185
monday	135,749
tuesday	83,313
wednesday	139,627
thursday	105,815
friday	107,250
saturday	154,279

Here we see an average for every day of the week. These types of signatures will typically stabilize for most Oracle databases and they can be used as inputs for proactive tuning activities. Now let's explore correlation analysis using AWR data.

### Correlation Analysis Reports with the AWR and ASH Views

Oracle Database 10g Oracle Wait Interface (OWI) automatically collects interesting statistics that relate to system-wide wait events from *dba\_hist\_waitstat* with detailed wait event information from *dba\_hist\_active\_sess\_history*.

In we want to perform advanced correlation analysis we can seek to identify correlations between instance-wide wait events and block-level waits. This is a critical way of using human insight and using the AWR and ASH information to isolate the exact file and object where the wait contention is occurring.

As a reminder, the ASH WRH\$ tables store the history of a recent session's activity and exposes it via the *dba\_hist\_active\_sess\_history* view. This data is designed as a rolling buffer in memory, and earlier information is overwritten when needed. To do this, we need the *dba\_hist\_active\_sess\_history* view which contains historical block-level contention statistics.

In the script *wait\_time\_detail\_10g.sql* we see that it that compares the wait event values from *dba\_hist\_waitstat* and *dba\_hist\_active\_sess\_history*. This comparison allows us to identify the exact objects that are experiencing wait events.

```
wait_time_detail_10g.sql

prompt
prompt This will compare values from dba_hist_waitstat with
prompt detail information from dba_hist_active_sess_history.
prompt

set pages 999
set lines 80

break on snap_time skip 2

col snap_time heading 'Snap|Time' format a20
col file_name heading 'File|Name' format a40
col object_type heading 'Object|Type' format a10
col object_name heading 'Object|Name' format a20
col wait_count heading 'Wait|Count' format 999,999
col time heading 'Time' format 999,999

select
  to_char(begin_interval_time,'yyy-mm-dd hh24:mi') snap_time,
  -- file_name,
```

```

object_type,
object_name,
wait_count,
time
from
  dba_hist_waitstat      wait,
  dba_hist_snapshot     snap,
  dba_hist_active_sess_history ash,
  dba_data_files        df,
  dba_objects           obj
where
  wait.snap_id = snap.snap_id
and
  wait.snap_id = ash.snap_id
and
  df.file_id = ash.current_file#
and
  obj.object_id = ash.current_obj#
and
  wait_count > 50
order by
  to_char(begin_interval_time,'yyyy-mm-dd hh24:mi'),
  file_name
;

```

Note that this script is also enabled to join into the `dba_data_files` view if we want to get the file names associated with the wait event. This is a very powerful script that can be used to quickly drill-in to find the cause of specific waits. Below we see a sample output from `wait_time_detail`:

```

SQL> @wait_time_detail

This will compare values from dba_hist_waitstat with
detail information from dba_hist_active_sess_hist.

```

Snap Time	Object Type	Object Name	Wait Count	Time
2004-02-28 01:00	TABLE	ORDOR	4,273	67
	INDEX	PK_CUST_ID	12,373	324
	INDEX	FK_CUST_NAME	3,883	17
	INDEX	PK_ITEM_ID	1,256	967
2004-02-29 03:00	TABLE	ITEM_DETAIL	83	69
2004-03-01 04:00	TABLE	ITEM_DETAIL	1,246	45
2004-03-01 21:00	TABLE	CUSTOMER_DET	4,381	354
	TABLE	IND_PART	117	15
2004-03-04 01:00	TABLE	MARVIN	41,273	16
	TABLE	FACTOTUM	2,827	43
	TABLE	DOW_KNOB	853	6
	TABLE	ITEM_DETAIL	57	331
	TABLE	HIST_ORD	4,337	176
	TABLE	TAB_HIST	127	66

This simple example should demonstrate how you can use the AWR and ASH data to create an almost infinite number of sophisticated custom performance reports.

Using the wealth of metrics within the AWR can be greatly useful for you to get detailed correlation information between any of the 500+ performance metrics captured by the AWR.

The AWR repository can also be used for Oracle Data Mining. If you use the ORACLE Database 10g Data Mining (ODM) option, you can automatically scan the AWR seeking statistically significant correlations between metrics using multivariate Chi-Square techniques to reveal hidden patterns within the performance information. The ORACLE Database 10g ODM uses sophisticated Support Vector Machines (SVM) algorithms for binary, multi-class classification models and has built-in linear regression functionality.

So far, AWR and ASH are the most exciting performance optimization tools in Oracle's history, and they can provide the foundation for the use of artificial intelligence techniques to be applied to Oracle performance monitoring and optimization. As Oracle evolves we expect that the AWR and ASH will largely automate the tedious and time-consuming task of Oracle tuning.

## Approaches to Self-tuning Oracle Databases

Until Oracle evolves into a complete self-tuning architecture, the Oracle DBA is responsible for adjusting the RAM memory configuration according to the types of connections. In general, we can use queries against the `v$` structures and STATSPACK or AWR data to locate those times when Oracle connections change their processing characteristics. We see three types of approaches to automated tuning:

- **Normal scheduled reconfiguration.** A bi-modal instance that performs OLTP during regular hours and DSS during off-hours will benefit from a scheduled task to reconfigure the SGA and PGA.
- **Trend-based dynamic reconfiguration.** You can use STATSPACK or AWR data to predict those times when the processing characteristics change and use the `dbms_job` or the ORACLE Database 10g `dbms_scheduler` package to fire ad-hoc SGA and PGA changes.
- **Dynamic reconfiguration.** Just as Oracle dynamically redistributes RAM memory for tasks within the `pga_aggregate_target` region, the Oracle DBA can write scripts that steal RAM from an underutilized area and re-allocate these RAM pages to another RAM area.

## Rules for Changing Memory Sizes

There are three conditions that affect the decision to resize the Oracle RAM regions, one for the data buffer cache, another for the shared pool and the third for PGA memory usage.

- **db\_cache\_size.** We may want to add RAM to the data buffer cache when the physical I/O requirement exceeds a pre-defined threshold.
- **shared\_pool\_size.** A high value for any of the library cache miss ratios may signal the need to allocate more memory to the shared pool.
- **pga\_aggregate\_target.** When we see high values for multi-pass executions, we may want to increase the available PGA memory.

## When to Trigger a Dynamic Reconfiguration

When your scripts detect a condition where a RAM memory region is overstressed, you are faced with a choice about which region will shrink to provide the RAM for the over-stressed area.

In practice, the choice of which area to reduce in size is a choice between the shared pool and the PGA aggregate memory. This is because the shared pool is almost always a small region when compared to the regions for the data buffers and PGA session memory.

continued on page 36

## Conclusion

Oracle automatic tuning has several dimensions, but it provided the Oracle DBA with an unparalleled ability to control the region size for almost every component of the SGA and PGA. As Oracle evolves, mechanisms will be developed to allow Oracle to automatically reconfigure RAM memory based upon processing demands. At the present rate, future releases of Oracle may have true artificial intelligence built-in to detect and correct even the most challenging Oracle optimization issues.

## References

*Oracle Tuning: Oracle Time-series Optimization with the Automatic Workload Repository*, Alexey B. Danchenkov and Donald K. Burleson (Rampant TechPress)

*Oracle9i High-Performance Tuning with STATSPACK*, Donald K. Burleson (Oracle Press)

*Oracle High-Performance Tuning with STATSPACK*, Donald K. Burleson (Oracle Press)

Introduction to Oracle Tuning Webcast (DBAZine):  
[www.dbazine.com/burlesonwebinar1.html](http://www.dbazine.com/burlesonwebinar1.html)

*Creating a Self-Tuning Oracle Database - Automating Oracle9i Dynamic SGA Performance*, Donald K. Burleson (Rampant TechPress)

Oracle Database 10g documentation library:  
[http://otn.oracle.com/pls/db10g/portal.portal\\_demo3?selected=1](http://otn.oracle.com/pls/db10g/portal.portal_demo3?selected=1)

Analyzing Oracle performance data on OTN:  
[http://otn.oracle.com/oramag/webcolumns/2003/techarticles/burleson\\_wait.html](http://otn.oracle.com/oramag/webcolumns/2003/techarticles/burleson_wait.html)

Oracle Database 10g reference manual:  
[http://download-west.oracle.com/docs/cd/B12037\\_01/server.101/b10755/toc.htm](http://download-west.oracle.com/docs/cd/B12037_01/server.101/b10755/toc.htm)

Using the RRDtool:  
<http://www.rrdtool.com>

*Oracle Database 10g New DBA Features*  
(Rampant TechPress)

Oracle Data Mining (ODM):  
<http://otn.oracle.com/products/bi/odm/odmining.html>

Solid-state RAM disk with Oracle:  
[www.storage-search.com/texasmemsysart1.pdf](http://www.storage-search.com/texasmemsysart1.pdf)

WISE – Workload Interface Statistical Engine  
[www.wise-oracle.com](http://www.wise-oracle.com)



### About the Author

**Donald K. Burleson** is considered one of the world's top Oracle Database experts with more than 20 years of full-time DBA experience. He specializes in creating database architectures for very large online databases and he has worked with some of the world's most powerful and complex systems.

A former adjunct professor emeritus, Burleson has written more than 30 books, published more than 100 articles in national magazines, and serves as series editor for Rampant TechPress.