Self Tuning of Oracle Database Using SQL Scripts

Pooja Rani[1] , Pariniyojit Kumar Singh[2], Hitesh Kumar Sharma[3]

[1] Department of Information Technology, ITM University, Gurgaon, Haryana, India

Abstract

The Oracle professionals or DBA's should know how to accurately predict the high-water mark of RAM demands for their database, and fully allocate the RAM, reserving only enough to accommodate spikes in user connections. For an Oracle server, in this paper the goal is to keep all the RAM memory demands of the database and database connections beneath the amount of physical RAM memory. In an Oracle environment, we can accurately control the amount of RAM memory that is used by the database instance System Global Area (SGA). This is because the Oracle database administrator can issue *alter system command* to change the RAM memory areas, and can grow and shrink the RAM memory areas on as needed basis.

Keywords: *SGA, PGA, DBWn, LGWR, CKPT, SMON*

1. Introduction

An Oracle database is a combination of oracle Instance and data files on the file system. Oracle Database = Oracle Instance + Datafiles Again Oracle Instance is nothing but Memory architecture and Background processes. Oracle database uses memory for its operation. The total memory allocated to the Oracle database can be broadly categorized into SGA (System Global Area) and PGA (Program Global Area).

We can also categorize SGA into fixed SGA and variable SGA. Fixed SGA is a component of the SGA that varies in size from platform to platform and release to release. It is compiled into the database. The fixed SGA contains a set of variables that point to the other components of the SGA and variables that contain the values of various parameters. The size of the fixed SGA is something over which we have no control and it is generally very small. Think of this area as a bootstrap section of the SGA, something Oracle uses internally to find the other bits and pieces of the SGA.[1]"

1.1 System Global Area

SGA Contains following data structure:

- Database buffer cache
- Redo log buffer
- Shared pool
- Java pool
- Large pool (optional)
- Data dictionary cache
- Other miscellaneous information

Variable SGA contains 4 main components as listed above, those are "Database Buffer Cache", "Redo Log Buffer", "Shared Pool" and "Large Pool". We call it variable SGA because we can alter the size of each of these components manually using ALTER SYSTEM command. The size of each of the components of variable SGA is determined by INIT.ORA parameters.

Following are the INIT.ORA parameter for each of the component:

• Database Buffer Cache – **db_block_buffers**
This is used to hold the data into the memory. When ever a user access the data, it gets fetched into database buffer cache and it will be managed according to LRU (Least recently used) algorithm.
• Redo Log Buffer – **log_buffer**
This memory block hold the data which is going to be written to redo log file.

• Shared Pool – **shared_pool_size**
This contains 2 memory section, 1) Library Cache 2) Dictionary Cache.

• Large Pool – **Large_pool_size**
If defined then used for heavy operations such as bulk copy during backup or during restore operation.

The total size of SGA is determined by a p arameter SGA_MAX_SIZE. Below is the simple calculation of memory sizes.

1.2 Program Global Area

PGA contains information about bind variables, sort areas, and other aspect of cursor handling. This is not a shared area and every user has its own PGA. But why PGA is required for every user? The reason being that even though the parse information for SQL or PLSQL may be available in library cache of shared pool, the value upon which the user want to execute the select or update statement cannot be shared. These values are stored in PGA. This is also called Private Global Area.Database buffer cache is again divided into 3 different types of cache.

1. Default Cache
2. Keep Cache
3. Recycle Cache

If we define the cache size using DB_CACHE_SIZE (or DB_BLOCK_BUFFER and specify the block size) then this will be **default cache.** The cache has a limited size, so not all the data on disk can fit in the cache. When the cache is full, subsequent cache misses cause Oracle to write dirty data already in the cache to disk to make room for the new data

2.3 Shared Pool Reserved Size

Shared Pool, as we have seen previously contains the parsed SQL statements and execution plans. With continuous use of database, after a p eriod of time the shared pool will get fragmented. New parsed SQL and execution plans comes and old one gets aged out and hence overwritten. This will also lead to larger packages being aged out with new entries going into shared pool. Hence access to such larger packages will take time to parse and create execution plan. This might cause performance issues. To avoid such situation, you can define a p arameter SHARED_POOL_RESERVED_SIZE. This will reserve some additional space other then shared_pool_size[2].

2. Process Architecture

Oracle has several process running in the background for proper functioning of database. Following are the main categories of process.

I. Server Process – to handle the requests of user processes connected to the instance. Server processes (or the server portion of combined user/server processes) created on behalf of each user's application can perform one or more of the following:

- Parse and execute SQL statements issued through the application
- Read necessary data blocks from datafiles on disk into the shared database buffers of the SGA, if the blocks are not already present in the SGA
- Return results in such a way that the application can process the information

II. Background Process - An Oracle instance can have many background processes; not all are always present. The background processes in an Oracle instance include the following: On many operating systems, background processes are created automatically when an instance is started.

Database writer (DBWn) - The **database writer process (DBWn)** writes the contents of buffers to datafiles. The DBW*n* processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk. Although one database writer process (DBW0) is adequate for most systems, you can configure additional processes (DBW1 through DBW9) to improve write performance if your system modifies data heavily. These additional DBW*n* processes are not useful on uniprocessor systems.

Log Writer (LGWR) – The **log writer process (LGWR)** is responsible for redo log buffer management–writing the redo log buffer to a redo log file on disk. LGWR writes all redo entries that have been copied into the buffer since the last time it wrote.

Checkpoint (CKPT) - When a **checkpoint** occurs, Oracle must update the headers of all datafiles to record the details of the checkpoint. This is done by the CKPT process. The CKPT process does not write blocks to disk; DBW*n* always performs that work.

System Monitor (SMON) – The **system monitor process (SMON)** performs crash recovery, if necessary, at instance startup. SMON is also responsible for cleaning up temporary segments that are no longer in use and for coalescing contiguous free extents within dictionary-managed tablespaces. If any dead transactions were skipped during crash and instance recovery because of file-read or offline errors, SMON recovers them when the tablespace or file is brought back online. SMON wakes up regularly to check whether it is needed.

Process Monitor (PMON) -The **process monitor (PMON)** performs process recovery when a user process fails. PMON is responsible for cleaning up the database buffer cache and freeing resources that the user process was using. For example, it resets the status of the active transaction table, releases locks, and removes the process ID from the list of active processes.

__Archiver Process (ARCn)__ -The **archiver process (ARCn)** copies online redo log files to a designated storage device after a log switch has occurred. ARC*n* processes are present only when the database is in ARCHIVELOG mode, and automatic archiving is enabled.An Oracle instance can have up to 10 ARC*n* processes (ARC0 to ARC9). The LGWR process starts a new ARC*n* process whenever the current number of ARC*n* processes is insufficient to handle the workload. The ALERT file keeps a record of when LGWR starts a new ARC*n* process.

__Lock Manager Server (LMS)__ - In Oracle9*i* Real Application Clusters, a Lock Manager Server process (LMS) provides inter-instance resource management.

__Queue Monitor (QMN*n*)__ – The **queue monitor process** is an optional background process for Oracle Advanced Queuing, which monitors the message queues. You can configure up to 10 queue monitor processes. These processes, like the Jnnn processes, are different from other Oracle background processes in that process failure does not cause the instance to fail.[3]

3. Implementation of Proposed System

The primary goal for an Oracle server is to keep the RAM memory demands of the database and the database connection below the amount of physical Ram memory. We can control the amount of RAM used by the database SGA in an Oracle environment by issuing alter system commands. Let's take a closer look at these new Oracle9i features and scripts, which allow you to see detailed RAM memory usage.

3.1 Algorithm

It defines three variables: ΔRT abbreviates for change in response time, BUFFER_SIZE denotes the current size of buffer, CACHE_SIZE corresponds to the size of cache memory

```
ALGORITHM

1. dbTuner (ESTIMATED_CACHE_SIZE)
2. Begin
3.              Run application, algorithm and process
4.              Calculate the change in response time (ΔRT)
5.              If (ΔRT>0)
                {
                Run Script
                {
                BUFFER_SIZE = BUFFER_SIZE + 1
                  Allocate more RAM and update CACHE_SIZE
       }
     Else IF (ΔRT<0)
     {
             Run Script
             {
                             BUFFER_SIZE = BUFFER_SIZE - 1
                  Reduce RAM and update CACHE_SIZE
       }
     }
 6.  Go To Step 4
 6.  Stop application, algorithm and process
 7.  End
```

4. Oracle 9i Scripts

The goal for an oracle server is to keep the RAM memory demands of the database and database connections below the amount of physical RAM memory. We can control the amount of the RAM used by the database SGA in an oracle environment by issuing alter system commands.

__**SCRIPT 1**__
A quick dictionary query (sc1.sql) against the v$parameter view will yield the correct for each PGA RAM region size.

```
set pages 999;
column pga_size format 999,999,999
select 2048576+a.value+b.value  pga from
v$parameter a, v$parameter b where
a.name='sort_area_size' and
b.name='hash_area_size'
```

The data dictionary query output shows that the Oracle PGA will use 3.6 megabytes of RAM for each connected Oracle session.

PGA
3,621,440
If we now multiply the number of connected users of the PGA demands for each user, we will know exactly how much RAM should be reserved for connected sessions.

__**SCRIPT 2**__
Computing total PGA RAM
This script reads both the sort_area_size and hash_area_size to compute the total PGA region.This script will display a prompt for the high water mark of connected users and then computes the total PGA RAM to reserve for dedicated oracle connections.The MS-Windows PGA session incurs a 2MB overhead in this example.
Sc2.sql

```
---------------------
Compute PGA sizes
set pages 999;
column pga_size format 999,999,999
accept hwm number prompt 'Enter the high-water mark of connected users:'
select &hwm*(2048576+a.value+b.value) pga from v$parameter a, v$parameter b
where a.name='sort_area_size' and b.name='hash_area_size'
```

Running the script,we see that we are prompted for the high water mark.we will assume that the HWM of connected sessions to the oracle database server is 100.Oracle will do the math and display the amount of RAM to reserve for oracle connections.

```
SQL>@Sc2

Enter the high water mark of connected users:100
Old   2: &hwm*(2048576+a.value+b.value)pga_size
New 2:100*(2048576+a.value+b.value) pga_size
PGA
------------------
362,144,000
```

Returning to our example Windows server, we are ready to calculate the optimum SGA size. Multiplying 100 by the amount needed for each PGA region(3.62 MB) and adding the 2 MB PGA overhead, gives us the total PGA size of 364 MB.The maximum size for the SGA is determined by subtracting the total PGA and the OS overhead from the total RAM on the server. Here is a summary:

Total RAM on Windows Server 1250 MB Less:
Total PGA regions for 100 users:
364 MB
RAM reserved for Windows(20%)
250 MB

____
Maximum SGA size
636 MB

This leaves 636 MB of free memory for the SGA. Therefore, the RAM allocated to the data buffers should be adjusted to make the SGA size less than 636 MB, the server will begin to page RAM, impairing the performance of the entire server.

SCRIPT 3

Script that adjusts the RAM caches

```
set heading off
set feedback off
set verify off
accept decrease_pool char   prompt 'Enter cache to
decrease: '
accept increase_pool char   prompt 'Enter cache to
increase: '
accept change_amount number prompt 'Enter amount to
change: '
spool Sc3.sql
select
   'alter system set &decrease_pool =
'||to_char(to_number(value)-&change_amount)||';'
from v$parameter where name =
lower('&decrease_pool');
select
   'alter system set &increase_pool =
'||to_char(to_number(value)+&change_amount)||';'
from v$parameter where name =
lower('&increase_pool');
spool off
set feedback on
@Sc3
```

```
************** OUTPUT *************
SQL> @Sc3
Enter cache to decrease: shared_pool_size
Enter cache to increase: db_cache_size
Enter amount to change: 2048576

alter system set shared_pool_size = 39283072;
System altered.
alter system set db_cache_size = 27825792;
System altered.
```

This script prompts the DBA for the name of the cache and the sizes and issues the proper appropriate alter system commands to adjust the regions.

SCRIPT 4

Script provides us with DBHR

```
column bhr format 9.99
column mydate heading 'yr.  mo dy Hr.'
select
   to_char(snap_time,'yyyy-mm-dd HH24')      mydate,
   new.name  buffer_pool_name,
   (((new.consistent_gets-old.consistent_gets)+
   (new.db_block_gets-old.db_block_gets))-
   (new.physical_reads-old.physical_reads))
   /   ((new.consistent_gets-old.consistent_gets)+
   (new.db_block_gets-old.db_block_gets))    bhr
from
   perfstat.stats$buffer_pool_statistics old,
   perfstat.stats$buffer_pool_statistics new,
   perfstat.stats$snapshot              sn
where
   (((new.consistent_gets-old.consistent_gets)+
   (new.db_block_gets-old.db_block_gets))-
   (new.physical_reads-old.physical_reads))
   /   ((new.consistent_gets-old.consistent_gets)+
   (new.db_block_gets-old.db_block_gets)) < .90
and
   new.name = old.name
and
   new.snap_id = sn.snap_id
and
   old.snap_id = sn.snap_id-1;
```

```
**************** OUTPUT **************
Here is a sample of the output from this script:
SQL> @Sc4
yr.  mo dy   Hr BUFFER_POOL_NAME      BHR

------------- -------------------- -----
2011-02-12  15    DEFAULT                  .94
2011-02-12  15    KEEP                     .98
2011-02-12  15   RECYCLE                   .84
2011-02-12  16   DEFAULT                   .91
```

```
2011-02-12  16   KEEP                        .96
2011-02-12  16   RECYCLE                     .68
```

This script provides us with data buffer hit ratio for each of the buffer pools at one hour intervals. It is important that the KEEP pool always has a 99-100 % DBHR. If this is not the case, data blocks should be to the KEEP pool to make it the same size as the sum of all object data blocks that are assigned to the KEEP pool.

SCRIPT 5

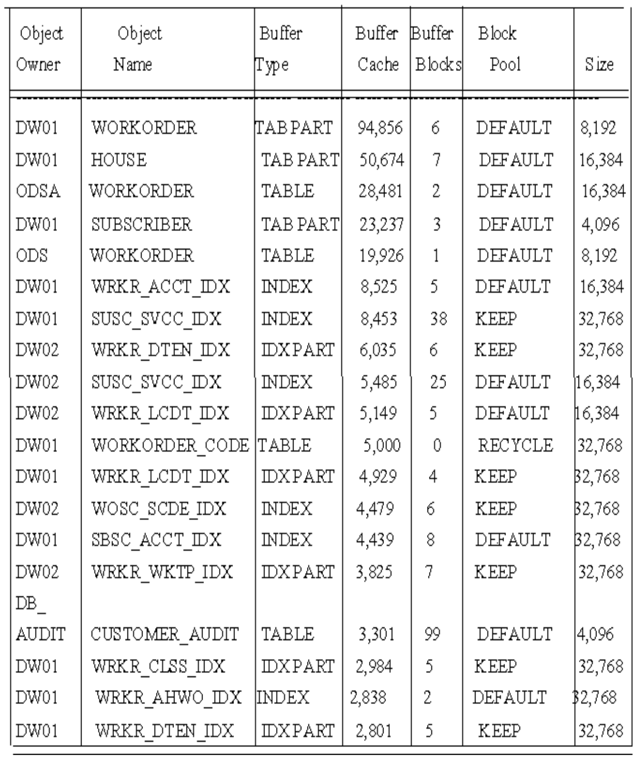
Script provides us with the contents of data buffers

```
set pages 999
set lines 92
ttitle 'Contents of Data Buffers'
drop table t1;
create table t1 as
select
   o.owner          owner,
   o.object_name    object_name,
   o.subobject_name subobject_name,
   o.object_type    object_type,
   count(distinct file# || block#)         num_blocks
from
   dba_objects  o,
   v$bh         bh
where
   o.data_object_id  = bh.objd
and
   o.owner not in ('SYS','SYSTEM')
and
   bh.status != 'free'
group by
   o.owner,
   o.object_name,
   o.subobject_name,
   o.object_type
order by
   count(distinct file# || block#) desc;
column c0 heading "Owner"                format a12
column c1 heading "Object|Name"           format a30
column c2 heading "Object|Type"                  format a8
column c3 heading "Number of|Blocks
in|Buffer|Cache" format 99,999,999
column c4 heading "Percentage|of object|blocks
in|Buffer" format 999
column c5 heading "Buffer|Pool"                 format a7
column c6 heading "Block|Size"                  format
99,999
select
   t1.owner                              c0,
   object_name                              c1,
   case when object_type = 'TABLE PARTITION' then
'TAB PART'
        when object_type = 'INDEX PARTITION' then
'IDX PART'
        else object_type end c2,
   sum(num_blocks)                            c3,
   (sum(num_blocks)/greatest(sum(blocks), .001))*100
c4,
   buffer_pool                        c5,
   sum(bytes)/sum(blocks)                       c6
from
   t1,
   dba_segments s
where
   s.segment_name = t1.object_name
and
   s.owner = t1.owner
and
   s.segment_type = t1.object_type
and
   nvl(s.partition_name,'-') = nvl(t1.subobject_name,'-')
group by
   t1.owner,
   object_name,
   object_type,
   buffer_pool
having
   sum(num_blocks) > 10
order by
   sum(num_blocks) desc;
```

* ******* OUTPUT ******************

5. Conclusion & Future work

Tuning the database can become quite complex, but Oracle9i offers the administrator and unparalleled ability to control the PGA and SGA. Until Oracle9i evolves into a completely self-tuning architecture, the DBA will be responsible for adjusting the dynamic configuration of the system RAM. It is intended to give the DBA a high-level overview of the silent features involved in scheduling dynamic reconfigurations within Oracle. In the future, we may expect complete self-tuning databases to emerge, but in the meantime the administrator must track the historical behavior of the database and apply it to predictive models. It is only in this way that scare instance resources can be proactively applied to develop an optimally-tuned Oracle database. As people get more sophisticated in their self-tuning endeavors, many more Oracle metrics may become self-tuning. For example, there are dozens of self-tuning parameters that are considered immutable that may be found to be changeable. As an example, let's consider the *optimizer_index_cost_adj* parameter.



Object Owner	Object Name	Buffer Type	Buffer Cache	Buffer Blocks	Block Pool	Size
DW01	WORKORDER	TAB PART	94,856	6	DEFAULT	8,192
DW01	HOUSE	TAB PART	50,674	7	DEFAULT	16,384
ODSA	WORKORDER	TABLE	28,481	2	DEFAULT	16,384
DW01	SUBSCRIBER	TAB PART	23,237	3	DEFAULT	4,096
ODS	WORKORDER	TABLE	19,926	1	DEFAULT	8,192
DW01	WRKR_ACCT_IDX	INDEX	8,525	5	DEFAULT	16,384
DW01	SUSC_SVCC_IDX	INDEX	8,453	38	KEEP	32,768
DW02	WRKR_DTEN_IDX	IDXPART	6,035	6	KEEP	32,768
DW02	SUSC_SVCC_IDX	INDEX	5,485	25	DEFAULT	16,384
DW02	WRKR_LCDT_IDX	IDXPART	5,149	5	DEFAULT	16,384
DW01	WORKORDER_CODE	TABLE	5,000	0	RECYCLE	32,768
DW01	WRKR_LCDT_IDX	IDXPART	4,929	4	KEEP	32,768
DW02	WOSC_SCDE_IDX	INDEX	4,479	6	KEEP	32,768
DW01	SBSC_ACCT_IDX	INDEX	4,439	8	DEFAULT	32,768
DW02	WRKR_WKTP_IDX	IDXPART	3,825	7	KEEP	32,768
DB_ AUDIT	CUSTOMER_AUDIT	TABLE	3,301	99	DEFAULT	4,096
DW01	WRKR_CLSS_IDX	IDXPART	2,984	5	KEEP	32,768
DW01	WRKR_AHWO_IDX	INDEX	2,838	2	DEFAULT	32,768
DW01	WRKR_DTEN_IDX	IDXPART	2,801	5	KEEP	32,768

References

[1] loannis Alagiannis (DIAS, I&C, EPFL) ,Towards Adaptive, Flexible, and Self-tuned database system by in EDIC-ru/05.05.2009.

[2] Foundations of Automated Database Tuning by *VLDB '06, September 12– 15, 2006, Seoul, Korea.*Copyright 2006 VLDB Endowment, ACM

[3] Rethinking Database System Architecture: Towards a Self-tuning RISC-style Database System in Cairo, Egypt, 2000

[4] AutoAdmin: Self-Tuning Database Systems Technology, *Copyright 2006 IEEE*

[5] Self-Tuning Database Systems: A Decade of Progress, Copyright 2007 VLDB Endowment

[6] Automatic Physical Database Tuning: A relaxation based Approach, Copyright 2005

[7] SQL Memory Management in Oracle9i,Hong Kong, China, 2002

[8] Self-Tuning for SQL Performance in Oracle Database 11g. *2009 IEEE 25th* International Conference on Data Engineering