ORACLE9I MEMORY MANAGEMENT: EASIER THAN EVER

Sushil Kumar, Oracle Corporation
Sushil.Kumar@oracle.com

INTRODUCTION

Oracle9i has made significant advances in the area of memory management. It is now possible to dynamically resize SGA components and control the PGA memory utilization at the instance level. But that is not all; Oracle9i also provides a set of server-based advisories that help administrators tune various memory components optimally. This presentation takes a closer look at these features, explains how exactly they work, and recommends best practices for using them.

DYNAMIC SGA

OVERVIEW

Since its inception, the System Global Area (SGA) has been a static allocation of memory, shareable across all Oracle threads of execution. The size of the shareable memory is calculated based on the values of initialization parameters and once allocated, the amount of shareable memory used by an Oracle instance cannot grow or shrink. Oracle subsystems assume memory is allocated at startup, and it is not relinquished during the lifetime of the instance. If the database administrator wants to increase the size of the buffer cache, the instance must be restarted with new parameter values.

Oracle9i makes it simple to add or remove memory to or from an Oracle instance by allowing administrators to change the SGA configuration without shutting the instance down. Dynamic SGA allows administrators to use the ALTER SYSTEM command to

- Grow the size of SGA components (Buffer Cache, Shared Pool, Large Pool).
- Shrink SGA by reducing the size of SGA components to an Oracle prescribed minimum.

Oracle9i also includes advisories to help administrators determine the optimal size for the buffer cache and the shared pool. These advisories can be used to determine whether the SGA needs to be shrunk or grown for the present workload.

There are numerous advantages of dynamic SGA. For instance, it allows the buffer cache to relinquish memory to other SGA components (such as shared and large pool) if the memory requirements of these components increase. Conversely, it also allows the size of buffer cache to increase at the expense of the shared pool, if the buffer cache hit ratio is low. It also makes it possible to accommodate changes in the memory available to database instance resulting either from changes to system hardware or changes to the OS resource manager enforced resource allocation.

CONCEPTS

NEW INITIALIZATION PARAMETERS

Oracle9i introduces a set of new initialization parameters for sizing the buffer cache. These parameters are more intuitive and user friendly. The new parameters are:

New Parameter	Unit of Specification	Deprecated Parameter
DB_CACHE_SIZE	KB, MB, GB	DB_BLOCK_BUFFERS

New Parameter	Unit of Specification	Deprecated Parameter
DB_KEEP_SIZE	KB, MB, GB	BUFFER_POOL_KEEP
DB_RECYCLE_SIZE	KB, MB, GB	BUFFER_POOL_RECYCLE

As may be noted above, the new parameters are specified in terms of amount of memory, and not in the unit of number of blocks as was the case with the deprecated parameters. The other major difference is that the KEEP and RECYCLE pools are allocated independent of the default buffer cache. Therefore unlike DB_BLOCK_BUFFERS, DB_CACHE_SIZE does not subsume the KEEP (DB_KEEP_SIZE) and RECYCLE (DB_RECYCLE_SIZE) buffer pools.

In addition, another set of new parameters has been introduced in Oracle9i to support the multiple block size feature. Oracle9i allows creation of tablespaces with different block sizes. However before creating a tablespace with a "non-standard" block size (i.e. block size different than that specified by DB_BLOCK_SIZE parameter), the corresponding buffer cache should be created using the parameters listed below.

New Parameter	Unit of Specification	Description
DB_2K_CACHE_SIZE	KB, MB, GB	Buffer Cache for 2K Blocks
DB_4K_CACHE_SIZE	KB, MB, GB	Buffer Cache for 4K Blocks
DB_8K_CACHE_SIZE	KB, MB, GB	Buffer Cache for 8K Blocks
DB_16K_CACHE_SIZE	KB, MB, GB	Buffer Cache for 16K Blocks
DB_32K_CACHE_SIZE	KB, MB, GB	Buffer Cache for 32K Blocks

The DB_nK_CACHE_SIZE parameters can only be used to specify the cache size for non-standard block sizes. Therefore, if the "standard" block, as specified by DB_BLOCK_SIZE parameter, is 8K, it is illegal to set DB_8K_CACHE_SIZE parameter.

In order to be able to resize the buffer cache dynamically, a DBA must use the new parameters. The deprecated parameters continue to be static and hence their values cannot be changed without restarting the instance. Furthermore, these parameters cannot be combined with the new parameters i.e. combining them in the same parameter file will produce an error. If the deprecated parameters are used, a warning message will be written to the alert log to encourage users to migrate to the new parameter scheme.

The SGA_MAX_SIZE parameter has been added to specify the maximum size of the SGA for the lifetime of the instance. A resize operation will succeed as long as it does not seek to increase the total SGA size beyond this value.

UNIT OF ALLOCATION

Prior to Oracle9*i,* the SGA implementation allowed the operating system dependent code to determine the minimum size of an individual allocation for a particular subsystem requiring memory. The buffer cache, the shared and large pools, and the redo buffers were treated as independent logical sections of the SGA with the product lines determining the minimum allocation sizes and other attributes for shared memory.

In the dynamic SGA model, the generic code specifies the unit of allocation for the SGA memory. The new unit of allocation is called a *granule* whose size is platform dependent and is largely determined by the total (maximum) SGA size. On most UNIX platform, including Solaris, if the total SGA size is less than 128 MB, the granules are of 4 MB. Otherwise they are of 16 MB. On NT, the granule size is 4 MB if the value of the SGA_MAX_SIZE parameter is less than 128 MB and 8 MB otherwise. For other platforms, please refer to your OS specific documentation.

The buffer cache, the shared pool, the large pool, and other components allocate and free SGA space in units of granules. Oracle tracks SGA memory use in integral numbers of granules, by SGA component. No two components can divide the space within one granule.

Allocation of SGA Memory at Startup

At startup time, Oracle allocates virtual address space for the SGA based on the value of the SGA_MAX_SIZE parameter, rounded up to the next granule size, and the operating system memory limits.

When the instance is started, Oracle allocates granule entries, one for each granule to support SGA_MAX_SIZE bytes of address space. This allocation is made only for the virtual address space and not for the physical memory. Oracle rounds the value of each initialization parameter, which defines the amount of SGA memory a component allocates e.g. DB_CACHE_SIZE, SHARED_POOL_SIZE, etc., to the nearest granule size multiple. Each component allocates an integral number of granules. The minimum size of the SGA is two granules, one each for the buffer cache and shared pool, plus the memory required for the fixed SGA and other metadata, including redo buffers.

Growing a SGA Component

A database administrator can add memory to a SGA component by issuing the ALTER SYSTEM command to modify the corresponding initialization parameters value. Oracle then rounds up the given target size to the nearest multiple of granule size, and adds or takes away granules to meet the target size.

Adding memory to a SGA component will succeed only if Oracle has enough free granules to satisfy the request. Oracle does not automatically start freeing another component's granules for adding it to a given component. Instead, the database administrator must ensure that the instance has enough free granules to satisfy the growth requirement either by shrinking another component or by reserving enough granules at the time of instance startup by setting the value of the parameter SGA_MAX_SIZE appropriately. If the used SGA memory is less than the SGA_MAX_SIZE, Oracle can allocate more granules until the SGA size reaches this limit. If the value of the SGA_MAX_SIZE parameter is not explicitly specified, it is set equal to the current SGA size by default. Under such circumstances, the size of any SGA component cannot be increased without shrinking another component. The following examples will illustrate these concepts better:

Example 1

Initialization parameter values:

```
SGA_MAX_SIZE = 144M
```

Memory used by other non-variable SGA components : 16 MB

Maximum combined size of the shared pool and buffer cache: 144-16= 128 MB

```
DB_CACHE_SIZE = 96M
SHARED_POOL_SIZE = 32M
```

Now we try to grow the shared pool by 32M

```
SQL> alter system set shared_pool_size=64M;
alter system set shared_pool_size=64M
*
ERROR at line 1:
ORA-02097: parameter cannot be modified because specified value is
invalid
ORA-04033: Insufficient memory to grow pool
```

The operation did not succeed because enough free granules do not exist.

Let us now shrink the buffer cache by 32M:

```
SQL> alter system set db_cache_size=64M;
```

```
System altered.
```

Let us try the failed operation again:

```
SQL> alter system set shared_pool_size=64M;

System altered.
```

Example 2

Initialization parameter values:

```
SGA_MAX_SIZE = 256M
DB_CACHE_SIZE = 96M
SHARED_POOL_SIZE = 32M

SQL> alter system set shared_pool_size=64M;

System altered.
```

In this case, we did not have to shrink the buffer cache since enough free granules were allocated at the time of the instance startup due to larger value of the SGA_MAX_SIZE parameter.

Shrinking a SGA Component

Decreasing the number of granules for a SGA component is a more complicated than increasing them. In order to free memory, a component must perform two actions. First, the component must decide which granules it intends to free and prevent any new allocations of these granules. It should then wait to free the granules until all references to them have been removed. In case of the buffer cache, data contained in the granule to be freed can be written back to disk but for shared pool, Oracle must wait until the sessions/operations using those granules have completed. Once this is done, the component is allowed to free the granule.

A shrink operation, which tries to reduce the size of a SGA component by a large amount, may, therefore, take a long time to complete. This is particularly true for the shared pool since de-allocating memory from shared pool is more complicated than doing so from any other SGA component. Therefore an operation, which is seeking to reduce the shared pool to an extremely small size (e.g. 10M or lower), may never be able to complete resulting in a "hung" session. If such a situation is encountered, the resize operation in progress can be cancelled by interrupting the session using CTRL-C or CTRL-D keystrokes

Concurrent Resize Operations

Only one resize operation can be executed for a given SGA component at a time. Therefore, if another buffer cache resize operation is initiated while one is already in progress, it will fail with an error message. Multiple resize operations for different components can, however, be executed simultaneously i.e. a buffer cache and a shared pool resize operation can be executed concurrently.

Locking of Physical Memory

Some Operating Systems (OS) support the ability to lock the SGA in physical memory in order to ensure that no part of the SGA is ever paged out. The initialization parameter LOCK_SGA can be used to instruct Oracle to pin the SGA in physical memory using port-specific routines. The undocumented initialization parameter _USE_ISM in used on certain OS to share some virtual memory constructs (i.e. page tables) across processes and on versions of Solaris with no dynamic ISM(DISM) support, this causes the physical memory for the SGA to be locked for the duration of the instance. There may be a number other situations where the OS may implicitly lock the SGA depending on the system configuration e.g. the use of asynchronous IO in HP-UX.

Whenever the SGA is locked in the physical memory, Oracle9*i* pre-allocates and pins the amount of memory equal to the SGA_MAX_SIZE specification. In other words, an attempt to lock the SGA in the physical memory will result in the locking of memory equal to the maximum SGA size. While this guarantees the ability to grow the SGA dynamically, it can result in wastage of system memory especially if the current SGA size is significantly smaller than the SGA_MAX_SIZE specification. Therefore, in order to take advantage of true "dynamicity" of the SGA, the use of the LOCK_SGA parameter (if available) is not recommended.

On Solaris, the _USE_ISM parameter is set to TRUE by default for performance reasons. On versions of Solaris with no DISM support, this causes locking of physical memory equal to the SGA_MAX_SIZE specification. Although setting the _USE_ISM parameter to FALSE will change this behavior, doing so is strongly discouraged since it can lead to significant performance degradation. A better approach would be to upgrade the OS to Solaris8 which supports dynamic intimate shared memory (DISM) capability[1]. DISM enables Oracle to lock the amount memory equal to the current SGA size initially and grow the shared memory segment dynamically, whenever needed.

It is important to remind here that if the SGA is locked in the physical memory, either explicitly or implicitly, the memory freed by shrinking an SGA component will not be released to the OS. In simple terms, the overall size of the SGA cannot be reduced if it has been locked in the physical memory. The free memory (i.e. SGA_MAX_SIZE – Current SGA Size) will just be reserved for any future growth operation.

Viewing SGA Statistics

The view V$SGA can be used to find out the distribution of the SGA memory among the various components. The following table describes each of the individual statistics reported in this view:

Statistics Name	Description
Fixed Size	Fixed Overhead
Database Buffer	Size of the Buffer Cache
Redo Buffers	Size of Redo Log Buffer
Variable Size	Includes the Shared Pool, Java Pool, Large Pool and Free Memory

In Oracle9i Release 2, the V$SGA_DYNAMIC_COMPONENTS views the current sizes of dynamic SGA components as well as a summary of all resize operations performed since start up. This view also contains the granule size in use. Another new view introduced in Oracle9*i* Release 2, V$SGA_DYNAMIC_FREE_MEMORY, displays information about the amount of free SGA memory available for future resize operations.

In Oracle9*i* Release 1, the amount of free SGA memory available at any point in time can be derived by subtracting the sum of Shared Pool, Java Pool and Large Pool sizes from the "Variable Size".

The following script generates a detailed report of the SGA configuration:

```
set serverout on
declare
dbname varchar2(15);    -- Database Name
tsgasize number;        -- Total SGA Size
bcsize number;          -- Buffer Cache Size
spsize number;          -- Shared Pool Size
jpsize number;          -- Java Pool Size
lpsize number;          -- Large Pool Size
fsize number;           -- Fixed SGA Size
rbsize number;          -- Redo Buffers
used number;            -- Used SGA Memory
free number;            -- Free SGA Memory
granule_size number;    -- Granule Size
```

[1] DISM is supported on Solaris 8 release date 01/01 or later. Sun also recommends installing the patch 108528-16 while using the dynamic SGA feature.

```
tvsize number;         -- Total Variable Size

cursor c1 is
select name, value from sys.v$parameter where name in ('java_pool_size',
'large_pool_size');

cursor c2 is
select name, value from sys.v$sga;

begin

      select name into dbname from sys.v$database;
      select x.ksppstvl/(1024*1024) into granule_size from sys.x$ksppsv x,
sys.x$ksppi y
            where x.indx=y.indx and y.ksppinm='_ksmg_granule_size';
      for cur1 in c1 loop
            case cur1.name
                  when 'java_pool_size' then jpsize :=cur1.value;
                    when 'large_pool_size' then lpsize :=cur1.value;
            end case;
      end loop;

      for cur2 in c2 loop
            case cur2.name
                   when 'Fixed Size' then fsize := cur2.value;
                   when 'Variable Size' then tvsize := cur2.value;
                   when 'Database Buffers' then bcsize :=cur2.value;
                   when 'Redo Buffers' then rbsize :=cur2.value;
            end case;
      end loop;

      /*Getting Shared Pool Size. Can not use shared_pool_size parameter
value due
        to bug 1673506*/

      select cursiz_kghdsnew*granule_size into spsize
      from sys.x$ksmsp_dsnew;

      tsgasize := (fsize+tvsize+bcsize+rbsize);
      free := (tvsize - ((spsize*1024*1024)+lpsize+jpsize));
      used := tsgasize - free ;


      dbms_output.put_line('                          ');
      dbms_output.new_line;
      dbms_output.put_line('SGA Configuration for '||dbname);
      dbms_output.put_line('-------------------------------');
      dbms_output.put_line('                          ');
      dbms_output.new_line;
      dbms_output.put_line('Current SGA Size                 :
'||round(used/(1024*1024),2)||' MB');
      dbms_output.put_line('Maximum SGA Size                 :
'||round(tsgasize/(1024*1024),2)||' MB');
      dbms_output.put_line('Memory Available for SGA Growth:
'||round(free/(1024*1024),2)||' MB');
      dbms_output.put_line('Buffer Cache Size                : '||
round(bcsize/(1024*1024),2) ||' MB');
      dbms_output.put_line('Shared Pool Size                 : '|| spsize ||'
MB');
      dbms_output.put_line('Large Pool Size                  : '||
round(lpsize/(1024*1024),2) ||' MB');
      dbms_output.put_line('Java Pool Size                   : '||
round(jpsize/(1024*1024),2) ||' MB');
      dbms_output.put_line('Fixed SGA                      : '||
round(fsize/(1024*1024),2) ||' MB');
      dbms_output.put_line('Redo Buffers                   : '||
round(rbsize/(1024*1024),2) ||' MB');
      dbms_output.put_line('Granule Size                   :
'||granule_size||' MB');
end;
/
set serverout off
set doc on
```

The above script must be run as a SYSDBA user. Its output will look some thing like:

```
SGA Configuration for TEST9I
-------------------------------

Current SGA Size               : 165.78 MB
Maximum SGA Size               : 176.78 MB
Memory Available for SGA Growth: 11 MB
Buffer Cache Size              : 64 MB
Shared Pool Size               : 64 MB
Large Pool Size                : 5 MB
Java Pool Size                 : 32 MB
Fixed SGA                      : .27 MB
Redo Buffers                   : .52 MB
Granule Size                   : 16 MB
```

Tracking a Resize Operation in Progress

Oracle9i Release 2

The view V$SGA_CURRENT_RESIZE_OPS displays information about SGA resize operations which are currently in progress.

Oracle9i Release 1

Buffer Cache

The V$BUFFER_POOL contains information about any buffer cache resize operation in progress.

Shared Pool

Oracle9i Release 9.0.1 does not provide any dynamic view to track the progress of a shared pool resize operation. Users with SYSDBA privilege can execute the following query to find out the status of a resize operation:

```
select round((a.cursiz_kghdsnew*b.ksppstvl)/(1024*1024),2)
"Current Shared Pool Size (MB)",
round((a.tarsiz_kghdsnew*b.ksppstvl)/(1024*1024),2)
"Target Shared Pool Size (MB)"
from sys.x$ksmsp_dsnew a, sys.x$ksppsv b, sys.x$ksppi c
where b.indx=c.indx
and c.ksppinm='_ksmg_granule_size'
```

The output of this script will look something like:

```
SQL> /
Current Shared Pool Size (MB) Target Shared Pool Size (MB)
----------------------------- ----------------------------
64                            0
```

The "Target Shared Pool Size" column will display "0" if no shared pool resize operation is currently in progress. Otherwise it will list the value to which the shared pool is being resized to.

Resize Operation "History"

The V$SGA_RESIZE_OPS view contains information about the last 100 completed SGA resize operations, not including those currently in-progress. It is available only in Oracle9i Release 2.

Every resize operation is also recorded in the alert log, both in Oracle9*i* Release 1 and Release 2. The information recorded includes

- The ALTER SYSTEM command executed to resize a SGA component
- Time when the resize operation was initiated
- Current Size
- Target Size
- The outcome of the operation

Following is an excerpt from the alert log file showing the information mentioned above:

```
Wed Jun 13 13:46:20 2001
CKPT: Begin resize of buffer pool 3 (DEFAULT for block size 8192)
CKPT: Current size = 32 MB, Target size = 64 MB
CKPT: Resize completed for buffer pool DEFAULT for blocksize 8192
........
........
........
Wed Jun 13 13:46:58 2001
CKPT: Begin resize of buffer pool 3 (DEFAULT for block size 8192)
CKPT: Current size = 64 MB, Target size = 80 MB
CKPT: Could not allocate memory for
buffer pool DEFAULT, blocksize 8192
```

SGA Sizing Advisories

Buffer Cache

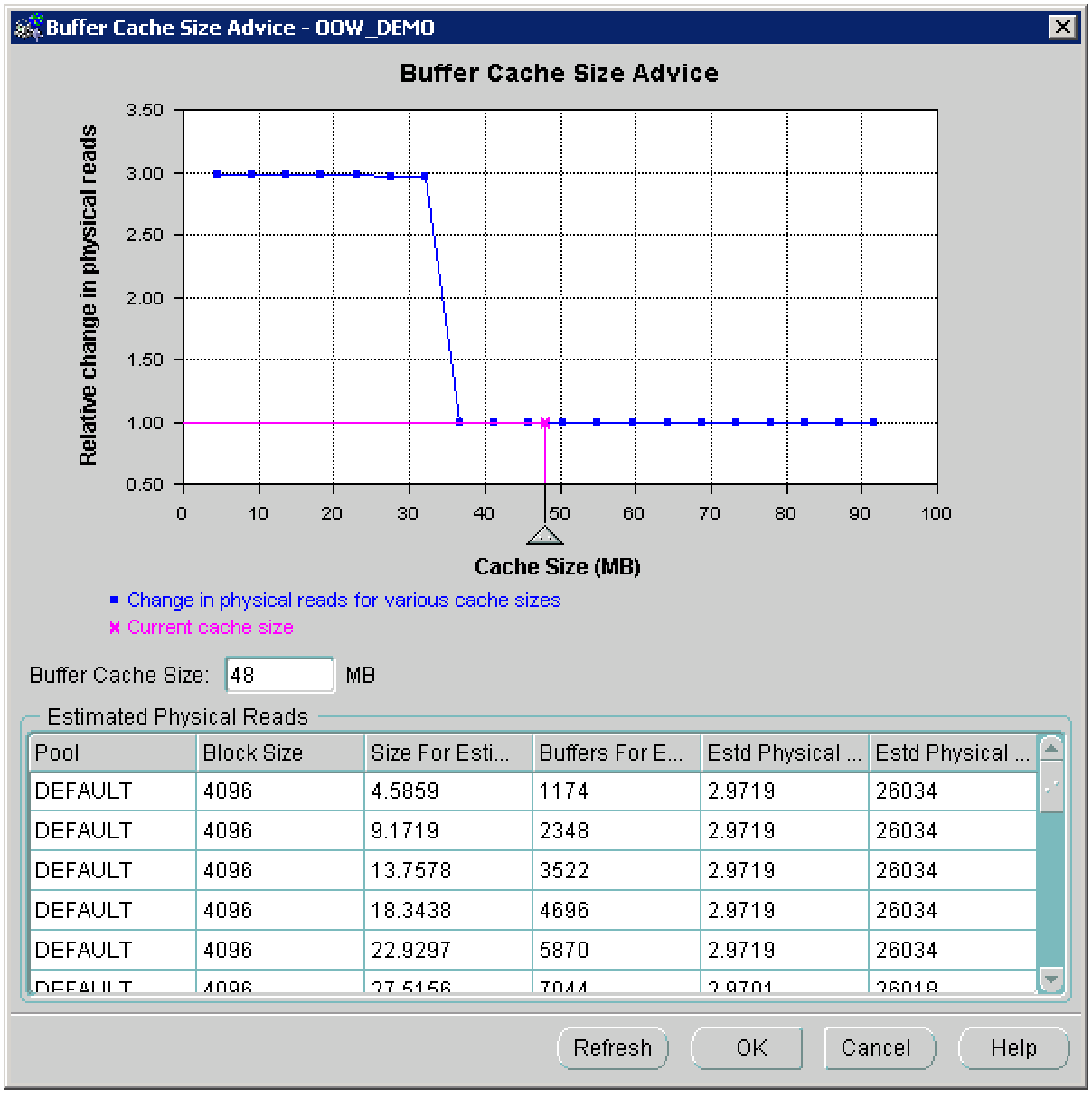
A new initialization parameter DB_CACHE_ADVICE was introduced in Oracle9*i* Release 1 to enable estimation of miss rates for different sizes of the buffer caches. This advisory uses simulation to populate the contents of a new fixed view: V$DB_CACHE_ADVICE. The view contains different rows that predict the number of physical reads for the cache size corresponding to the row. The rows also compute a "physical read factor" which is the ratio of the number of estimated reads to the number of reads actually performed during the measurement interval by the real buffer cache.

In Oracle9*i* Release 1, the buffer cache advisory is disabled by default. It can be enabled using the DB_CACHE_SIZE initialization parameter, which can be set to either OFF, READY or ON. While the values of OFF and ON activate and deactivate the advisory respectively, changing the value of this parameter from ON to READY preserves the data collected so far in the V$DB_CACHE_ADVICE view but stops further simulation. Also, the advisory must be disabled while resizing the buffer cache. Therefore, when an ALTER SYSTEM command is executed to resize the buffer cache, the value of the DB_CACHE_ADVICE parameter is automatically changed to READY, if it is set to ON at the time of initiating the resize operation. In order to continue collection of advisory statistics, it must be manually reactivated.

Oracle9*i* Release 2 introduces a comprehensive set of advisories including shared pool sizing advisor, SQL Execution Memory (PGA) Memory Advisor and Recovery Cost Estimator. All the advisories in Oracle9*i* Release 2 including the Buffer Cache Advisor are controlled by a newly introduced parameter STATISTICS_LEVEL. The Parameter DB_CACHE_ADVICE has, therefore, been deprecated in Oracle9*i* Release 2. By default, the STATISTICS_LEVEL parameter is set to TYPICAL thereby enabling all the advisories.

V$STATISTICS_LEVEL lists the status of the statistics or advisories controlled by the STATISTICS_LEVEL initialization parameter. Each row of V$STATISTICS_LEVEL represents one of these statistics or advisories. For more details on the STATISTICS_LEVEL parameter, please refer to Oracle9*i* Release 2 Performance Tuning Guide and Reference.

In Oracle9*i* Release 1, if the DB_CACHE_SIZE parameter is set to either READY or ON at instance start up, the additional memory required for simulation is separately allocated. However, if it is set to OFF while starting the instance and its value is later changed dynamically to enable advisory, the simulation memory will be borrowed from the shared pool which may cause some performance degradation when either the shared pool is very small or the buffer cache is very large. This can be avoided by setting the DB_CACHE_SIZE to either READY or ON at instance start up. The simulation algorithm for the buffer cache advisory has been revised in Oracle9*i* Release 2 to make it more light weight. The Oracle9*i* Release 2 buffer cache advisory uses sampling technique to minimize its memory requirement and performance overhead. The initial parameter setting, therefore, should not be an issue in Oracle9*i* Release 2.



Buffer Cache Size: 48 MB

Estimated Physical Reads

Pool	Block Size	Size For Esti...	Buffers For E...	Estd Physical ...	Estd Physical ...
DEFAULT	4096	4.5859	1174	2.9719	26034
DEFAULT	4096	9.1719	2348	2.9719	26034
DEFAULT	4096	13.7578	3522	2.9719	26034
DEFAULT	4096	18.3438	4696	2.9719	26034
DEFAULT	4096	22.9297	5870	2.9719	26034
DEFAULT	4096	27.5156	7044	2.9701	26018

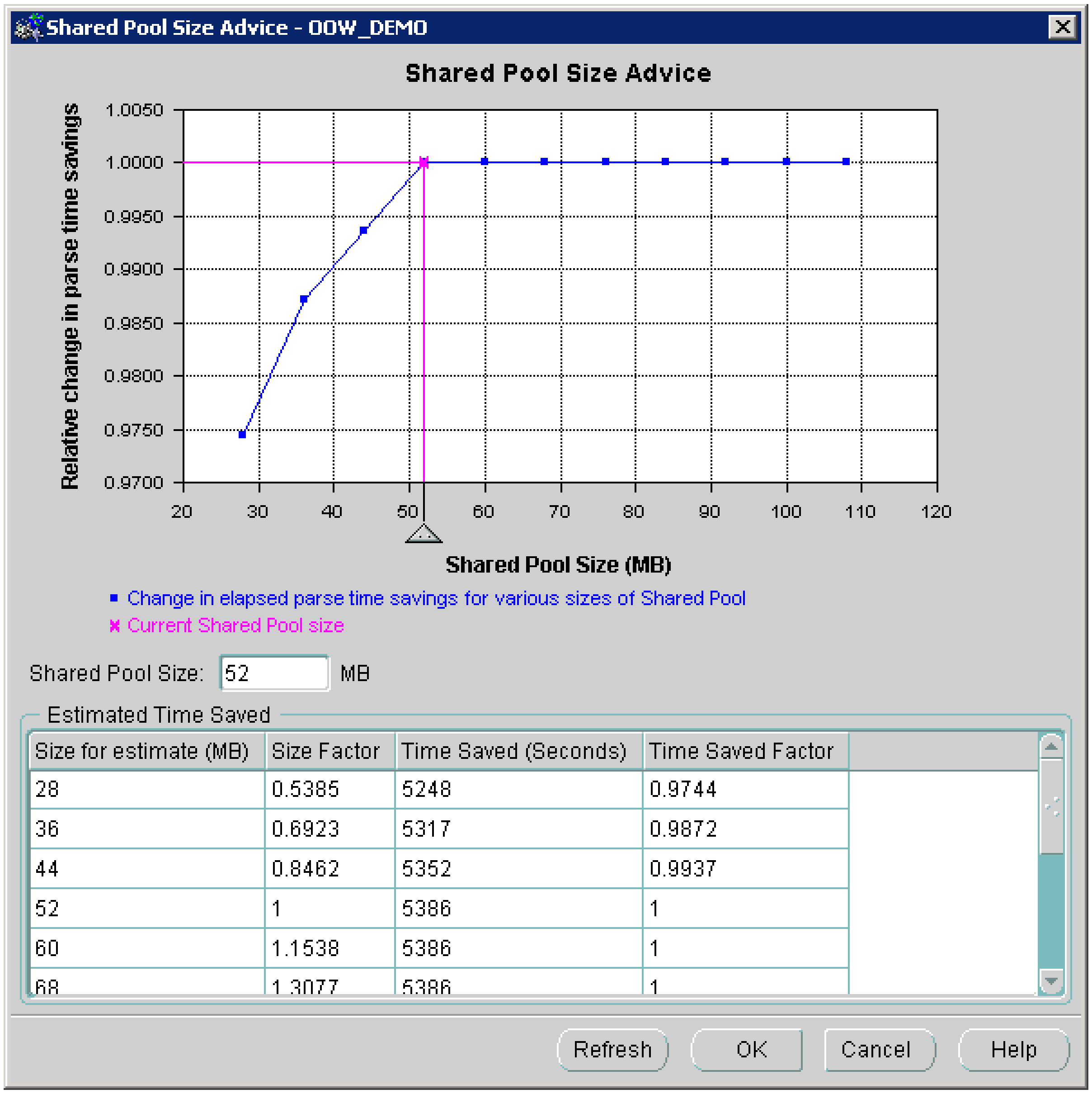
Figure 1: Buffer Cache Advisory (Oracle Enterprise Manager)

SHARED POOL

Oracle9*i* Release 2 introduces the shared pool advisory to provide information about library cache memory and predict the effect of altering the shared pool size on the total amount parsing activities in the system. Two new dynamic views, as described below, provide the information regarding how much memory the library cache is currently using, how much it is pinned, how much is on the shared pool's LRU list, and how the total instance wide parse time will change as a result of changing the shared pool size.

View Name	Description
V$SHARED_POOL_ADVICE	Displays information about estimated parse time savings in different sizes of shared pool. The sizes range from 50% to 200% of current shared pool size, in equal intervals. The value of the interval depends on current shared pool size. Parse time saved refers to the amount of time saved by keeping library cache memory objects in the shared pool, as opposed to having to reload these objects.
V$LIBRARY_CACHE_MEMORY	Contains information about memory allocated to library cache memory objects in different namespaces. A memory object is an internal grouping of memory for efficient management. A library cache object may consist of one or more memory objects.

These views display any data when shared pool advisory is on and reset when the advisory is turned off.


Shared Pool Size Advice - OOW_DEMO
Shared Pool Size Advice
Relative change in parse time savings
1.0050
1.0000
0.9950
0.9900
0.9850
0.9800
0.9750
0.9700
20
30
40
50
60
70
80
90
100
110
120
Shared Pool Size (MB)
■ Change in elapsed parse time savings for various sizes of Shared Pool
✖ Current Shared Pool size
Refresh
OK
Cancel
Help

Shared Pool Size: 52 MB

Estimated Time Saved

Size for estimate (MB)	Size Factor	Time Saved (Seconds)	Time Saved Factor
28	0.5385	5248	0.9744
36	0.6923	5317	0.9872
44	0.8462	5352	0.9937
52	1	5386	1
60	1.1538	5386	1
68	1.3077	5386	1

Figure 2: Shared Pool Advisory (Oracle Enterprise Manager)

Customer Implementation Case Studies

It is obvious that Oracle9*i* makes the SGA tuning process significantly simpler. While the ability to dynamically resize the SGA components help prevent down time, the SGA sizing advisories take the guesswork out of the memory tuning process. It is hardly surprising, therefore, that these features are being widely used by customers to optimize the system resource utilization and enhance availability. Following table lists some of these customers:

Customer Name	Business Description	System/Environment Details
Talkline	Mobile phone & services company in Germany (http://www.talkline.de)	• Sun E-10000 (20 CPUs, 8GB RAM) • 4 x A5200 (72 x 18 GB disks), 1 x A3500 (18 x 9 GB disks), 340 MB/sec. throughput

		• DWH: 1/2 TB raw data, largest table 100 GB
Nordac	A leading IT partner and service provider for medium-sized companies in Germany (http://www.nordac.de)	50 GB 2-node RAC database supporting a Logistics application running on 1 GHz single CPU ES45 Alpha Server (True64) with 4GB RAM
Bauer Verlagsgruppe	A major German publishing group (http://www.hbv.de)	3 GB 2-node (Two IBM Netfinity 5800R systems with 4 CPUS and 4GB RAM each) RAC OLTP database serving up to 250 users
Postbank	A Germany based international banking group (http://www.postbank.de)	Oracle9i Release 2 RAC running on HP-UX

Automatic SQL Execution Memory Management

What Is SQL Execution Memory?

Besides SGA, the Oracle Database also assigns each server process a private memory region called the "Program Global Area" (PGA). A PGA is created for each server process when it is started i.e. when a new session is initiated while using the dedicate server configuration or, when a new shared server process is created. It contains data and control information for a server process and, unlike the SGA, each server process has exclusive access to its PGA.

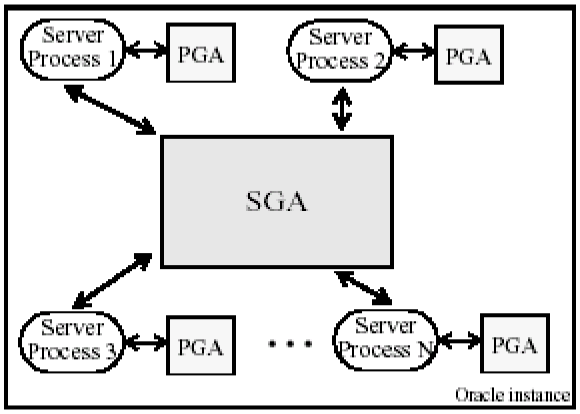


Figure 3: Oracle Database Instance Memory Model

Although the actual content of the PGA varies depending on whether a process is running in dedicated or shared mode, it can generally be categorized as follows:

- Session Memory: The memory allocated to hold session's logon information and other such details. For shared server processes, this information is stored in the SGA.
- SQL Execution Memory: The memory allocated on behalf of SQL statements being executed. The SQL Execution Memory has a persistent and a run time area
  - The persistent area is allocated when a cursor is opened for SQL execution. It contains the information that persist across multiple execution of the same statement (i.e. cursor) such as, bind details, data type conversion, etc. The persistent area is de-allocated when the cursor is closed. For shared server processes, the persistent area is a part of the SGA.

- The runtime area contains information used while a SQL statement is being executed. Its size depends on the type and complexity of SQL statement as well as the number and size of rows being processed. It is de-allocated once the execution completes. For shared sever processes, the run time area is resides in the PGA for DML/DDL operation and in the SGA for queries. For complex queries, such as those used for reporting purposes or as adhoc queries in a data-warehousing environment, a large portion of the run time SQL execution memory is dedicated as the working area for operations such as sort, hash-join, bitmap-merge, etc.

How Does SQL Execution Memory Impact Query Performance?

Generally speaking, larger working area can significantly improve the performance of these operations and thereby, reduce the response time of queries. Ideally, the size of work area should be big enough such that it can accommodate all input data and auxiliary structure needed by the operation. This is referred to as the *cache* size of the work area. When the actual amount of memory available is less than the cache area, the response time increases since only a part of the input data can be accommodated in memory necessitating an extra pass over all or part of the input data. This is referred to as the *one-pass* size of the work area. When enough memory is not available to run an operation even in the one-pass mode, multiple passes over input data are required causing dramatic increase in performance time. For example, a sort operation that needs to sort 10GB of data may need a little more than 10 GB memory to run in the cache mode and about 40 MB to run in the one-pass mode. If the amount of memory available to this operation is less than 40 MB, it will run in the multi-pass mode.

Figures 4 and 5 below illustrate the response time characteristics of sort and hash-join operations respectively as a function of available memory.

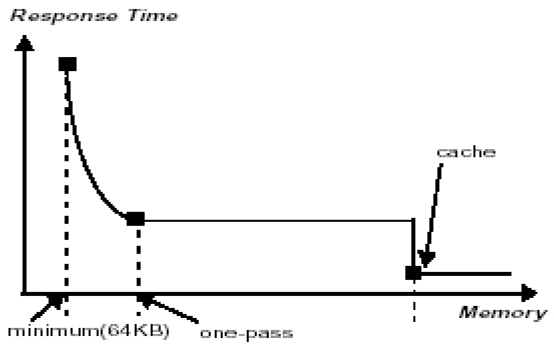


Figure 4: Effect of Memory on Sort Performance

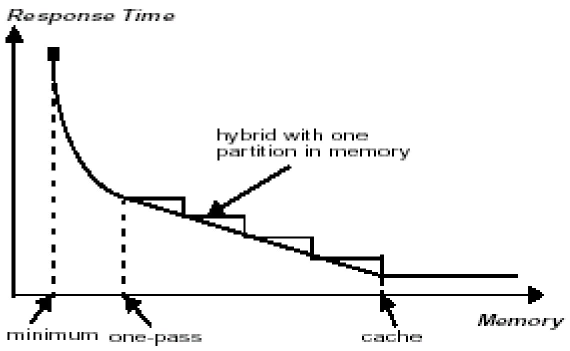


Figure 5: Effect of Memory on Hash-Join Performance

The *one-pass point* denotes the start of region when the operation runs in the one-pass mode whereas the *cache point* corresponds to the case when the amount of available memory is equal to the cache size of the operation. The sort curve is flat between these two points since the sort operation does not benefit from additional memory unless it can run in the cache mode. The response of hash-join operation decreases in steps between one-pass and cache points with each steps corresponding to an extra hash partition that can be kept in memory.

It is, therefore, obvious that proper sizing of the SQL work areas is extremely critical to performance of complex long running queries. Prior to Oracle9*i*, DBAs needed to manually adjust parameters like SORT_AREA_SIZE, HASH_AREA_SIZE, BITMAP_MERGE_AREA_SIZE and CREATE_BITMAP_AREA_SIZE to control the sizes of these working areas. This may be a challenging task since the queries running against any real world database widely differ in their nature and complexity making it very difficult to come up with a setting that optimizes the overall system performance without leading to any resource wastage.

Thankfully, Oracle9*i* provides an entirely new way of managing the PGA memory. As opposed to relying on DBAs to individually size the work areas, Oracle9*i* just lets them specify the maximum PGA memory that can be used by a database instance using a new initialization parameter, PGA_AGGREGATE_TARGET. The database server then automatically determines the optimal work area sizes for each active operation and distributes the available memory among them in a manner which maximizes the overall system performance.

How does Automatic SQL Execution Memory Management Work?

The Automatic SQL Execution Memory Management feature uses a sophisticated feedback mechanism (as depicted in figure 6) to dynamically regulate the amount of memory used by various active operations.

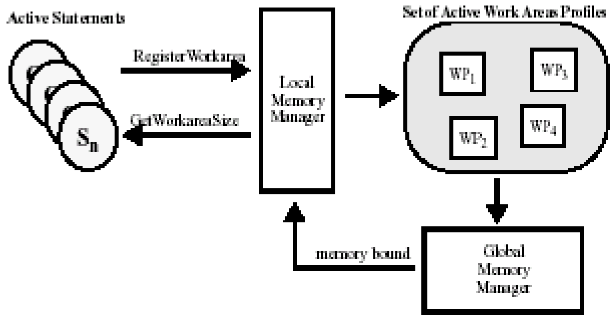


Figure 6: Memory Management Feedback Loop

When a SQL operation is started, it registers its work area profile using "local memory manager" services. The work area profile describes the characteristics of a work area such as its type (e.g. sort, hash-join, group-by), its current minimum, one-pass and cache memory requirement, its degree of parallelism (DOP) if it is a parallel operation, and amount of PGA memory currently in use. The local memory manager stores the registered work area profiles in the SGA. This information is continuously updated by the SQL operation to reflect its current memory need and consumption. This makes it possible for the database server to accurately determine the instance wide PGA memory need and consumption at any time by merely looking at the registered active work areas profiles. Finally, the work profile is de-registered, or removed from the set of active work area profiles, when the SQL operation completes.

The global memory manager is a service executed by one of the server background process. It indirectly determines the size of each active work areas by publishing a "global" limit or bound at regular intervals, generally every three seconds. The global memory bound is automatically derived from the number and characteristics of all active work area profiles and is used to constrain the size of each work area. Hence, the higher the global bound is, the more is memory available to each active operation. Since the goal of the global memory manager is to restrict the total instance wide PGA memory consumption below the PGA_AGGREGATE_TARGET setting, it also means that the bound is high when the system workload is light and vice-versa.

The calculation of the global memory bound is a two-step process. First an internal target (let us call it SQL Memory Target) is derived from the PGA_AGGREGATE_TARGET setting for amount of PGA memory available to SQL work areas. This to account for the part of PGA which isn't dynamically resizable (i.e. persistent area + a part of the run time area) and, to compensate for unforeseen factors such as delay on part of active operations to adapt to the newly published bound. In Oracle9*i*, the PGA memory used by shared server (MTS) sessions isn't automatically tuned. The Automatic SQL Execution Memory management, however, still tries to limit the overall instance PGA memory consumption to PGA_AGGREGATE_TARGET by treating the PGA memory used by shared server session as the "untunable" memory and adjusting the internal "SQL Memory Target" accordingly.

Once the SQL memory target is determined, it is then translated into a local limit, the global memory bound. While the global memory bound is usually published by a background process periodically, there may be situations where the bound may become very "stale" due to sudden increase in the workload thereby needing an immediate refresh by the SQL operation which detects the staleness. However, such incidences should be rare and most of the time the background computation of the bound should be adequate.

The feedback loop is closed by the local memory manager. It uses the current value of the memory bound and the current profile of a work area to determine the correct amount of PGA memory, called *expected size*, which can be made available to this work area. The calculation of the expected work area size for each active operation is done based on the following rules:

- The expected size can never be less than the minimum memory requirement of the operation and more than its cache size.
- If the global memory bound is between minimum and cache requirement, the expected size will be equal to the bound. The only exception to this rule is a sort operation. For a sort operation, the expected size under will be equal to one pass size if the bound is less than its cache size is less than the bound. This is due to the fact that the sort operation does not benefit from more than one pass memory unless the whole operation can be performed in cache, as explained earlier.
- For parallel operations, the expected size is multiplied by the degree of parallelism.
- Finally, no single operation will be allowed to "hog" all available memory. Therefore, the expected size can never be more than 5% of the overall target for serial and more than 30% for parallel operations.

The expected size is checked periodically by SQL operations, which then adapt their work area size to the specified value. Figure 7 below graphically illustrates how the expected size is determined based on the global memory bound. Let us say that the PGA_AGGREGATE_TARGET is set to 150 MB and the derived internal SQL memory target is 133 MB. The global memory manager then sets the bound to 20 MB i.e. no single work area can use more than 20 MB of the PGA memory. Now, let us also assume that the available memory needs to be distributed among six currently active operations with work area profile WP1, WP2, WP3…WP6. The work area profile WP1 corresponds to a serial sort operation with 27 MB cache and 7 MB one pass requirement. Similarly, the work area profile WP3 is for a parallel hash-join operation with degree of parallelism 2 and it requires 67 MB of memory to run in cache and 11MB to run in one-pass mode. With a global memory bound of 20 MB, the expected size for WP1 will be 7 MB (its one-pass requirement) and that of WP3 will be 40 MB i.e. twice the bound since its degree of parallelism is 2.

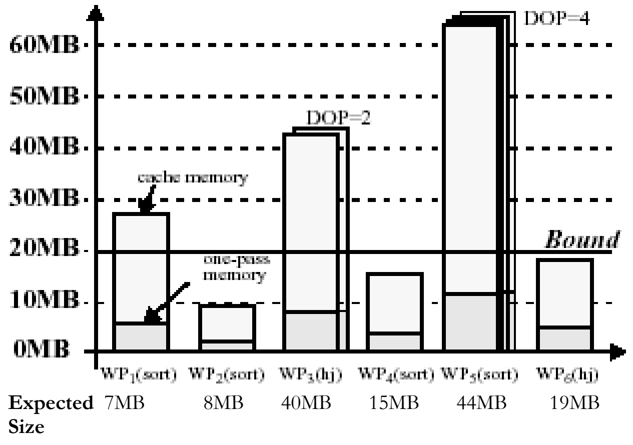


Figure 7: Calculation of Expected Size

BENEFITS OF AUTOMATIC SQL EXECUTION MEMORY MANAGEMENT

MANAGEABILITY

The Automatic SQL Execution Memory Management feature makes the management of PGA memory significantly easier by liberating administrators from having to size individual work areas. Instead, they can just set an instance wide upper limit on the amount of PGA memory available to an instance and leave the rest to the database server. In other words, with Oracle9i, DBAs no longer need to worry about parameters such as SORT_AREA_SIZE, HASH_AREA_SIZE, CREATE_BITMAP_AREA_SIZE and BITMAP_MERGE_AREA_SIZE; they now have to deal with just a single parameter, PGA_AGGREGATE_TARGET which is much easier and intuitive to configure. This feature also makes it possible for administrators to restrict the PGA memory consumed by an instance no matter what the workload is. This is significant because in the old scheme of things, the total instance PGA memory could grow in an uncontrolled fashion due to a sudden increase in the workload thereby degrading the performance of all applications sharing a host machine.

PERFORMANCE

Besides making the PGA memory configuration easier for administrators, the Automatic SQL execution memory management feature also maximizes the system performance by ensuring the most optimal use of available memory. Figures 8 and 9 summarize the results of an Oracle internal performance evaluation comparing the manual (pre-Oracle9i) and automatic modes. The test was performed on a SUN E4000 hardware with 10 167 MHz CPUs, 2 GB physical memory using a 30GB TPC-H database with the number of users varying from 1 to 20. The observations were recorded for three different configurations, 1. Manual SQL memory management with SORT_AREA_SIZE and HASH_AREA_SIZE set to 5 MB (Manual-5), 2. Manual SQL memory management with SORT_AREA_SIZE and HASH_AREA_SIZE set to 15 MB (Manual-15), and 3. Automatic SQL memory management with the parameter PGA_AGGREGATE_TARGET set to 1.5 GB.

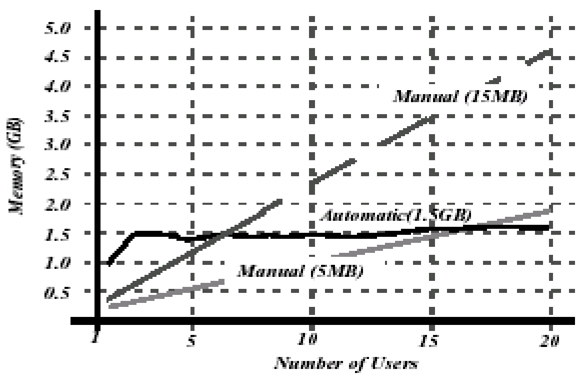


Figure 8: Automatic Vs. Manual (Memory Used)

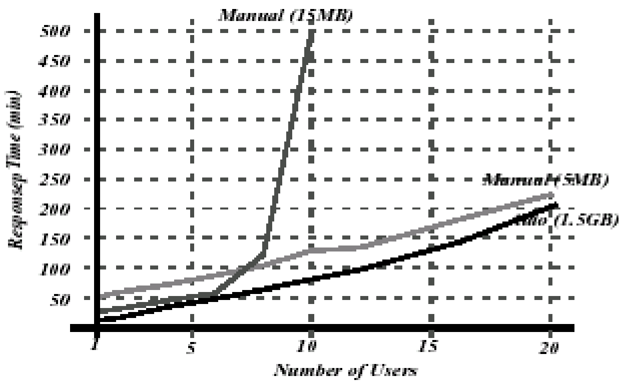


Figure 9: Automatic Vs. Manual (Response Time)

As can be seen in figure 8, the memory consumption increases linearly in both the manual modes. Also, while the available memory remains mostly underutilized (up to 16 users) for Manual-5 configuration, the system starts over-allocating memory with 6 or more users in Manual-15 configuration causing it to thrash very quickly. The Automatic mode obviously does a better job: it ensures full utilization of the available memory while keeping the total memory consumption below the administrators defined target irrespective of the number of users.

Figures 10 and 11 illustrate how well the Automatic SQL Memory management feature adapts to variations in the database workload. When subjected to a fluctuating workload, as shown in figure 8, it quickly adjusts the global memory bound ensuring that the total memory consumption remains very close (within about 5%) to the administrator defined limit.

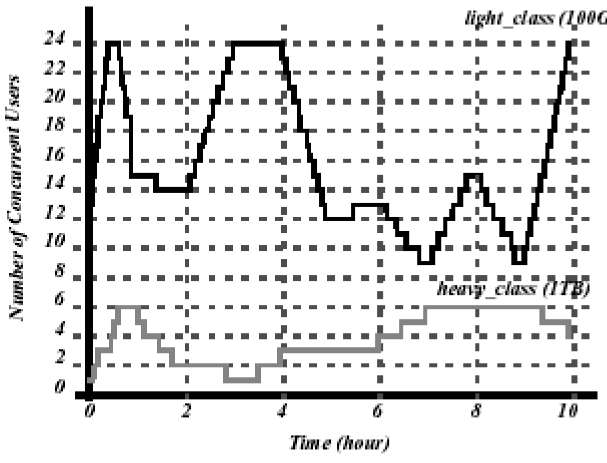


Figure 8: Varying Workload

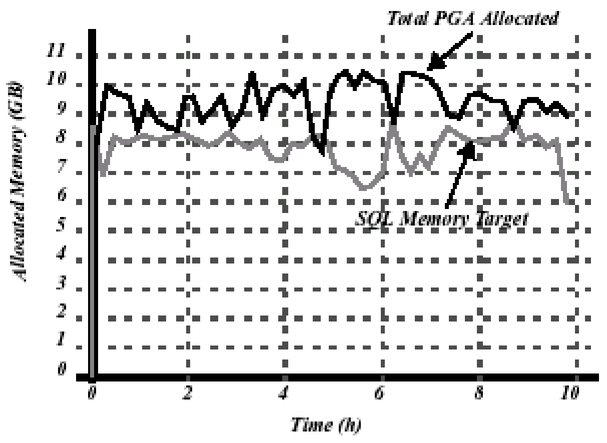


Figure 9: Total PGA Memory Consumption

The temporary excess allocation is expected since operations may take some time to adapt to a new global memory bound. It is, however, very short-lived and total memory consumption quickly falls below the PGA_AGGREGATE_TARGET setting.

Ability to Free Memory Back to OS

Prior to Oracle9i, there was no way to free the PGA memory once allocated to a process until it is terminated. The Automatic SQL Memory Management feature uses a new real free system call which allow the database instance to give memory back to the operating system when it is no longer being used or when the value of the parameter PGA_AGGREGATE_TARGET is reduced.

Enabling Automatic SQL Memory Management

The Automatic SQL Memory Management feature can be enabled by setting the parameter PGA_AGGREGATE_TARGET to a non-zero value. The default value of this parameter is zero meaning that the automatic mode is disabled by default. This is primarily to ensure backward compatibility. However, in view of all the benefits explained above, Oracle Corporation strongly recommends using this feature.

When the Automatic SQL Execution Memory Management feature is enabled, any *_AREA_SIZE settings are ignored. It is, however, still possible to temporarily revert to the manual mode, either at system or session level, by setting the parameter WORKAREA_SIZE_POLICY to MANUAL. Under such circumstances, the SQL work areas are sized based on SORT_AREA_SIZE, HASH_AREA_SIZE, etc., settings and the amount of memory used by operation running in the manual mode is treated as "untunable memory". The internal SQL memory target is adjusted accordingly and the amount of memory made available to operations running in the auto mode is reduced in order to honor the PGA_AGGREGATE_TARGET setting. It must be understood, however, that if the amount of memory used by the operation running in the manual mode is large, the total PGA consumption of an instance may exceed the PGA_AGGREGATE_TARGET value. Once again, the manual mode is supported only for backward compatibility reasons and it should be used only under exceptional circumstances.

Setting PGA_AGGREGATE_TARGET initially

For a New Database

The initial value of the PGA_AGGREGATE_TARGET initialization parameter should be based on the total amount of memory available to an instance, both SGA as well as PGA. The following rules-of-thumb may be used to divide memory between SGA and PGA depending on the nature of workload.

- For OLTP systems, the PGA memory requirement is minimal. Hence, most of the available memory (about 80%) should be given to the SGA and rest (i.e. 20%) to the PGA.
- For DSS or Data Warehousing systems running complex, memory intensive queries, most of the memory (about 70%) should be assigned to the PGA leaving rest (30%) for the SGA.
- For mixed workload, the PGA memory requirement will vary from one situation to another. However, as a guideline, about 60% of memory many be given to the SGA and rest (40%) to the PGA.

For an Existing Database

If the total amount of memory available to an instance cannot be precisely determined, the other option is to use the PGA usage statistics collected by the Oracle9*i* Database. The V$PGASTAT view contains a number of statistics related to the PGA consumption of an instance and is populated in both automatic as well as manual mode. Once the instance has operated in the manual mode for a while, the value of the statistic "Maximum PGA allocated" can provide a very a good starting value for the PGA_AGGREGATE_TARGET. This method should be used for picking up the right PGA target for databases migrated from Oracle8*i* or older releases.

Monitoring SQL Memory Usage

Following views can be used to monitor the functioning of the Automatic SQL Execution Memory Management feature.

INSTANCE LEVEL

V$PGASTAT

This view should be the primary reference for monitoring PGA memory usage. The information contained in this view includes PGA memory currently used, maximum PGA memory allocated since instance start up and the "PGA cache hit percentage". The PGA cache hit percentage is a new concept in Oracle9*i* defined as the percentage of the total amount of data processed by memory intensive SQL operators which was accommodated in the available PGA memory. For example, let us assume that four sort operations have been performed in an instance, three with 1 MB input data and fourth with 100 MB input data. The total input size is 103 MB. Now if one of the smaller sort operation is run in the one-pass mode, an extra pass over 1 MB of data is performed. Therefore, 1 MB of extra data had to be processed due to PGA not being large enough. The PGA cache hit percentage in this case, therefore would be (103/(103+1))*100 i.e. 99.03%. For further details on the meaning and definition of statistics available in the V$PGASTAT view, please refer to Oracle9i Database Performance Tuning Guide and Reference[2].

V$SYSSTAT

The following new statistics have been added to V$SYSSTAT and V$SESSTAT views.

Statistics Name	Description
work area memory allocated (KB)	Total amount of PGA memory dedicated to work areas allocated on behalf of a given session (V$SESSTAT) or on the system (V$SYSSTAT). This includes work areas allocated under both MANUAL as well as AUTO mode. For DSS workload this should represent most of the PGA memory.
work area executions - optimal size	The cumulative count of work areas that were executed in cache mode. A sort area can be said to have an optimal size if it did not need to spill to disk. Same for hash-join.
work area executions - one pass size	The cumulative count of work areas using the one pass size. One pass is generally used for big work areas where spilling to disk cannot be avoided.
work area executions - multipasses size	The cumulative count of work areas running in more than one pass. This should be avoided and may be a symptom of a poorly tuned system.

PROCESS/SESSION/QUERY LEVEL

V$SESSTAT

As described above.

V$PROCESS

The following four new columns have been added to the V$PROCESS view to report the PGA memory allocated and used by an Oracle process.

Column	Description

[2] Chapter 14, Memory Configuration and Use, http://otn.oracle.com/docs/products/oracle9i/doc_library/release2/server.920/a96533/memory.htm#47603

PGA_USED_MEM	PGA memory currently used by the process.
PGA_ALLOC_MEM	PGA memory currently allocated by the process. Includes free PGA memory not yet released to the OS by the server process.
PGA_FREEABLE_MEM	Part of the allocated PGA memory that can be freed.
PGA_MAX_MEM	The maximum PGA memory ever allocated by the process.

V$SQL_WORKAREA

This view displays information about work areas used by SQL cursors. Each SQL statement stored in the shared pool has one or more child cursors which are listed in the V$SQL dynamic view. The V$SQL_WORKAREA dynamic view lists all work areas needed by these child cursors. It can be joined with V$SQLAREA on (address, hash_value) and with V$SQL on (address, hash_value, child_number).

V$SQL_WORKAREA_ACTIVE

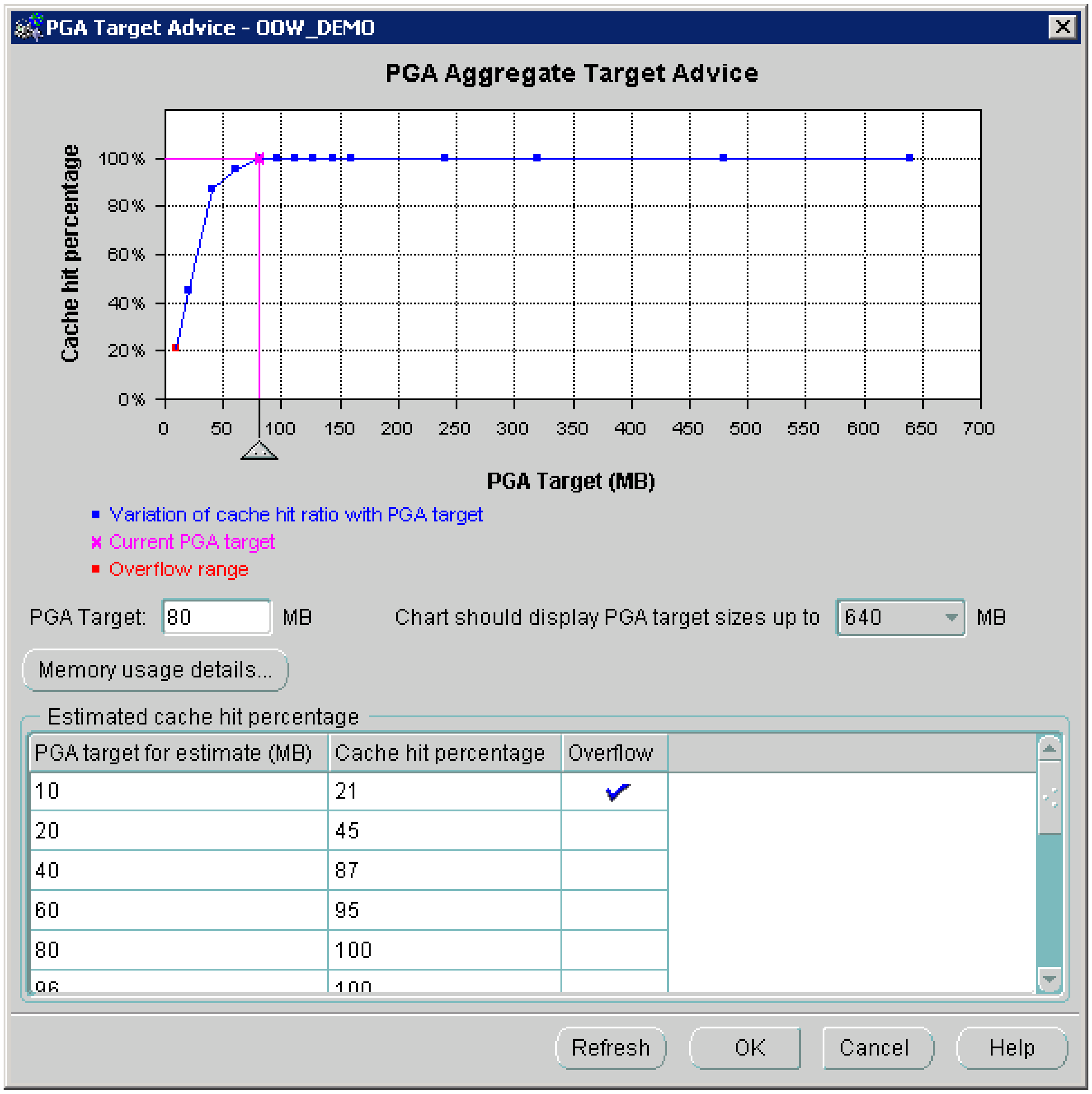
This view is structurally same as V$SQL_WORKAREA. But while the latter contains work area for all cached cursors, V$SQL_WORKAREA_ACTIVE view only list those which are currently active.

TUNING PGA_AGGREGATE_TARGET

Oracle9i Release 2 assists administrators in deciding the optimal PGA size by providing two advice performance views, V$PGA_TARGET_ADVICE and V$PGA_TARGET_ADVICE_HISTOGRAM. By examining these two views, it is possible to determine how key PGA statistics will be impacted as the value of the PGA_AGGREGATE_TARGET parameter is changed. In both these views, the values of PGA_AGGREGATE_TARGET used for the prediction are derived from fractions and multiples of the current value of that parameter, to assess possible higher and lower values. Oracle generates PGA advice performance views by recording the workload history and then simulating this history for different values of PGA_AGGREGATE_TARGET. The simulation process is performed continuously in the background by the global memory manager.

The figure 12 below shows the content of V$PGA_TARGET_ADVICE view as displayed in Oracle Enterprise Manager. The curve shows how the PGA "cache hit percentage" metric improves as the value of the configuration parameter PGA_AGGREGATE_TARGET increases. The "over flow" zone of the curve refers to the values of PGA_AGGREGATE_TARGET smaller than the minimum PGA requirement of the instance. If PGA_AGGREGATE_TARGET is set within the over-allocation zone, the memory manager will over-allocate memory and actual PGA memory consumed will be more than the limit set by the DBA. It is therefore meaningless to set a value of PGA_AGGREGATE_TARGET in that zone.

Beyond the over-allocation zone, the value of the PGA cache hit percentage increases rapidly. This is due to an increase in the number of work areas which run cache or one-pass and a decrease in the number of multi-pass executions. At some point, somewhere around 40 MB on the given curve, there is an inflection in the curve which corresponds to the point where most (probably all) work areas can run at worst one-pass. After this inflection, the cache hit percentage keeps increasing but at a lower pace up to the point where it starts to taper off and shows only slight improvement when PGA_AGGREGATE_TARGET is increased. In the above example, this happens when PGA_AGGREGATE_TARGET reaches 80 MB where the cache hit ratio starts approaching close to 100%. The given curve tells us that the current setting of PGA_AGGREGATE_TARGET (80MB) is optimal and no further gains are likely if the amount of the PGA memory available to the instance is increased.



■ Variation of cache hit ratio with PGA target
✖ Current PGA target
■ Overflow range

PGA Target: 80 MB

Chart should display PGA target sizes up to 640 MB

Estimated cache hit percentage

PGA target for estimate (MB)	Cache hit percentage	Overflow
10	21	✓
20	45	
40	87	
60	95	
80	100	
96	100	

Figure 12: PGA Advisory (Oracle Enterprise Manager)

Customer Implementation Case Studies

In view of the manageability and performance benefits enumerated earlier, customers have been quick to adopt the Automatic SQL Execution Memory management features. Following are some of the customers who have extensively tested this feature and have successfully deployed it in their production environment.

Customer Name	Business Description	System/Environment Details
Colgate USA	Consumer goods multi-national (http://www.colgate.com)	3 TB Oracle9i Release 2 Data Warehouse running on 24 CPU IBM Regatta p690 with 96 GB RAM, AIX 5.1
Talkline	Mobile phone & services company in Germany	• Sun E-10000 (20 CPUs, 8GB RAM)

	(http://www.talkline.de)	• 4 x A5200 (72 x 18 GB disks), 1 x A3500 (18 x 9 GB disks), 340 MB/sec. throughput • DWH: 1/2 TB raw data, largest table 100 GB
Postbank	A Germany based international banking group (http://www.postbank.de)	Oracle9i Release 2 RAC running on HP-UX

CONCLUSION

Memory is a critical system resource. Since memory access is many times faster than accessing data from disk, effective utilization of memory is necessary for optimal system performance. Administrators, therefore, continuously strive to tune memory related parameters to maximize system performance and ensure the most efficient use of system memory. Oracle9*i* automates much of the tuning process and allows administrators to alter the instance memory configuration dynamically. These features provide improved system performance, ensure optimal memory utilization and help reduce the maintenance downtime.