Doc ID: Note:223730.1 **Content Type:** TEXT/PLAIN
Subject: Automatic PGA Memory Managment in 9i **Creation Date:** 17-DEC-2002
Type: BULLETIN **Last Revision Date:** 08-JAN-2003
Status: PUBLISHED

PURPOSE

This Document briefly describes how Oracle 9i manage PGA work area and how to tune it and some of the common issues and some of the common misunderstood issues. This Note is continuation to [NOTE:147806.1] and [NOTE:148346.1] and should provide more information about this New 9i Feature.

SCOPE & APPLICATION

DBAs and Application Developers might benefit from this article.

Automatic PGA Memory Management

Process Global Area, often known as the Program Global Area (PGA) resides in the process private memory of the server process. It contains global variables and data structures and control information for a server process. example of such information is the runtime area of a cursor. Each time a cursor is executed, a new runtime area is created for that cursor in the PGA memory region of the server process executing that cursor.

The performance of complex long running queries, typical in a DSS environment, depend to a large extent on the memory available in the Program Global Area (PGA). which is also called work area.

The size of a work area can be controlled and tuned. Generally, bigger work areas can significantly improve the performance of a particular operator at the cost of higher memory consumption. Ideally, the size of a work area is big enough that it can accommodate the input data and auxiliary memory structures allocated by its associated SQL operator. This is known as the optimal size of a work area (e.g. a memory sort). When the size of the work area is smaller than optimal (e.g. a disk sort), the response time increases, because an extra pass is performed over part of the input data. This is known as the one-pass size of the work area. Under the one-pass threshold, when the size of a work area is far too small compared to the input data size, multiple passes over the input data are needed. This could dramatically increase the response time of the operator. This is known as the multi-pass size of the work area.

In Oracle8i administrators sized the PGA by carefully adjusting a number of initialization parameters, such as, SORT_AREA_SIZE, HASH_AREA_SIZE, BITMAP_MERGE_AREA_SIZE, and CREATE_BITMAP_AREA_SIZE, etc.

Oracle9i provides an option to completely automate the management of PGA memory. Administrators merely need to specify the maximum amount of PGA memory available to an instance using a newly introduced initialization parameter PGA_AGGREGATE_TARGET. The database server automatically distributes this memory among various active queries in an intelligent manner so as to ensure maximum performance benefits and the most efficient utilization of memory. Furthermore, Oracle9i can adapt itself to changing workload thus utilizing resources efficiently regardless of the load on the system. The amount of the PGA memory available to an instance can be changed dynamically by altering the value of the PGA_AGGREGATE_TARGET parameter making it possible to add to and remove PGA memory from an active instance online. Since the database engine itself is better equipped to determine SQL execution memory requirements, database administrators should use this feature and not try to tune the PGA manually. This should translate to better

throughput for large number of users on the system as well as improved response time for queries.

The automatic SQL execution memory management feature is enabled by setting the parameter WORKAREA_SIZE_POLICY to AUTO and by specifying a size of PGA_AGGREGATE_TARGET in the initialization file. These two parameters can also be set dynamically using the ALTER SYSTEM command. In the absence of either of these parameters, the database will revert to manual PGA management mode. In Oracle9i Release 2, advisory for PGA_AGGREGATE_TARGET was introduced. Just like in Buffer Cache Advisory, the PGA Advisory will suggest the appropriate size for PGA memory and thus make PGA tuning an even simpler task.

How To Tune PGA_AGGREGATE_TARGET

The first question we will have when we set this parameter is what is the best value for it ?

To determine the appropriate setting for PGA_AGGREGATE_TARGET parameter I recommend to follow the following steps

1- Make a first estimate for PGA_AGGREGATE_TARGET based on the following rule

- For OLTP systems

   PGA_AGGREGATE_TARGET = (<Total Physical Memory > * 80%) * 20%

- For DSS systems

   PGA_AGGREGATE_TARGET = (<Total Physical Memory > * 80%) * 50%

So for example if we have an Oracle instance configured on system with 16G of Physical memory, then the suggested PGA_AGGREGATE_TARGET parameter value we should start with incase we have OLTP system is (16 G * 80%)*20% ~= 2.5G and incase we have DSS system is (16 G * 80%)* 50% ~= 6.5 G.

In the above equation we assume that 20 % of the memory will be used by the OS, and in OLTP system 20 % of the remaining memory will be used for PGA_AGGREGATE_TARGET and the remaining memory is going for Oracle SGA memory and non-oracle processes memory. So make sure that you have enough memory for your SGA and also for non-oracle processes

2- Second step in tuning the PGA_AGGREGATE_TARGET is to monitor performance using available PGA statistics and see if PGA_AGGREGATE_TARGET is under sized or over sized. Several dynamic performance views are available for this purpose:

- V$PGASTAT

This view provides instance-level statistics on the PGA memory usage and the automatic PGA memory manager. For example:

```
SELECT * FROM V$PGASTAT;

NAME                                                     VALUE
---------------------------------------------------------------
aggregate PGA target parameter                           524288000 bytes
aggregate PGA auto target                                463435776 bytes
global memory bound                                      25600 bytes
total PGA inuse                                          9353216 bytes
total PGA allocated                                      73516032 bytes
maximum PGA allocated                                    698371072 bytes
total PGA used for auto workareas                        0 bytes
maximum PGA used for auto workareas                      560744448 bytes
total PGA used for manual workareas                      0 bytes
maximum PGA used for manual workareas                    0 bytes
over allocation count                                    0 bytes
total bytes processed                                    4.0072E+10 bytes
```

```
total extra bytes read/written                  3.1517E+10 bytes
cache hit percentage                            55.97 percent
```

Main statistics to look at

(a) aggregate PGA auto target : This gives the amount of PGA memory Oracle can use for work areas running in automatic mode. This part of memory represent the tunable part of PGA memory,i.e. memory allocated for intensive memory SQL operators like sorts, hash-join, group-by, bitmap merge and bitmap index create. This memory part can be shrinked/expanded in function of the system load. Other parts of PGA memory are known as untunable, i.e. they require a size that can't be negociated (e.g. context information for each session, for each open/active cursor, PL/SQL or Java memory).

So, the aggregate PGA auto target should not be small compared to the value of PGA_AGGREGATE_TARGET. You must ensure that enough PGA memory is left for work areas running in automatic mode.

(b) total PGA used for auto workarea: This gives the actual tunable PGA memory used by the system. The 'maximum PGA used for auto workareas' gives the maximum reached by previous statistic since instance startup.
(c) total PGA in used: This gives the total PGA memory in use. The detail of this value can be found in the PGA_USED_MEM column of the v$process view.

Oracle92 only:
(d) over allocation count: Over-allocating PGA memory can happen if the value of PGA_AGGREGATE_TARGET is too small to accommodate the untunable PGA memory part plus the minimum memory required to execute the work area workload. When this happens, Oracle cannot honor the initialization parameter PGA_AGGREGATE_TARGET, and extra PGA memory needs to be allocated. over allocation count is the number of time the system was detected in this state since database startup. This count should ideally be equal to zero.

Oracle92 only:
(e) cache hit percentage: This metric is computed by Oracle to reflect the performance of the PGA memory component. It is cumulative from instance start-up. A value of 100% means that all work areas executed by the system since instance start-up have used an optimal amount of PGA memory. This is, of course, ideal but rarely happens except maybe for pure OLTP systems. In reality, some work areas run one-pass or even multi-pass, depending on the overall size of the PGA memory. When a work area cannot run optimally, one or more extra passes is performed over the input data. This reduces the cache hit percentage in proportion to the size of the input data and the number of extra passes performed. this value if computed from the "total bytes processed" and "total extra bytes read/written" statistics available in the same view using the following formula:

$$\text{PGA Cache Hit Ratio} = \frac{\text{total bytes processed} * 100}{(\text{total bytes processed} + \text{total extra bytes read/written})}$$

- V$SQL_WORKAREA_HISTOGRAM (Oracle92 only)

This view shows the number of work areas executed with optimal memory size, one pass memory size, and multi-pass memory size since instance start-up. Statistics in this view are subdivided into buckets that are defined by the optimal memory requirement of the work area. Each bucket is identified by a range of optimal memory requirements specified by the values of the columns LOW_OPTIMAL_SIZE and HIGH_OPTIMAL_SIZE.

Example :

The following query shows statistics for all nonempty buckets.

```
SELECT LOW_OPTIMAL_SIZE/1024 low_kb,(HIGH_OPTIMAL_SIZE+1)/1024 high_kb,
       optimal_executions, onepass_executions, multipasses_executions
FROM   v$sql_workarea_histogram
WHERE  total_executions != 0;
```

The result of the query might look like the following:

```
LOW_KB HIGH_KB OPTIMAL_EXECUTIONS ONEPASS_EXECUTIONS MULTIPASSES_EXECUTIONS
------ ------- ------------------ ------------------ ----------------------
8      16      156255             0                  0
16     32      150                0                  0
32     64      89                 0                  0
64     128     13                 0                  0
128    256     60                 0                  0
256    512     8                  0                  0
512    1024    657                0                  0
1024   2048    551                16                 0
2048   4096    538                26                 0
4096   8192    243                28                 0
8192   16384   137                35                 0
16384  32768   45                 107                0
32768  65536   0                  153                0
65536  131072  0                  73                 0
131072 262144  0                  44                 0
262144 524288  0                  22                 0
```

The query result shows that, in the 1024 KB to 2048 KB bucket, 551 work areas used an optimal amount of memory, while 16 ran in one-pass mode and none ran in multi-pass mode. It also shows that all work areas under 1 MB were able to run in optimal mode.

You can also use V$SQL_WORKAREA_HISTOGRAM to find the percentage of times work areas were executed in optimal, one-pass, or multi-pass mode since start-up.

Example :

```
SELECT optimal_count, round(optimal_count*100/total, 2) optimal_perc,
       onepass_count, round(onepass_count*100/total, 2) onepass_perc,
       multipass_count, round(multipass_count*100/total, 2) multipass_perc
FROM
       (SELECT decode(sum(total_executions), 0, 1, sum(total_executions)) total,
               sum(OPTIMAL_EXECUTIONS) optimal_count,
               sum(ONEPASS_EXECUTIONS) onepass_count,
               sum(MULTIPASSES_EXECUTIONS) multipass_count
        FROM   v$sql_workarea_histogram
        WHERE  low_optimal_size > 64*1024);  ---- for 64 K optimal size
```

- V$SQL_WORKAREA_ACTIVE

This view can be used to display the work areas that are active (or executing) in the instance. Small active sorts (under 64 KB) are excluded from the view. Use this view to precisely monitor the size of all active work areas and to determine if these active work areas spill to a temporary segment.

Example :

```
SELECT to_number(decode(SID, 65535, NULL, SID)) sid,
       operation_type OPERATION,trunc(EXPECTED_SIZE/1024) ESIZE,
       trunc(ACTUAL_MEM_USED/1024) MEM, trunc(MAX_MEM_USED/1024) "MAX MEM",
       NUMBER_PASSES PASS, trunc(TEMPSEG_SIZE/1024) TSIZE
FROM V$SQL_WORKAREA_ACTIVE
ORDER BY 1,2;
```

```
SID OPERATION         ESIZE     MEM       MAX MEM    PASS TSIZE
--- ----------------- --------- --------- --------- ----- -------
8   GROUP BY (SORT)   315       280       904         0
8   HASH-JOIN         2995      2377      2430        1   20000
9   GROUP BY (SORT)   34300     22688     22688       0
11  HASH-JOIN         18044     54482     54482       0
12  HASH-JOIN         18044     11406     21406       1   120000
```

This output shows that session 12 (column SID) is running a hash-join having its work area running in one-pass mode (PASS column). This work area is currently using 11406 KB of memory (MEM column) and has used, in the past, up to 21406 KB of PGA memory (MAX MEM column). It has also spilled to a temporary segment of size 120000 KB. Finally, the column ESIZE indicates the maximum amount of memory that the PGA memory manager expects this hash-join to use. This maximum is dynamically computed by the PGA memory manager according to workload.

When a work area is deallocated—that is, when the execution of its associated SQL operator is complete—the work area is automatically removed from the V$SQL_WORKAREA_ACTIVE view.

- [NOTE:148346.1] have some other queries we use to monitor SQL execution memory

3- The Third and last step is tuning the PGA_AGGREGATE_TARGET. In Oracle 9i Release 2 we have 2 new views that help us in this task

- V$PGA_TARGET_ADVICE
- V$PGA_TARGET_ADVICE_HISTOGRAM

By examining these two views, you will be able to determine how key PGA statistics will be impacted if you change the value of PGA_AGGREGATE_TARGET.

To enable automatic generation of PGA advice performance views, make sure the following parameters are set:

- PGA_AGGREGATE_TARGET
- STATISTICS_LEVEL. Set this to TYPICAL (the default) or ALL; setting this parameter to BASIC turns off generation of PGA performance advice views.

The content of these PGA advice performance views is reset at instance startup or when PGA_AGGREGATE_TARGET is altered.

V$PGA_TARGET_ADVICE view predicts how the statistics cache hit percentage and over allocation count in V$PGASTAT will be impacted if you change the value of the initialization parameter PGA_AGGREGATE_TARGET.

The following select statement can be used to find this information

```
SELECT round(PGA_TARGET_FOR_ESTIMATE/1024/1024) target_mb,
       ESTD_PGA_CACHE_HIT_PERCENTAGE cache_hit_perc,
       ESTD_OVERALLOC_COUNT
FROM   v$pga_target_advice;
```

The output of this query might look like the following:

```
TARGET_MB  CACHE_HIT_PERC ESTD_OVERALLOC_COUNT
---------- -------------- --------------------
63         23             367
125        24             30
250        30             3
375        39             0
500        58             0
600        59             0
700        59             0
800        60             0
900        60             0
1000       61             0
1500       67             0
2000       76             0
3000       83             0
4000       85             0
```

From the above results we should set the PGA_AGGREGATE_TARGET parameter to a value where we avoid any over allocation, so lowest PGA_AGGREGATE_TARGET value we can set is 375 (where ESTD_OVERALLOC_COUNT is 0)

After eliminating over-allocations, the goal is to maximize the PGA cache hit percentage, based on your response-time requirement and memory constraints.

V$PGA_TARGET_ADVICE_HISTOGRAM view predicts how the statistics displayed by the performance view V$SQL_WORKAREA_HISTOGRAM will be impacted if you change the value of the initialization parameter PGA_AGGREGATE_TARGET. You can use the dynamic view V$PGA_TARGET_ADVICE_HISTOGRAM to view detailed information on the predicted number of optimal, one-pass and multi-pass work area executions for the set of PGA_AGGREGATE_TARGET values you use for the prediction.

Common issues

1- When we set the PGA_AGGREGATE_TARGET and WORKAREA_SIZE_POLICY to auto then the *_area_size parameter are automatically ignored and oracle will automatically use the computed value for these parameters.

2- In Oracle 8i and earlier, the PGA memory was static, once the process started and started to allocate memory for it's PGA area then it will not release it back to the OS unless the process exits or dies. But the OS and under heavy memory pressure will decide to page out unused memory pages belongs to a process PGA to the swap space.

In Oracle 9i and under the automatic PGA memory management, Oracle will be able to unallocate memory from a process PGA which is not using it any more so another process can use it, also it can adjust the different work areas size to accommodate the current workload and the amount of memory can be used.

3- Using automatic PGA memory management feature will help limiting resources used by oracle process, and will use it more efficiently.

4- Using automatic PGA memory management will help also reducing the possibility of getting ora-4030 errors unless we hit a OS limit, because work area sizes will be controlled and adjusted automatically based on the PGA_AGGGREGATE_TARGET parameter first and then the current work load.

5- If column ESTD_OVERALLOCATION_COUNT in the V$PGA_TARGET_ADVICE VIEW is nonzero, It indicates that PGA_AGGREGATE_TARGET is too small to even meet the minimum PGA memory needs. 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 you set. It is therefore meaningless to set a value of PGA_AGGREGATE_TARGET in that zone.

6- Some customer reported that SQL LOADER in Oracle 9i is slower than SQL Loader in Oracle 8i, and example of this is [BUG:2230115] which was closed as not a bug. Using PGA_AGGREGATE_TARGET alleviated the problem.

7- PGA_AGGREGATE_TARGET is not supported on VMS, for more information please refer to [NOTE:175216.1] "Oracle9i Release Notes Release 1 (9.0.1) for Alpha OpenVMS". ORA-3113 is returned on instance startup when init.ora PGA_AGGREGATE_TARGET is set.
8- Setting PGA_AGGREGATE_TARGET in 9.0.1 on HP-UX 11.0 may panic the OS. for more information please refer to [NOTE:43507.1] "ALERT HP-UX Patch Levels Advised" and Bug:2122307.

Known Bugs

- [BUG:2109788]
Details: Attempting to set pga_aggregate_target over 4000Gb should error with ORA-4032 but no error is signalled.
Fixed-Releases: 9.2.0.1.0

- Bug:2122307 HP System crash when setting PGA_AGGREGATE_TARGET to 10M or more in Oracle 9.0.1.

This is basically an OS Problem that cause the crash. The system call pattern automatic PGA management is using causing HP/UX to try to extend fixed region

and leads to memory allocation failures.

To resolve the bug both this patch and PHKL_25188 (or later) must be installed.

RELATED DOCUMENTS

- [NOTE:147806.1] Oracle9i New Feature Automated SQL Execution Memory Management
- [NOTE:148346.1] Oracle9i Monitoring Automated SQL Execution Memory Management
- [NOTE:175216.1] Oracle9i Release Notes Release 1 (9.0.1) for Alpha OpenVMS
- [NOTE:43507.1] ALERT HP-UX Patch Levels Advised
- Oracle 9i Database Performance tuning Guide and reference, Chapter 14

.