The Self-Managing Database: Automatic SGA Memory Management

An Oracle White Paper
Nov. 2003

ORACLE

The Self-Managing Database: Automatic SGA Memory Management

The Self-Managing Database: Automatic SGA Memory Management

INTRODUCTION

One of the key self-management enhancements in the Oracle Database 10g is Automatic Shared (SGA) Memory Management. This functionality automates the management of shared memory used by an Oracle Database 10g instance and liberates administrators from having to manually configure the sizes of shared memory components. Besides making more effective use of available memory and thereby reducing the cost incurred of acquiring additional hardware memory resources, the Automatic Shared Memory Management feature will significantly simply Oracle database administration by introducing a more dynamic, flexible and adaptive memory management scheme.

This paper introduces this functionality and illustrates its advantages.

CURRENT CHALLENGES

The Shared Global Area (SGA) in Oracle comprises multiple memory *components* -- a component being a pool of memory used to satisfy a particular class of memory allocation requests. Examples of memory components include the shared pool (used for allocating memory for SQL and PL/SQL execution), java pool (used for java objects and other java execution memory), buffer cache (used for caching disk blocks), etc.

In past releases, the Oracle administrator was required to manually set a number of parameters for specifying different SGA component sizes, such as SHARED_POOL_SIZE, DB_CACHE_SIZE, LARGE_POOL_SIZE, and JAVA_POOL_SIZE.

The task of manually adjusting the sizes of individual SGA components could pose a few challenges. It may not be easy to determine the optimal sizes of these components suitable for a given workload. Oracle9i alleviated this problem to a great extent by introducing advisory mechanisms that allow DBAs to determine the optimal sizes of the buffer cache and shared pool. However, these recommendations still had to be implemented by the administrator. This challenge is further compounded in situations in which the workload tends to vary with the time of the day e.g online users during the day and batch jobs at night. Sizing for peak load could mean memory wastage while under-sizing may

cause out-of-memory errors (ORA-4031). For example if a system is configured with a big large pool to accommodate a nightly RMAN backup job, most of this memory – which could have been better utilized in the buffer cache or shared pool for OLTP activity – remains unused for the most part of the day. At the same time, the cost of failures could be prohibitive from a business point of view leaving administrators with few other options.

INTRODUCING AUTOMATIC SHARED MEMORY MANAGEMENT

To resolve these challenges, Oracle Database 10g introduces Automatic Shared Memory Management. In Oracle Database 10g, DBAs can just specify the total amount of SGA memory available to an instance using the new parameter SGA_TARGET. The database server then automatically distributes the available memory among various components as required. The Automatic Shared Memory Management feature is based on a sophisticated algorithm internal to the database that continuously monitors the distribution of memory and changes it periodically as needed, according to the demands of the workload.

The SGA_TARGET Parameter

The SGA_TARGET parameter reflects the total size of the SGA and includes memory for:

- Fixed SGA and other internal allocations needed by the Oracle instance
- Log buffer
- Shared Pool
- Java Pool
- Buffer Cache
- Keep/Recycle buffer caches (if specified)
- Non standard block size buffer caches (if specified)
- Streams Pool (New in Oracle Database 10g)

An important point to note is that SGA_TARGET now includes the entire memory for the SGA. This is a change from past releases in which memory for the internal allocations and fixed SGA was added to the sum of the configured SGA memory parameters. Thus, SGA_TARGET allows the user to precisely control the size of the shared memory area allocated by Oracle.

Automatically Managed SGA Components

When SGA_TARGET is set, the most commonly configured components are sized automatically. These include:

1. Shared pool (for SQL and PL/SQL execution)

2. Java pool for (java execution state)

3. Large pool (for large allocations such as RMAN backup buffers)

4. Buffer cache

There is no need to set the of size any of the above components explicitly and by default the parameters for these components will appear to have values of zero. Whenever a component needs memory, it can request that it be transferred from another component via the internal auto-tuning mechanism. This will happen transparently without user-intervention.

The performance of each of these components is also monitored by the Oracle instance. Now the instance uses internal views and statistics to determine how to optimally distribute memory among the automatically sized components. Thus, as the workload changes, memory would be redistributed to ensure optimal performance with the new workload. This algorithm is never complacent and always tries to find the optimal distribution by taking into consideration long term as well as short terms trends.

In the following example, the shared pool advisory shows that the shared pool is sized to a value below the knee of the curve and hence, growing the shared pool will considerably improve parse times. In this scenario, memory may be transferred from the buffer cache to the shared pool by the auto-tuning algorithm, in order to ensure a more optimal distribution of the memory.

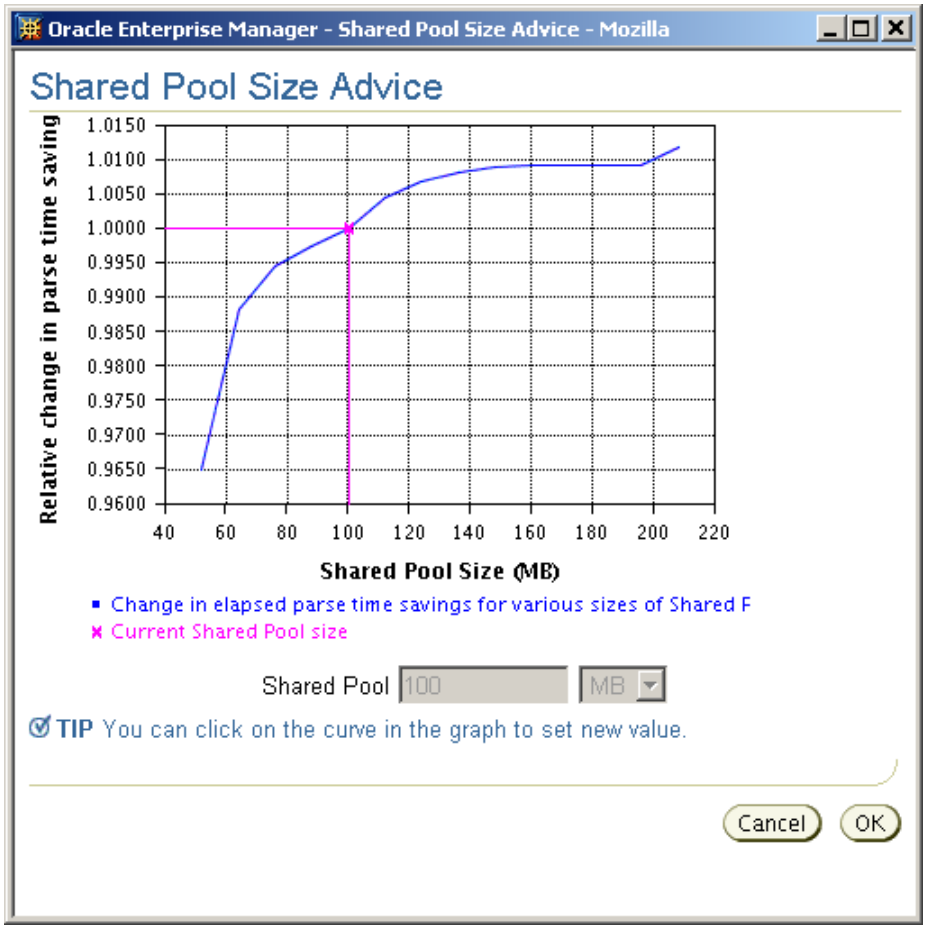

Fig 1: Shared Pool Advisory

The administrator can still exercise some control over the sizes of the auto-tuned components by specifying minimum values for each of these components. This can be useful in cases in which the administrator knows that an application needs a minimum amount of memory in certain components to function properly. The minimum value of a component is specified by setting the corresponding parameter for the component.

Here is an example configuration:

SGA_TARGET = 256M

SHARED_POOL_SIZE = 32M

DB_CACHE_SIZE = 100M

In the above example, the shared pool and the default buffer pool will not be sized below the specified values (32M and 100M, respectively). This implies that the remaining 124M can be distributed across the 4 components. Thus, the actual distribution of values between the SGA components may be as follows:

Actual Shared Pool Size = 64M

Actual buffer cache size = 128M

Actual java pool size = 60M

Actual Large Pool Size = 4M

The fixed view V$SGA_DYNAMIC_COMPONENTS displays the current size of each SGA component while the parameter values (e.g. DB_CACHE_SIZE, SHARED_POOL_SIZE) specify the minimum values. The current sizes of the SGA components can also be determined by looking at the Enterprise Manager memory configuration page.

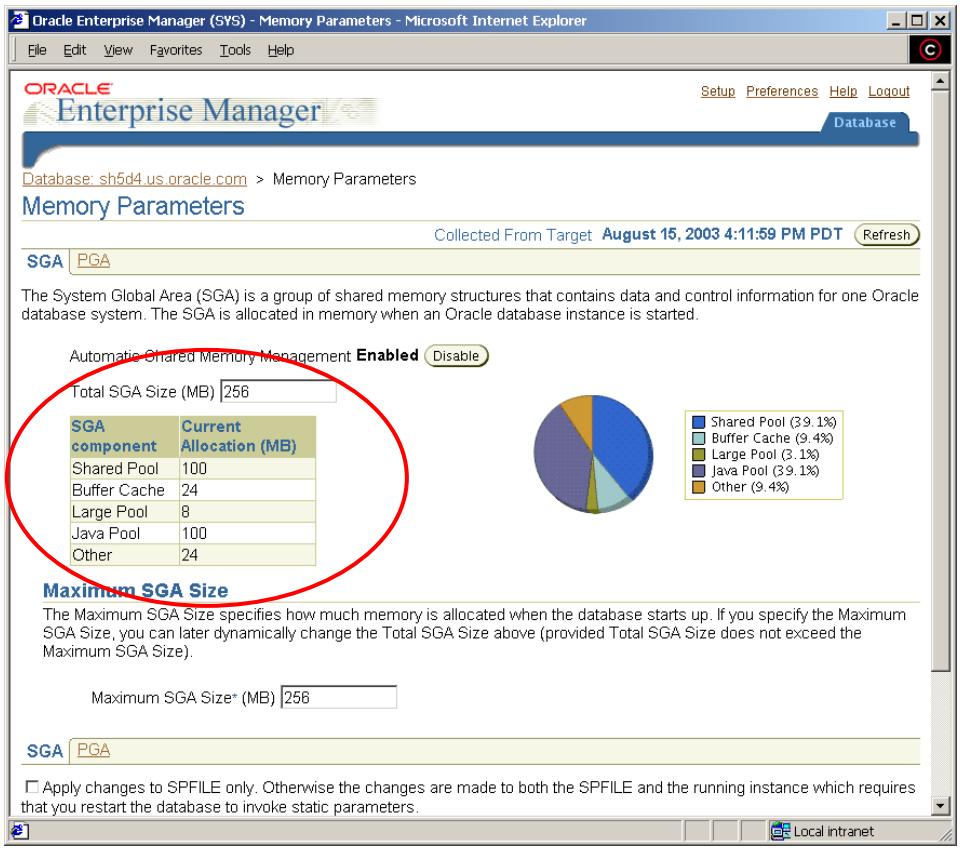

Fig 1: EM displays the current sizes of automatically tuned SGA components

Manually Sized SGA Components

There are a few SGA components whose sizes are not automatically adjusted. The administrator needs to specify the sizes of these components explicitly, if needed by the application. Such components are:

- Keep/Recycle buffer caches (controlled by DB_KEEP_CACHE_SIZE and DB_RECYCLE_CACHE_SIZE)

- Additional buffer caches for non-standard block sizes (controlled by DB_<N>K_CACHE_SIZE, N={2,4,8,16,32})
- Streams Pool (controlled by the new parameter STREAMS_POOL_SIZE)

The sizes of these components is determined by the administrator-defined value of their corresponding parameters. These values can, of course, be changed any time either using Enterprise Manager or from the command-line via the ALTER SYSTEM command.

The memory consumed by manually sized components reduces the amount of memory available for automatic adjustment. So for example, in the following configuration:

- SGA_TARGET = 256M
- DB_8K_CACHE_SIZE = 32M
- STREAMS_POOL_SIZE = 24M

The instance has only 200M (256 – 32 – 24) remaining to be distributed among the automatically sized components

BENEFITS

More Flexible and Adaptive Memory Utilization

The most significant benefit of using automatic SGA memory management is that the sizes of the different SGA components are flexible and will adapt to the needs of a workload without requiring user intervention.

Let us illustrate this with an example. Consider a manual configuration in where 1G of memory is available for SGA and distributed as follows (for the purpose of simplicity we ignore other SGA components for now):

```
SHARED_POOL_SIZE=128M
DB_CACHE_SIZE=896M
```

In this case, if the application ever tries to allocate more than 128M of memory from the shared pool, it will receive an ORA-4031 indicating that available shared pool has been exhausted. Note that when this condition happens, there may be free memory in the buffer cache - but it is not accessible to the shared pool. The user will then manually have to shrink the buffer cache and grow the shared pool to work around this problem.

Instead with automatic management, the DBA can simply set:

```
SGA_TARGET = 1G
```

In this case, if the application needs more shared pool memory for avoiding an ORA-4031 error condition, it will simply obtain that memory by acquiring it from the buffer cache.

Enhanced Performance

Besides maximizing the use of available memory, the Automatic Shared Memory Management feature can enhance workload performance as well. With manual configuration, it is possible that compiled SQL statements will frequently age out of the shared pool because of its inadequate size. This will manifest in terms of frequent hard parses and, hence, reduced performance.

However when automatic management is enabled, the internal tuning algorithm will monitor the performance of the workload and grow the shared pool if it determines that doing so will reduce the number of parses required. This is one of the most wonderful aspects of Automatic Shared Memory Management feature since it provides enhanced out-of-box performance, without requiring any additional resources or manual tuning effort.

Ease of Use

Having just a single parameter to deal with greatly simplifies the job of administrators. DBAs can now just specify the amount of SGA memory an instance has its disposal and forget about the rest. They do not need to figure out the sizes of individual components any more. In addition, they can be assured of the fact that no out of memory errors will be generated unless the system has truly run out of memory.

ENABLING AUTOMATIC SHARED MEMORY MANAGEMENT

The Automatic Shared Memory Management feature can be enabled either using EM or by setting the SGA_TARGET parameter.

When migrating from a manual scheme, it is best to tally the existing values of the SGA parameters and add a small amount (e.g. 16MB) to account for fixed SGA and internal overhead. At the same time the values of the automatically sized components can be removed from the parameter file.

For instance, when migrating from the following configuration:

```
SHARED_POOL_SIZE=256M
DB_CACHE_SIZE=512M
LARGE_POOL_SIZE=256M
LOG_BUFFER=16M
```

The above parameters can be replaced with

SGA_TARGET = 256 M + 512M + 256 M + 16M + 16 M (fixed SGA overhead) = 1056 M

Automatic Shared Memory Management may also be enabled dynamically. If you are using Enterprise Manager, you can enable SGA tuning by clicking the enable button on the Automatic Shared Memory Management screen.

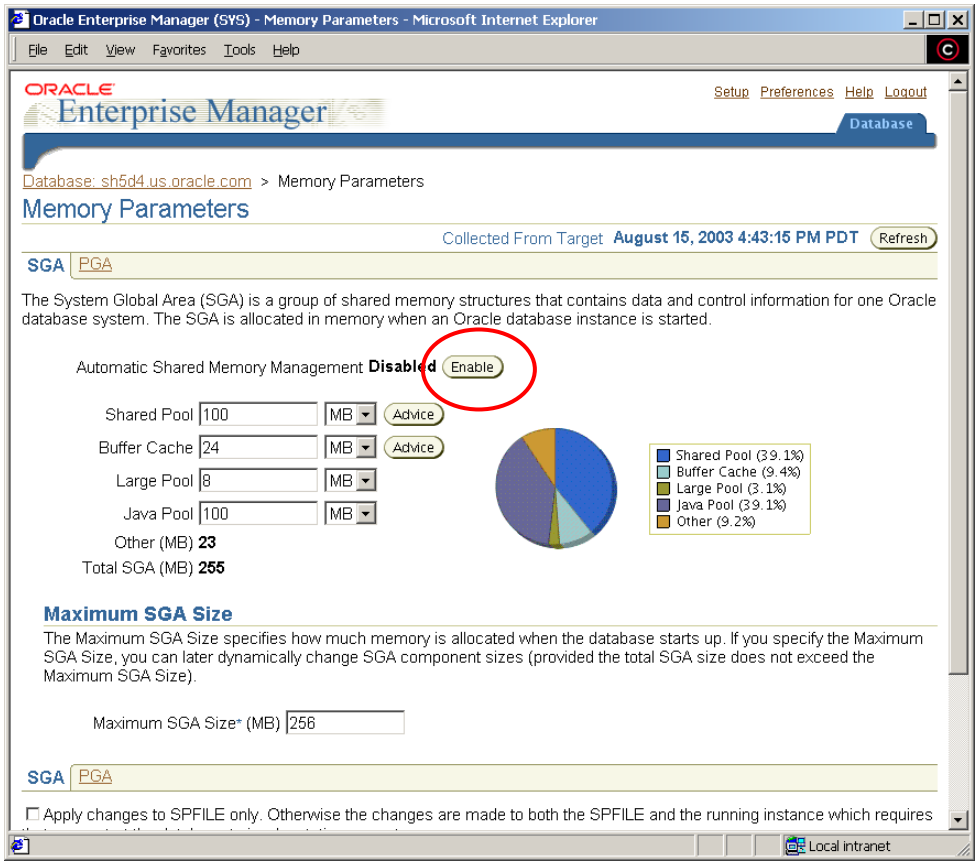

Fig 2: Enabling Automatic Shared Memory Management using Enterprise Manager

While enabling the Automatic Shared Memory Management feature using EM, the appropriate value for SGA_TARGET is automatically calculated according to the formula described above. In addition, EM also unsets all the parameters specifying the size of individual components in order to maximize the benefit of automatic management.

If using command line interface, the steps involved in enabling Automatic Shared Memory Management are as follows:

- Dynamically set SGA_TARGET to the current SGA size. The current size of the SGA can be determined from the fixed-view V$SGA via the following query:

```
select sum(value) from v$sga;
```

- Next dynamically set each of the auto-tuned component sizes to zero so that the automatic shared memory tuning algorithm can modify the sizes as needed.

If the above query for example returns the result of 536870912 (or 512M) then the steps for enabling auto SGA are as follows:

```
alter system set sga_target=512M;
alter system set db_cache_size = 0;
```

```
alter system set shared_pool_size = 0;
alter system set large_pool_size = 0;
alter system set java_pool_size = 0;
```

DYNAMIC MODIFICATION OF SGA PARAMETERS

Dynamic Modification of SGA_TARGET

The SGA_TARGET parameter is dynamic and can be increased up to the value specified by the parameter SGA_MAX_SIZE. The value of this parameter can also be reduced. In that case, one or more automatically tuned components are identified to release memory. The value of the SGA_TARGET parameter can be reduced until one or more auto-tuned components reach their minimum size.

The change in the amount of physical memory consumed when SGA_TARGET is modified depends on the OS platform. On some Unix platforms that do not support dynamic shared memory, the physical memory in use by the SGA is equal to the value of SGA_MAX_SIZE. On such platforms, there is no real benefit in running with a value of SGA_TARGET less than SGA_MAX_SIZE and setting SGA_MAX_SIZE on those platforms is, therefore, not recommended. On other platforms, such as Solaris and Windows, the physical memory consumed by the SGA is equal to the value of SGA_TARGET parameter.

Note that when SGA_TARGET is resized, the only components to be affected are the auto-tuned components. Any manually configured components remain unaffected.

For example, if we have an environment with the following configuration:

```
SGA_MAX_SIZE=1024M
SGA_TARGET = 512M
DB_8K_CACHE_SIZE = 128M
```

In this example, the value of SGA_TARGET can be resized up to 1024M and can also be lowered until one or more of the buffer cache, shared pool, large pool, or java pool reaches its minimum size (the exact value depends on environmental factors such as the number of CPUs on the system). But the value of DB_8K_CACHE_SIZE will remain fixed at all times at 128M.

Also, when SGA_TARGET is reduced, if the values for any auto-tuned component sizes have been specified to limit their minimum sizes, then those components will not shrink below their respective minimums. Therefore, if we have the following combination of parameters:

```
SGA_MAX_SIZE=1024M
SGA_TARGET = 512M
DB_CACHE_SIZE = 96M
DB_8K_CACHE_SIZE = 128M
```

In this example, in addition to the DB_8K_CACHE_SIZE being permanently fixed at 128M, the primary buffer cache will not shrink below 96M. This

imposes an additional restriction on how far the value of SGA_TARGET can be reduced.

Dynamic Modification of Parameters for Automatically Managed Components

When the parameter SGA_TARGET is not set, the rules governing resize for all SGA_TARGET component parameters are the same as in earlier releases. This is because in the absence of SGA_TARGET, the Automatic Shared Memory Management feature is disabled.

However, as mentioned earlier, when Automatic Shared Memory Management is enabled, the manually specified size of an automatically sized component (e.g. SHARED_POOL_SIZE), serves as a lower bound for the size of that component. It is possible to modify this limit dynamically by altering the value of the corresponding parameter.

If the specified lower limit for the size of a given SGA component is less than its current size, there is no immediate change in the size of that component. The value simply limits the auto-tuning algorithm to that reduced minimum size in the future.

For example, if:

```
SGA_TARGET = 512M,
SHARED_POOL_SIZE = 256M
(Current) Shared Pool size = 284M
```

In this example, dynamically resizing the SHARED_POOL_SIZE parameter down to 128M or lower has no effect on the current size of the shared pool.

Also note that setting the size of an automatically sized component to zero disables the enforcement of any user minimum on the size of the component. As stated earlier, this is the default behavior of automatically sized components when SGA_TARGET is set.

However, if the value of the parameter is raised to be greater than the current size of the component, the component will grow in response to the resize to accommodate the increased minimum. In the above example, if the value of SHARED_POOL_SIZE is resized up to 300M, then the shared pool will grow till it reaches 300M. This resize will happen at the expense of one or more auto-tuned components.

It is important to note that manually limiting the minimum size of one or more automatically sized components reduces the total amount of memory available for dynamic adjustment, thereby limiting the system's ability to adapt to workload changes. Consequently, the use of this option is not recommended barring exceptional cases. The default automatic management behavior has been designed to maximize both system performance and the use of available resources.

Modification of Parameters for Manually Sized Components

Parameters for manually sized components can be dynamically altered as well, the difference being that the value of the parameter always specifies the precise size of its corresponding component.

Therefore, if the size of a manual component is increased, extra memory is taken away from one or more automatically sized components. If the size of a manual component is decreased, the memory that is released is given to the automatically sized components.

For example:

```
SGA_TARGET = 512M
DB_8K_CACHE_SIZE=128M
```

In this case, increasing DB_8K_CACHE_SIZE to 144M (or by 16M) will mean that the 16M will be taken away from the automatically sized components. Likewise, shrinking DB_8K_CACHE_SIZE to 112 M (or by 16M) will mean that the 16M will be given to the automatically sized components.

PERSISTENCE OF AUTO TUNED VALUES

The sizes of the automatically tuned components are remembered across shutdowns if a server parameter file (SPFILE) is used. This means that the system will not need to learn the characteristics workload from scratch each time and will pick up where it left off from the last shutdown.

For this reason it is highly recommended that an SPFILE be used in conjunction with the Automatic Shared Memory Management feature.

CONCLUSIONS

Memory is a precious system resource and administrators currently spend a significant amount of their time optimizing its use. With Automatic Shared Memory Management, they are relieved of this time consuming and often tedious exercise. The flexibility and adaptiveness of this solution will ensure the best possible utilization of existing resources and thereby help organizations reduce capital expenditure. Just another example of how the Oracle Database 10g is going to let administrators play more strategic roles and allow businesses to become more profitable!

White Paper Title
Nov. 2003
Author:Tirthankar Lahiri, Arvind Nithrkashyap
Contributing Authors:Sushil Kumar, Brian Hirano, Kant Patel, Poojan Kumar

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

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

Oracle Corporation provides the software
that powers the internet.

Oracle is a registered trademark of Oracle Corporation. Various
product and service names referenced herein may be trademarks
of Oracle Corporation. All other product and service names
mentioned may be trademarks of their respective owners.

Copyright © 2002 Oracle Corporation
All rights reserved.