

**Database Tuning:
System Procedures and Methodologies
That Aid in Tuning your Oracle Database**

Justin Falk

Table of Contents

Abstract	Pg. 4
Introduction	Pg. 4
Tuning Early	Pg. 4
Setting Goals	Pg. 5
Prioritized Method	Pg. 5
Tuning Business Rules	Pg. 5
Tuning the Data Design	Pg. 5
Tuning the Application Design	Pg. 6
Tuning the Logical Structure	Pg. 6
Tuning the SQL	Pg. 7
Tuning the Access Paths	Pg. 8
Tuning the Memory	Pg. 8
Oracle Memory Structure	Pg. 8
System Global Area	Pg. 9
Database Buffer Cache	Pg. 9
The Shared Pool	Pg. 9
Redo Log Buffer	Pg. 10
Large Pool and Java Pool	Pg. 10
Program Global Area	Pg. 10
Automatic SQL execution memory management	Pg. 11
PGA Size and Query Efficiency	Pg. 11
Analyzing Memory Allocation	Pg. 11

SGA Allocation Check	Pg. 12
Determine PGA Usage	Pg. 13
Dynamically modifying the SGA	Pg. 14
SGA and PGA Advisories	Pg. 15
Oracle Enterprise Manager	Pg. 15
Memory Tuning Rules of Thumb	Pg. 17
Tuning the I/O and Physical Structure	Pg. 19
Tuning Resource Contention	Pg. 19
Tuning the Underlying Platform	Pg. 19
When to Stop Tuning?	Pg. 19
Summary	Pg. 20
Sources	Pg. 21

Abstract

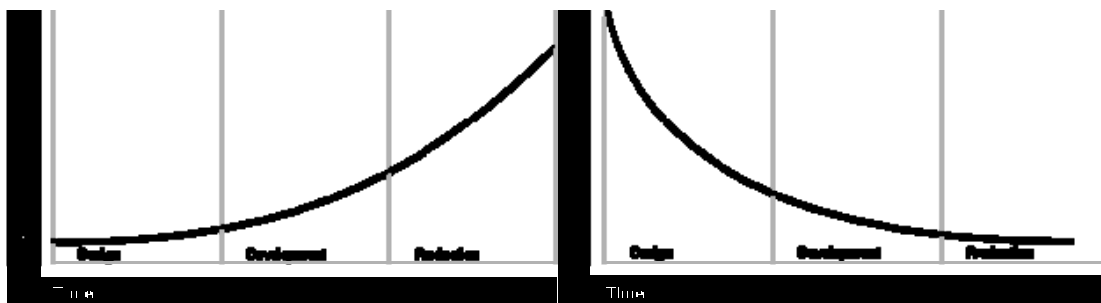
In this paper, many details of database tuning including goals, design, memory, and storage are discussed. The Oracle Guide ten step tuning process is explained and each step is briefly discussed with miscellaneous examples. The main focus of the paper is on memory tuning and its details. Generic terms and ideas are presented but most statements are implied to be used with an Oracle 9i or later system.

Introduction

Changing the specifications of any of the environment or database variables to make the database behave optimally is referred to as database tuning. It can also be defined as the activity of making a database application run more quickly (Shasha & Bonnet, 2006). Many tuning methods exist including design setup, parameter changes, SQL tweaking, application design, etc. Tuning should provide as much access as possible with the least amount of idle time. Increasing data throughput through disk arrays or internet bandwidth could be an example of tuning in the form of optimal throughput. Also increasing the system processor or memory speed could be an example of tuning through optimal response time. There are many ways to perform database tuning, and no particular way is always the best. Database tuning is dependent on the situation and environment. All factors should be taken into consideration when discussing tuning.

Tuning Early

In order to yield the best results it is best to start tuning before implementing your system. Careful thought should take place within the design phase when creating the database. Physical system hardware may have to accommodate for tuning needs such as a RAID or multiprocessor setup. Tuning after the design phase may cause costly upgrades as well as costly downtimes. A global perspective should be used to try to account for all aspects of the system before creating it.



Cost of Tuning Late vs. Cost of Tuning Early

Figure 1 (Oracle Corporation, 1997)

Setting Goals

At this time it is important to decide which tuning goals should be set. Definitive goals should be defined based on certain business procedures and objectives. Without setting goals it is difficult to determine how the system should be designed. Setting goals will also establish business processes if tuning is successful. Goals should be attainable, realistic, and quantitative. Always keep goals into consideration and disburse them amongst all members of the design team. It is imperative that other members understand how to meet these goals as certain sacrifices or improvements may need to take place in the design of the system.

Prioritized Method (10 Steps)

There are many methodologies to tuning, and as stated before no methodology is always better than the other in all situations. Some tuning processes frequently yield higher performance results than others but may not be effective in certain situations. Regardless of the method used a plan should be developed to determine where to start. Tuning results will vary based on the decision of how to tune. Rather than picking tuning methods randomly, use a planned approach. One thing commonly agreed upon is to start tuning the database using the most effective methods first. Methods that will yield the highest results should be implemented over methods that will not have as much impact. Although verbiage may differ slightly, a common 10 step method exists for tuning priority. This method is often called tuning by diminished returns, meaning the first steps will have higher results and later steps will yield a lower return. This is also a universally preferred method because previous steps may often effect tuning decisions in subsequent steps. Often an iterative approach is used in the 10 step method for optimal results, meaning the process is repeated from step 1 to step 10 multiple times possibly skipping or not skipping steps in between. The topics are described in order below.

Tuning Business Rules

Dependant on how rigid certain business rules are the database may be difficult to ideally setup. There may be physical limitations not known by the people designing the business rules that will cause unforeseen problems in design. Careful attention should be paid to the business rules and to keep them abstract. Less complicated rules allow for more freedom in the creation of the system as well as avoidance of unforeseen problems. More flexibility is given to system designers if there is less restriction as result of simple logical business rules. In certain instances business rules may need to be redesigned if that is possible to meet the expectations or goals of tuning. Without a middle ground or compromise systems may not perform as expected.

Tuning the Data Design

The data design consists of physical data types, their attributes, and their relations. Data design also consists of table types, summary values, and indexes. Careful thought

should be taken into how to structure the data design to match the business functions. Normalization is a key process necessary in almost every database to make it most efficient. Proper relationship and elimination of redundant data will help all tuning aspects. Indexing primary keys, or setting up index tables should be considered.

Tuning the Application Design

The primary function of retrieving, inserting, or updating data in a database is through some sort of an application. Business processes will determine this procedure but the application will determine the interaction with the database. A poorly tuned business application can potentially affect not just a few users but an entire business operation. You can gain quite a bit by tuning the application server based upon its specific business requirements and available resources. One example of application tuning would be by storing commonly used data within the application rather than having to look it up in the database for every transaction. For example, the discount rate for the day could be cached within application memory rather than retrieved for every sale. The application also can help database performance by validity checks of data such as incorrect length of a vehicle identification number or the capitalization of alphanumerics. The application could also check for data to correlate correctly to the business rules including data integrity. Memory leaks may need to be resolved within the application that cause a poor experience. Anything that can help relieve cost from the database in the application layer may want to be strongly considered and incorporated into its design.

Tuning the Logical Structure

Oracle's storage structure is made up of the following pieces. The first part of the structure is the data block that stores the data. Data blocks make up extents. An extent is created whenever a segment is created or if the current segment is too small to hold data that is being added. A collection of extents make up a single table. All tables are saved in a tablespace.

There are many things to consider when making sure your database is effective. Every piece of data in your database is stored in a tablespace. You should take actions to make them function properly. There are two different types of tablespaces, data dictionary managed ones and locally managed ones.

Data dictionary managed tablespaces have their information stored within the data dictionary. Every time an extent is allocated, information is updated in both the SYS free extents table and the used extents table. If the extents are allocated and deallocated frequently, it will be hard to keep up with the space usage. You can specify default storage parameters when creating data dictionary managed tablespaces. If the size that you set is too small for larger objects, the storage parameters may be altered. This is not true for locally managed tablespaces.

To reduce fragmentation and unnecessary activity, Oracle created the locally managed tablespace concept. This concept improves the performance. In this type of

tablespace, the extent information is stored in a bitmap in the data file. When the extents are allocated and deallocated, the space usage is tracked in the header. During the creation of a locally managed tablespace, you should choose an Extent Management clause. Autoallocate is the default selection. This allows the extent size to be automatically selected. This clause should be used if there will be different size objects in the tablespace. Uniform should be selected if you want control over the unused space and can pinpoint the amount of space that will be used for the objects.

Extent Management details how much data is stored before another extent is created. There are several storage parameters that can be set. The initial extent is how big the first segment's extent will be. The next extent is the second extent size. Minextents and maxextents are set to determine the number of extents that a segment can have.

Locally managed tablespaces also has a Segment space management clause. This clause lets you determine how free and used space in a segment is managed. There are two options Manual and Auto. Manual uses free lists to manage free space. Freelists contain the list of linked free blocks in a segment. The pctfree and pctused parameters determine how the available space is used. PCTfree sets the amount of blocks that are reserved for growth. The pctused determines when a block can be released back to the freelist. Coalescing is only used in dictionary managed tablespaces. This is the process of defragging free space in the tablespace. Automatic segment-space management is new in Oracle 9i. It manages space without freelists, pctfree and pctused. ASSM manages free space by using bitmaps to determine if there is more or less space available.

Another challenge that Database administrators face with storage tuning is that it is very hard to schedule time to take the database offline to perform maintenance. ASM or Automatic Storage Management is a new feature with Oracle 10g. ASM will automatically place the database files on different disk groups. The data is spread evenly across all of the storage resources and this improves performance of the database. ASM will also reconfigure the database files if storage resources are added or dropped. With this new tool, Administrators can change the storage configurations without taking the database offline.

Tuning the SQL

SQL is a language that allows for much flexibility in a database. There is often a multitude of ways a query can be written. Depending on the result needed some ways are significantly more efficient than others. Applications and interfaces dealing with the database may need to have SQL statements retuned for optimal performance. Dynamic Views (V\$) can be used such as V\$\$SQL to identify which queries are taking up the most CPU time or to identify how much memory is being used. There are many programs to help identify SQL inefficiencies such as Enterprise Manager Tuning Pack, IBM DBMon, and SQL Navigator. One should ask what the question really is and find alternative ways to structure the SQL to return the needed result back in a more efficient manner. Indexes or statements may need to be restructured. Some triggers may need to be disabled or

modified to avoid chaining or looping effects. Any queries causing locking or contention should be declared a problem query. There are also many packages available to find the root cause of the problem query. Simply restructuring the SQL statement may be easier to correct than tuning the access or data storage structure.

Tuning the Access Paths

Even with proper structure there must adequate access to the data. Without tuning access much time may be spent by users waiting for results. Access paths refer to the many ways data is found within the system. A common way to enhance this procedure is through adding indexes. Also, if not too costly, more time may be spent in the normalization of relational data. Hash clusters, B-tree indexes, range partitioning, list partitioning, and many other methods may be considered.

Tuning the Memory

Memory management is the key component to optimal performance of any computerized system of operations. Memory management controls the memory hierarchy of a system as a whole, or control of allocation of a fixed level within the memory hierarchy. Information stored within the system is shuttled between one realization of memory to another. The goal is to maintain maximum hit rate in each form of memory. This movement of information may be executed in several ways. A user action like copying a file from disk to memory in order to edit it is one way. Another way is that system software may transfer a page between swapping device and memory when a page fault occurs or system hardware may need a set of words from memory to cache when a word within the set is accessed.

This cardiograph dance of memory utilization procedures is done for the sole purpose of maximizing system processes that aid in quick execution and completion of application tasks. To maximize limited computer resources and maintain a reliable, healthy Oracle database system one must look into the various ways and tools available from both the Oracle manager counsel or thru Data Dictionary Views. It all starts with an understanding of the Oracle Memory Structure.

Oracle Memory Structure

Oracle uses a part of its memory allocation to hold both program code and data to make processing much faster than if it had to fetch data from the disk constantly. It is the use of the memory structure that enables Oracle to share executable code among several users, without having to go through all types of pre-execution processing every time a user invokes a piece of code. The Oracle server does not always write changes to the disk directly. It writes data changes to the memory area first. When it is ready it will write data to the disk making each session much faster because it only takes nanoseconds compared to the milliseconds it takes to write to a disk. With this process Oracle is able to overcome the I/O limitations of constantly writing to a disk system.

(SGA)System Global Area

The Oracle Memory Structure is composed of two main areas, the System global area and the Program global area. The System global area is the part of the total memory that all server processes shared. The SGA is the most important memory component in an Oracle instance. Its purpose is to speed up query performance and to manage high amounts of concurrent database activity. The SGA is not a one piece, but a combination of several memory structures. The main components of the SGA process are the database buffer cache, the shared pool, the redo log buffer, the java pool, and the optional large pool.

Database Buffer Cache

The database buffer cache refers to the memory buffers the database can use to hold the data that the server process reads from the data files on disk in response to a user's request. It is designed to hold the more frequently used data in the memory buffers. Otherwise, it would have to conduct I/O to and from disk storage. Oracle maintains a least recently used (LRU) list that contains a list of all free memory. The LRU algorithm ensures that only the most recently accessed data is retained in the buffer cache. When sizing the buffer cache it is very important for the proper performance of your database to keep these rules of thumb. The larger the size of the buffer cache, the fewer the number of disk reads and writes, thus creating a better performance of the database.

Oracle has the flexibility to use multiple buffer block sizes to data. This allows you to assign a specific buffer pool size for an object when you create the object. First you need to analyze the main buffer pool types and their size by viewing the initialization parameters of the keep buffer pool, recycle buffer pool, and the default buffer pool. The all important principle to keep in mind when tuning the buffer cache is to touch as few blocks as possible. An important indicator of performance is the buffer cache hit ratio, which measures the percentage of times a user access the data they need from the buffer cache. To calculate the hit rate of your instance use the following formula.

$$\text{Hit rate} = (1 - (\text{physical read})/(\text{logical reads})) * 100$$

The Shared Pool

The shared pool is another component of the SGA memory structure to consider when tuning your database. When a user sends SQL statements to the Oracle server for execution, the Oracle server parses the statement to check for validity of the statement. The Oracle server validates the existence of the objects being referenced in the SQL statement by reading the definitions of the objects from the data dictionary. Oracle also ensures that the operations being requested are valid for the objects being reference. All this statements and executes of various objects that it reads during parsing of SQL statements in the shared pool. Oracle uses a parsing tree and the execution plan from shared pool to execute the statement. This reduces the time required to execute the same

SQL statement subsequently, which provides increase performance. The shared pool consists of library cache and data dictionary cache.

Redo Log Buffer

The redo log buffer is another crucial component of the system global area. When a server process changes data in the data buffer cache via an insert, a delete, or an update, it generates redo data that is recorded in the redo log buffer. Proper sizing of the redo log buffer is important because it keeps Oracle from writing too often or too infrequently to the buffer. Use the `log_buffer` parameter to set the size of the redo log buffer, and it stays fixed for the duration of the instance.

Large Pool and Java Pool

The large pool is an optional memory pool, and Oracle manages it differently from the shared pool. A large pool is mostly used for accommodating the recovery manager operations. The value of this pool can be initialized in the initialization file by using the `large_pool_size` parameter and assigning a specific size. A large memory pool is important if one is using shared server architecture. The java pool is designed for a database that contains a lot of java code. Java pool memory is reserved for both Java Virtual Machine and the code for Java based applications. The default size for this memory pool is 20MB. In the case of Enterprise JavaBeans or CORBA applications, one could be looking at java pool sizes of greater than a gigabyte.

(PGA)Program Global Area

Of the two major components of the total system memory assigned to the Oracle database and its users, the SGA is the one that gets most of the attention and tuned component. But, the nature of the database transactions determines the size of the PGA relative to the SGA. Where most transactions are short, the PGA usage is very low. While complex queries, which are more typical of a distributed database system require a large amount of PGA. The PGA is a memory area that is allocated when Oracle starts a server process. Unlike the SGA, which is shared by multiple background processes, the PGA is used by only one process. Its memory structure is composed of the following areas. The private SQL area and the SQL work area.

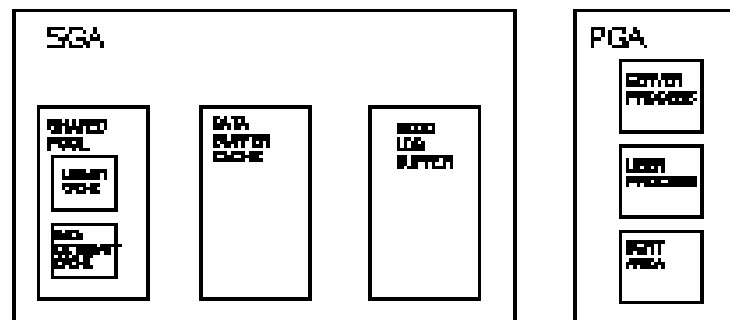


Figure 2 (SelfTest Software, 2003)

Automatic SQL Execution Memory Management

To automate the management of the PGA, you should first configure the initialization parameter *pga_aggregate_target*. Set the value to a predetermine amount that is big enough and available for future use. Also set the *workarea_size_policy* parameter to auto. Oracle server automatically allocates this memory among queries currently executing in the database. Oracle 9i provides the following guidelines to calculate the size of the *pga_aggregate_target* parameter. For an online transaction processing system (OLTP), Oracle recommends using 20 percent of the system memory allocated to Oracle for allocation to the *pga_aggregate_target* parameter. For a Decision Support System (DSS), Oracle recommends using 70 percent of the total memory allocation for Oracle, leaving 30 percent for the SGA. In a mixes environment Oracle recommends using 60 percent of the memory for SGA and the rest for PGA. Always try avoiding PGA targets that are too small, as the database will have to resort to a large number of multi pass sorts. You can to set the two initialization parameters to automate the management of the PGA by simply adjusting the following parameters.

```
Workarea_size_policy = AUTO
Pga_aggregate_targe = <X>k
```

To dynamically change the set target of the PGA use this syntax:

```
Sql> alter system set pga_aggregate_target = <X>m;
```

Where X is the amount of memory desired.

PGA Size and Query Efficiency

The purpose of PGA size and query efficiency is to reduce response time. Where all the sorts are performed in the PGA should be performed completely in the “cache” of the work area. Or at least maintain a one pass basis, and try to eliminate multiple passes over the same set of data. If you have a total system memory of 2GB and the table that needs to be sorted consists of 10GB worth of data, multiple passes over the same data will mean that your queries will take longer to complete.

Analyzing Memory Allocation

How do you find out and measure memory in the SGA and its various components? First you do a simple check of the SGA allocation by issuing a command the command *show sga*.

You will get the following listing.

Total System Global Area	135338868 bytes
Fixed Size	453492 bytes
Variable Size	109051904 bytes
Database Buffers	25165824 bytes
Redo Buffers	667648 bytes

Table 1.

SGA Allocation Check

Although this query gives you the total SGA allocation through the initialization parameters, you really don't get clear idea of the instance and database information that various database processes need or is currently distributing the memory among the various processes. To see the current usage of the SGA that has been allocated to your instance run the query V\$SGASTAT.

```
SQL> select * from v$sgastat
```

POOL	NAME	BYTES
	fixed_sga	453492
	buffer_cache	25165824
	log_buffer	656384
shared pool	errors	25144
shared pool	enqueue	171860
shared pool	KGK heap	3756
shared pool	KQR M PO	376332
shared pool	KQR S PO	61960
shared pool	KQR S SO	256
shared pool	sessions	410720
shared pool	sql area	1343452
shared pool	IM buffer	2098176
shared pool	KGLS heap	615756
shared pool	processes	144000
shared pool	free memory	46017524
shared pool	PL/SQL DIANA	528372
shared pool	FileOpenBlock	695504
shared pool	PL/SQL MPCODE	208068
shared pool	library cache	2126500
shared pool	miscellaneous	4174888
shared pool	MTTR advisory	12692
shared pool	PLS non-lib hp	2068
shared pool	joxs heap init	4220
shared pool	kgl simulator	565152
shared pool	sim memory hea	21164
shared pool	table definiti	168
shared pool	trigger defini	1700
shared pool	trigger inform	1044
shared pool	trigger source	160
shared pool	Checkpoint queue	282304
shared pool	VIRTUAL CIRCUITS	265160
shared pool	dictionary cache	1610880
shared pool	ksm_file2sga region	148652
shared pool	KSXR receive buffers	1033000
shared pool	character set object	274508
shared pool	FileIdentificatonBlock	323292
shared pool	KSXR large reply queue	166104
shared pool	message pool freequeue	834752
shared pool	KSXR pending messages que	841036
shared pool	event statistics per sess	1718360
shared pool	fixed allocation callback	180
large pool	free memory	8388608
java pool	free memory	33554432

43 rows selected.

Table 2.

As you can see the output gives us a more detail list of the SQL area, processes, sessions, library cache, dictionary cache, java pool, and the amount of free memory left in the shared pool. If this memory is consistently getting low, it indicates that the shared

memory pool size should be increased. But, if you notice that a little bit of memory is in the free pool, it is an indication that the shared memory pool size is appropriate for the instance.

Determine PGA Usage

To determine how the PGA memory is being used by various Oracle process all you need execute the V\$PROCESS view. These processes will include both the user sessions connected to the database and the Oracle background processes.

```
Sql> select
      program, pga_used_mem, pga_alloc_mem, pga_max_mem
    from
      v$process
   order by pga_used_mem desc;
```

PROGRAM	PGA_USED_MEM	PGA_ALLOC_MEM	PGA_MAX_MEM
ORACLE.EXE	4349880	5462848	5462848
ORACLE.EXE	1391680	1523476	1589012
ORACLE.EXE	442156	509800	509800
ORACLE.EXE	247952	349488	546096
ORACLE.EXE	186368	507052	507052
ORACLE.EXE	161208	224472	224472
ORACLE.EXE	144248	265500	207404
ORACLE.EXE	140876	207404	207404
ORACLE.EXE	136092	1256756	1256756
ORACLE.EXE	132564	198892	198892
ORACLE.EXE	56708	144936	144936
PSEUDO	0	0	0

13 rows selected.

Table 3.

To check how the PGA is being allocated by the instance, you need to query the V\$PGASTAT dynamic view. This view can help find out the aggregate pga_auto target, the current amount of PGA in use, and how the total PGA allocation stacks up against the estimated PGA needed for optimal and one pass execution of sort operations.

```
SQL> select * from v$pgastat;
```

NAME	VALUE UNIT
aggregate PGA target parameter	25165824 bytes
aggregate PGA auto target	16100352 bytes
global memory bound	1257472 bytes
total PGA inuse	7286784 bytes
total PGA allocated	10590208 bytes
maximum PGA allocated	21141504 bytes
total freeable PGA memory	0 bytes
PGA memory freed back to OS	0 bytes
total PGA used for auto workareas	0 bytes
maximum PGA used for auto workareas	527360 bytes
total PGA used for manual workareas	0 bytes
maximum PGA used for manual workareas	0 bytes
over allocation count	0
bytes processed	11882496 bytes
extra bytes read/written	0 bytes

cache hit percentage	100 percent
16 rows selected.	

Table 4.

The aggregate PGA auto target, which is 16MB, is determined by the `ini.ora` parameter `pga_aggregate_target`. This is the total memory that all Oracle connections can have in this instance. The total PGA in use component is about 7MB as compared to the total PGA allocated currently of about 10MB. The maximum PGA allocated value of about 21MB is the highest amount of PGA memory allocated at any time since the instances started. From these values one can see that the estimated PGA memory requirements for optimal and one pass operations are well below the `pga_aggregate_target`, indicating that the PGA memory is more than enough to prevent inefficient multi pass sorting of data in the PGA.

Dynamically Modifying the SGA

Oracle 9i has the capability to dynamically adjust most of the key components of the SGA. This means that you don't have to shutdown the database to modify the necessary initialization parameters, and changes can be done on the fly. With `alter` command we can change the parameters shown in the list below.

```
SQL> sho sga
```

Total System Global Area	135338868 bytes
Fixed Size	453492 bytes
Variable Size	109051904 bytes
Database Buffers	25165824 bytes
Redo Buffers	667648 bytes

Table 5.

```
SQL> alter system set db_cache_size = 100000;
System alter.
```

Total System Global Area	135338868 bytes
Fixed Size	453492 bytes
Variable Size	130023424 bytes
Database Buffers	4194304 bytes
Redo Buffers	667648 bytes

Table 6.

These procedures are of great benefit when you have peak time activities and wish to arrange for a larger amount of memory to be available to the instance during peak periods, without having to go into a lot of database performance tuning. If your monitor for the instance tells you that your hit ratio is below a critical threshold, you can simply increase the shared pool size to bring up these ratios.

Oracle 9i also gives us the ability to track the progress of an ongoing SGA dynamic resizing operation. By using the `V$SGA_CURRENT_RESIZE_OPS`, we can view the following columns like `initial_size`, `current_size`, and `target_size` which can help us see how the resizing operation is going.

SGA and PGA Advisories

Oracle 9i pro contains several wonderful tools call advisories. Advisories are like crystal balls that let you see into the future by conducting what if scenarios about performance levels. These comprehensive set of advisories for the key components of the SGA and PGA are V\$STATISTICS_LEVEL for various factors such as physical reads when memory changes. These simulated scenarios will allow you to pick a higher or lower allocation setting for the components of Oracle's memory. Oracle can load these advisories by default and the initialization parameter statistics_level must be set to typical for Oracle to populate the various dynamic views.

The buffer cache advisory uses the data dictionary view V\$DB_CACHE_ADVICE to give us the proper sizing of the buffer cache. Oracle takes the current workload and estimates through simulation how the cache hits would change if one was to change the database cache to a different size.

The shared pool advisory uses the V\$SHARED_POOL_ADVICE view to provide us with the information about the library cache component of the shared pool. The view will show simulation results of estimated parse time savings for different sizes of the shared pool. Oracle performs a simulation for 50% to 200% of the current size of the shared pool, thus giving us a reduction of a total processing time.

The PGA advisory contains two views the V\$PGA_TARGET_ADVICE and V\$PGA_TARGET_ADVICE_HS-TOGRAM. These views help us determine the ideal size for your PGA memory. In this advisory, Oracle simulates the workload history for several values of the *pga_aggregate_target* parameter.

Oracle Enterprise Manager

We can manage memory parameter with Oracle Enterprise Manager. Oracle provides a GUI console that is easy to use and manipulate. Instead of using the V\$DB_CACHE_ADVICE view and the buffer cache advisory, we can quickly get an estimate of ideal memory configuration through the OEM. When you open the OEM console, select Database and click the Instance icon. Next, choose the configuration option, and from here you can choose to modify memory and other database configuration parameters. In figure 3, we can see and adjust many of the SGA memory components.

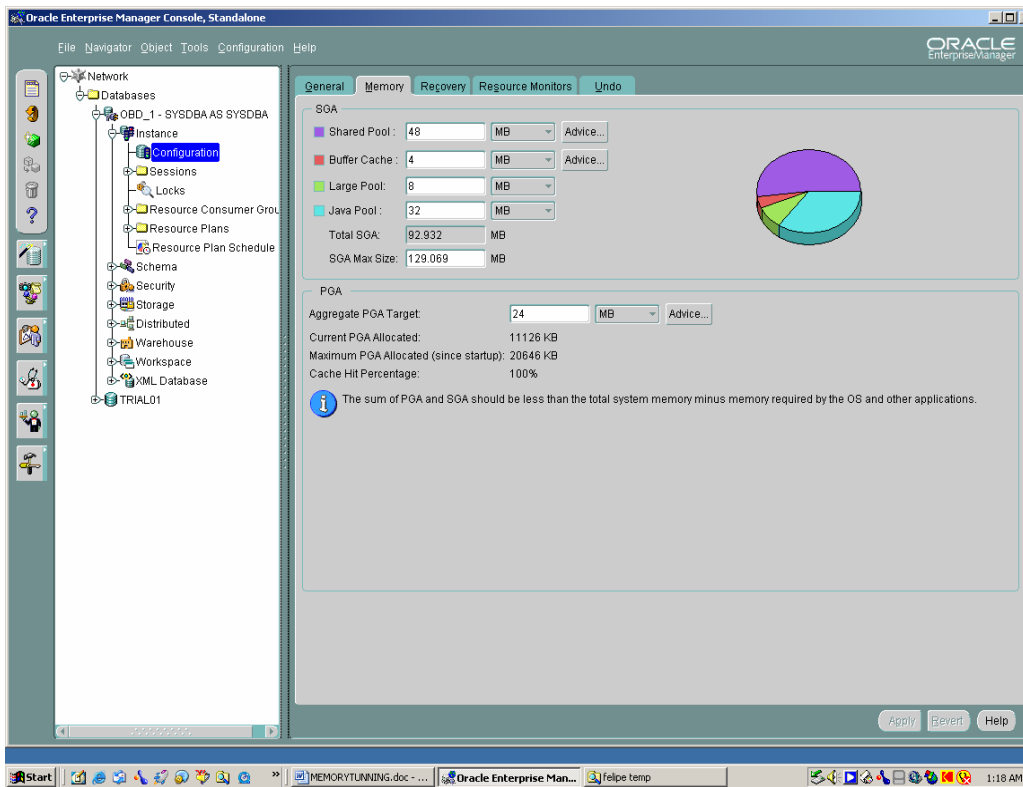


Figure 3.

Also look into the PGA aggregate target advice from the OEM console. From my PGA target advice window, you can see that my database does not have many transactions. But we can get to see the essential visuals from the graph and the table below it. In both the graph and the table we can gather the needed information to either increase the buffer size or get advice on the shared pool size.

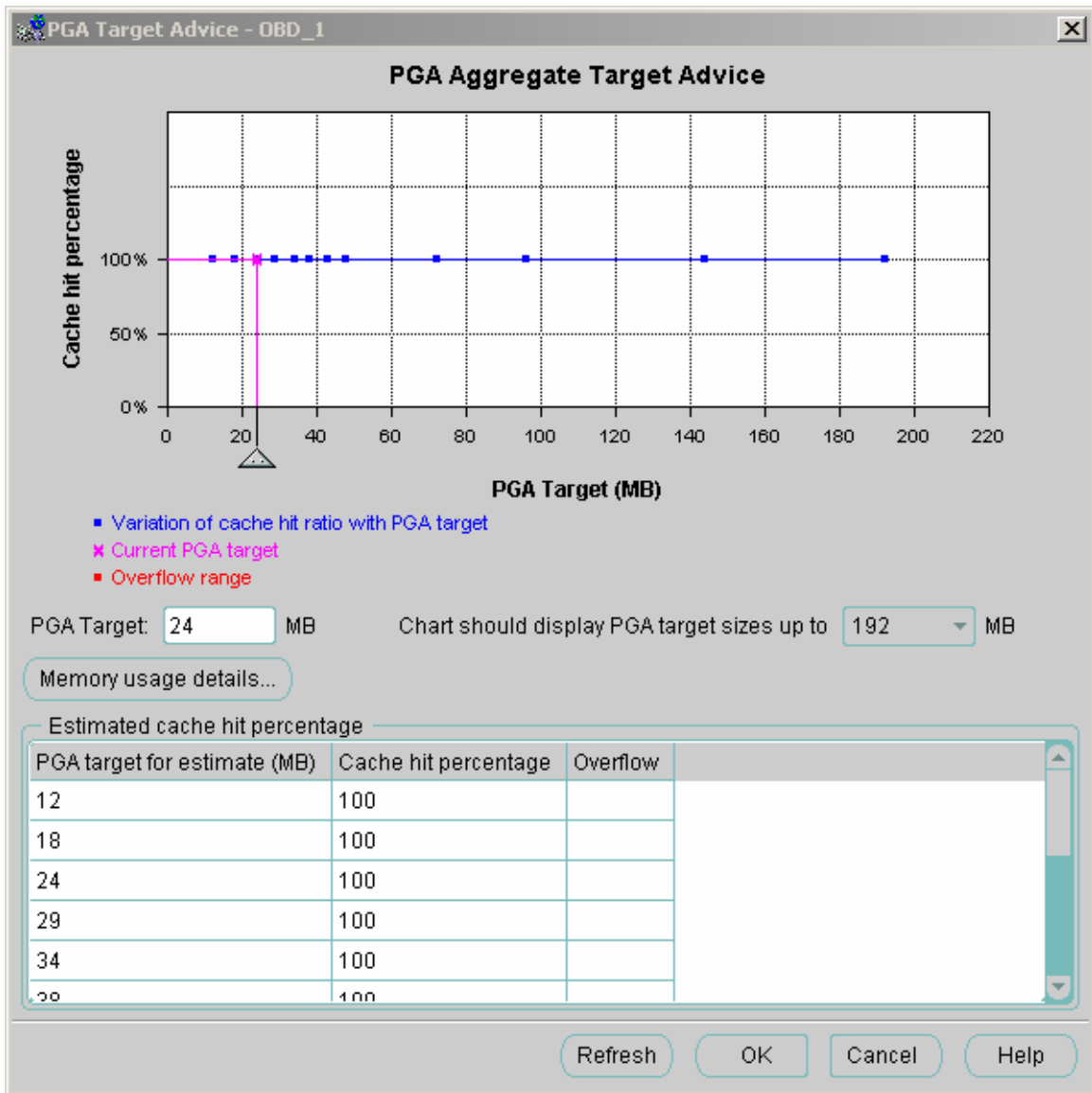


Figure 4.

Memory Tuning Rules of Thumb

The main goal of a DBA is to maintain high level of performance. To achieve this, the total available memory on a system should be configured in such a manner, that all components of the system function at optimum levels. In the general sense the following rule of thumb will list the various components in a system where Oracle is in the back end. The related components of the Operating System may include structures like the file system buffer cache. The User memory is the pool from which PGA areas will allocate server processes.

System Component	Allocated % of memory
Oracle SGA Component	~50%
Operating System + Related Components	~15%
User Memory	~35%

Table 7.

Oracle SGA Component	Allocated % of memory
Database Buffer Cache	~80%
Shared Pool Area	~12%
Fixed Size + Misc.	~1%
Redo Log Buffer	~0.1%

Table 8.

System Components	Allocated memory (in MB)
Oracle SGA Components	~1024
Operating System + Related Components	~306
User Memory	~692

Table 9.

Oracle SGA Component	Allocated memory (in MB)
Database Buffer Cache	~800
Shared Pool Area	~128-188
Fixed Size + Misc.	~8
Redo Log Buffer	~1 (average size is 512K)

Table 10.

The final configuration after multiple alterations of the tuning process may look different from the above figures. It should be kept in mind that these numbers are good start. Frequent monitoring of the various cache hit ratios and the ratio of sorts in memory versus sorts of disk will determine the final numbers for the above components.

Tuning the I/O and Physical Structure

The reading and writing (I/O) of data to hard disks often limits or hinders the performance speed of the database system. The speed at which data can be accessed on the hard disks is significantly slower than modern day system memory. Hard disks may be upgraded to faster ATA interface speeds or faster RPM (revolutions per minute) speeds. An alternative approach may be to use SCSI interface drives or a RAID array. Raid striping will often boost performance but raid mirroring will help aid in recovery. Highly accessed data may be spread across the physical structure for frequent use. This will help eliminate resource contention. Partitioning data based on business units across multiple tablespaces will also help increase speed and avoid contention.

Tuning Resource Contention

Resource contention refers to the priority given to user or oracle processes. Many times there is competition in the contention for processes. As any system becomes larger there will be more frequent occurrences of resource contention. Simultaneous processing by multiple users will cause instances in which contention should be considered. A common contention most are familiar with is gridlock contention causes by more than one user performing data manipulation. Other common areas of contention that may need to be considered are latch contention, share pool contention, and block contention.

Tuning the Underlying Platform

Every system has an underlying platform on which the system resides. This could be the design of the server or its operating system. It could also involve physical allocation of resources. Upgrades or simple reallocation of memory to certain processes are examples of tuning the underlying platform. A Windows specific example would be changing the amount of system memory given to outside applications. A Unix specific example would be changing the size of its buffer cache. Another example may be a physical memory swap to a higher frequency memory or switching over system memory to a higher bus speed.

When to Stop Tuning?

With the initial goals of the system in mind carefully designed tests should be implemented. Tests should be simple and repeatable. Measuring the test results should be easy and quick. Easier repeatable tests will prove more efficient in determining the success of tuning on multiple occasions. Depending on the results of the tests, conclusions much like hypothesis may be made on tuning effectiveness. Certain processes may be ruled a success and others ruled out as inefficient. Decisions can be made based on the results if further using the process is required. In certain cases there may be a negative impact worsening system performance. Certain tuning processes may need to be reversed if ill-advised effects or results are visible. Records should be kept as to rule out chance happenings of problems. Reading records will discern effectiveness of

a process. Records will also provide a way for quantifying process results especially measuring its cost effectiveness. Small results may not be worth the cost of high costs and vice versa.

Care must be taken with repeatedly performing tests due to frequent caching of memory. After tuning is determined effective be careful not to jump to conclusions. The tuning process may only be temporary or particular to one situation. Do not fully implement the process across the whole database before carefully considering its consequences. Also, the acknowledgement by another administrator may help avoid illogical decisions or confirm logical ones. Do not be hasty as the negative effects may outweigh the positive ones. When the goals set forth prior to tuning are met it is best to stop tuning. Although possible further tuning could take place it will typically not be worth the trouble compared to the time spent. If the end user criteria or objectives are satisfied it is often best to document it and explain the process. Later if further tuning is desired there is reference that will help determine how to start. The process may later be extrapolated for further enhancement or completely avoided if not related to the problem.

Summary

There are a multitude of ways tuning can be performed on your database system. It is best to begin tuning early if not before the system is implemented. Certain methods may want to be explored over others based on cost, downtime, complexity, and diminished returns. There is no particular priority that is always better than others in all situations. Global effects of tuning changes should be considered before implementing into the production environment. The business processes should closely match to the design of the system. Ideal results will be considered optimal based on properly preset and measurable goals for tuning the system. Only proper tuning will provide the most advantageous system possible for a group of selected needs.

SOURCES

Alapati, R. Sam, (2003). Expert Oracle9i Database Administration, The Author's Press, p240-p245.

Dictionary of computing 3rd edition, 1991, Oxford University Press, Walton Street, Oxford New York

Oracle Corporation, Performance Tuning Method, Retrieved November 21, 2006 from <http://www.thinkunix.net/unix/db/oracle/docs-7.3/DOC/server/doc/A48506/method.htm>

Self Test Software, Oracle 1Z0-031 study guide. A Kaplan Professional Company, 2003. Retrieved November 21, 2006 from www.selftestsoftware.com

Shasha, B. & Bonnet, P. Database Tuning, Principles, Experiments and Troubleshooting Techniques, Retrieved November 11, 2006 from <http://www.mkp.com/dbtune>

Stallings, William, Computer Organization and Architecture, Designing for Performance, 1996. (4th ed.)
Prentice Hall, Upper Saddle River, New Jersey. p240-p245.

Niemiec, R., TUSC, Tuning for the advanced DBAs, Retrieved November 21, 2006 from <http://www.db-system.com/Wtun300.htm>

Theriault, M., Carmichael, R., & Viscusi, J. Oracle 9i DBA 101, Learn the essentials of Oracle Database Administration, McGraw Hill Companies 2002