第 5 章　内存管理

Oracle 数据库实例启动时，就需要分配共享内存，启动后台进程，如何分配和设置共享内存参数，对于 Oracle 来说是非常重要的。不当的内存分配轻则影响性能，重则导致数据库故障，在生产实际中不容忽视。

Oracle 数据库所使用的内存主要涉及两个方面：PGA 和 SGA。本章就 Oracle 的内存管理问题进行探讨。

5.1　PGA 管理

PGA 指的是程序全局区（Program Global Area），是服务器进程（Server Process）使用的一块包含数据和控制信息的内存区域，PGA 是非共享的内存，在服务器进程启动或创建时分配（在系统运行时，排序、连接等操作也可能需要进一步的 PGA 分配），并为 Server Process 排他访问，所以 PGA 中的数据结构并不需要通过 Latch 来保护。

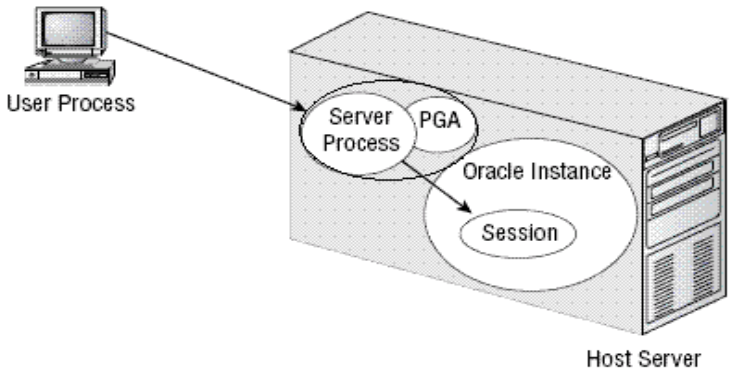
5.2.1 什么是 PGA

进程的创建通常有两种模式：专用服务器模式（Dedicated Server）及共享服务器模式（Shared Server）。在专用服务器模式下，Oracle 会为每个会话启动一个 Oracle 进程；而在共享服务器模式下，通常在服务器端启动一定数量的服务器进程，然后由多个客户端请求共享同一个 Oracle 服务进程。通常数据库都应当运行在专用服务器模式下。PGA 的内容依专用模式和共享服务器模式而有所不同，但是通常来说，PGA 中包含私有 SQL 区(存放绑定信息、运行时内存结构等)、Session 信息等内容。

从内存分配与使用上 PGA 可以被区分为两个区域：

- 固定 PGA（Fixed PGA）- 固定 PGA 和固定 SGA 类似，包含了大量原子变量、小的数据结构和指向可变 PGA 的指针，这些变量在源码中定义，在编译时分配，可以被认为是 PGA 的保留内存。
- 可变 PGA（Variable PGA）- 可变 PGA 通过具体的内存 Heap 分配来实现，其空间分配与使用时可以变化的，通过内部视图 X$KSMPP（[K]ernel [S]ervice [M]emory [P]GA hea[P]）可以查询可变 PGA 内存的分配和使用情况。PGA 的可变区中主要包含会话内存及私有 SQL 区等。

下图简要说明了 PGA 的创建过程，当客户端向服务器发送连接请求，服务器监听到客户端请求，在专用服务器模式下，会在服务器端衍生一个 Server Process 来代理用户的请求，服务器进程进而向实例发起连接，创建会话（CREATE SESSION），而 PGA 就为 Server Proces 所分配和使用：



可变 PGA 部分实际上是我们最为关注的 PGA 部分。虽然 PGA 的内容对于专用和共享模式会有所不同，但是通常来说，可变 PGA 又进一步的由以下两部分组成：

- 会话内存 - Session Memory：用于存放会话的登录信息以及其他相关信息，对于共享服务器模式，这部分内存是共享而非私有的。
- 私有的 SQL 区 - Private SQL Area：Private SQL Area 包含绑定变量信息、查询执行状态信息以及查询工作区等。每个发出 SQL 查询的会话都拥有一块私有 SQL 区，对于专用服务器模式，这部分内存在 PGA 中分配，对于共享服务器模式，这部分内存在 SGA 中分配。

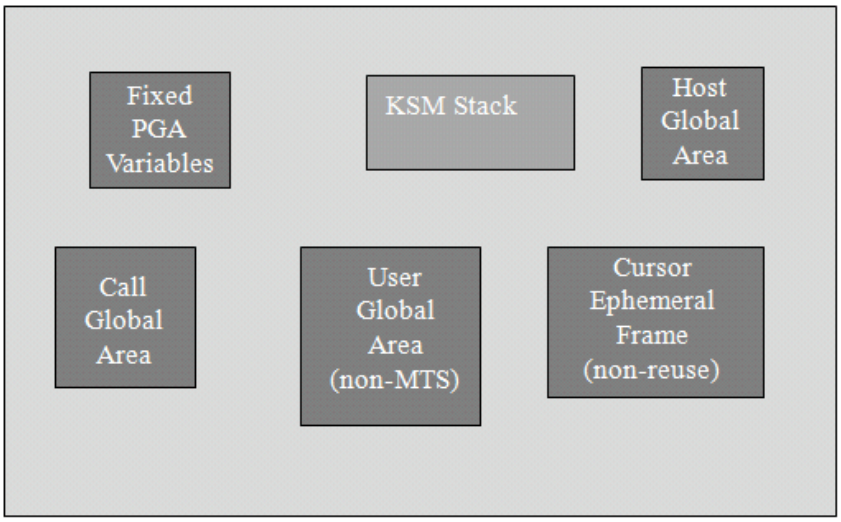
在这里还需要了解的一个概念是游标（Cursor）。Oracle 的应用程序或用户的应用程序执行时，都可能显示或隐式的打开游标（Open Cursor）来进行任务处理，打开游标就需要分配 SQL Area。管理私有 SQL 区是用户进程的责任，而分配和回收则依赖于具体的应用程序，为了防止过度的 SQL 工作区分配，Oracle 通过 OPEN_CURSORS 参数来限制每个用户进程能够同时打开的游标数量。一个私有 SQL 区在 Cursor 打开时分配，当执行结束游标关闭时释放。

简单来说，用户进程的任务执行以及 Cursor 的使用是 PGA 内存的主要消耗者，也是我们在进行数据库性能优化时最关心的内容，实际上**数据库的活动主要就是 Cursor 的活动**。

进一步的，私有 SQL 区又由以下两部分组成：

- 永久区域 - Persistent Area：这个区域包含绑定变量等信息，这部分内存只有在游标被关闭时才会被释放。
- 运行时区域 – Runtime Area：这个区域存放了 SQL 语句运行时所需要的信息，在执行请求时首先创建，其中包含了查询执行的状态信息（如对于全表扫描，则记录全表扫描的进度等）、SQL work areas（这部分区域在内存密集型请求下分配，如 Sort 或者 Hash-Join 等，对于 DML 语句来说，SQL 语句执行完毕就释放该区域，对于查询语句则是在记录返回后或查询取消时释放）

下图简要说明了 PGA 的整体结构，图示中包含了固有 SGA 部分，也包含了游标运行时示意：



5.2.2 UGA 与 CGA

在上图 PGA 的介绍中，注意到存在一块成为 UGA（User Global Area -用户全局区）的内存区域，这也是可以经常见到的一个名词。UGA 由用户会话数据、游标状态和索引区组成。在共享服务器模式下，一个共享服务进程被多个用户进程共享，此时 UGA 是 Shared Pool 或 Large Pool 的一部分，而在专用服务器模式下，UGA 则是 PGA 的一部分。

不考虑 Shared Server 模式，在 Dedicated 模式下，PGA 与 UGA 关系，就如同 Process 和 Session 的关系，PGA 是服务于进程的内存结构，包含进程信息；而 UGA 是服务于会话的，它包含的是会话的信息。UGA 中包含如下信息：

- 打开游标的永久区和运行区；
- 包的状态信息以及变量信息；
- Java 会话的状态信息；
- 启用角色信息、跟踪事件；
- 起作用的 NLS 参数；
- 所有打开的 database links；
- 会话访问控制信息等

和 PGA 一样，UGA 也由两组区组成，固定 UGA 和可变 UGA（或者说 UGA 堆）。固定 UGA 包含了大概 70 个原子变量、小的数据结构以及指向 UGA 堆的指针。

UGA 中的内存分配可以通过内部表 X$KSMUP（X$KSMUP - [K]ernel [S]ervice [M]emory [U]GA Hea[P]）查询得到。UGA 堆包含了存储一些固定表（X$表）的永久内存（依赖于特定参数的设置，如 OPEN_CURSORS，OPEN_LINKS 和 MAX_ENABLED_ROLES）。除此以外，大部分的 UGA 用于私有 SQL 区和 PL/SQL 区。

从 Oracle9iR2 开始，有一系列新的隐含参数被引入用于控制自动的 PGA 管理，这其中有一个关键的参数是_use_realfree_heap，当设置这个参数为 true 时，Oracle 会为 CGA、UGA 单独分配堆，而不从 PGA 中分配。它的默认值为 false，而当设置了 pga_aggregate_target 后，它

的值自动被改为 true：

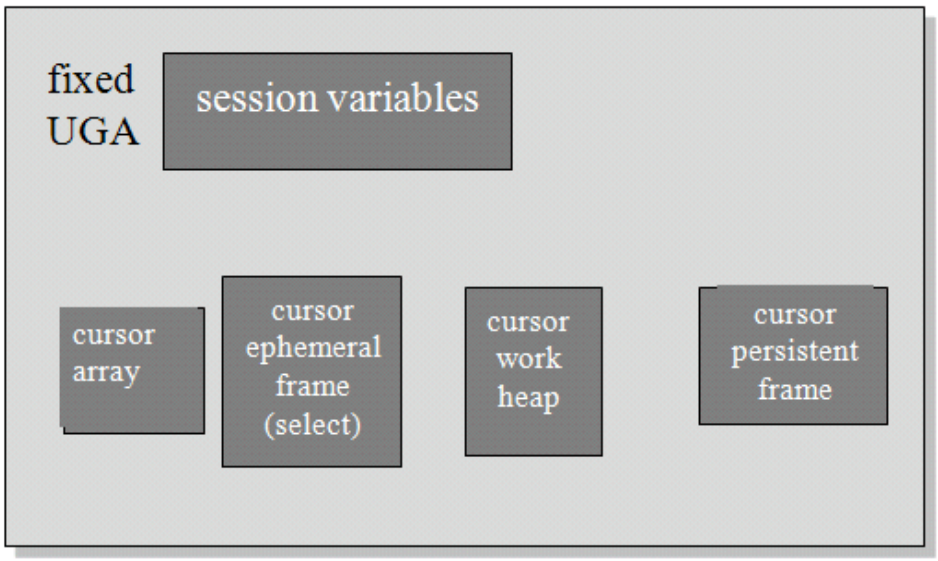
```
SQL> SELECT x.ksppinm NAME, y.ksppstvl VALUE, x.ksppdesc describ
  2    FROM SYS.x$ksppi x, SYS.x$ksppcv y
  3   WHERE x.indx = y.indx AND x.ksppinm LIKE '%&par%';
Enter value for par: realfree
NAME                           VALUE    DESCRIB
------------------------------ -------- ------------------------------------------------
_realfree_heap_max_size        32768    minimum max total heap size, in Kbytes
_realfree_heap_free_threshold  4194303  threshold for performing real-free, in Kbytes
_realfree_heap_mode            0        mode flags for real-free heap
_use_realfree_heap             TRUE     use real-free based allocator for PGA memory
```

_use_realfree_heap 是自动管理 PGA 技术的关键技术变化，realfree 代表着实时释放。Oracle9i 之前手工管理的 PGA 的主要问题在于，UGA 缺省的在 PGA 中分配，当会话执行了诸如排序、HASH-JOIN 等操作，耗用了大量 PGA 内存，而当会话执行完毕之后，内存会释放给 PGA 而不是 OS，在很多时候这会导致过度的 PGA 内存使用（在以前版本 PGA 内存分配和回收是通过 malloc()以及 brk()调用来完成的）；从 Oracle9iR2 开始，自动的 PGA 内存管理当_use_realfree_heap 为 true 时，PGA 的内存分配将会通过 mmap()调用来实现，这样当调用结束时将不必将内存返回给进程而直接返回给 OS，从而实现了更好的 PGA 内存分配与使用。

通过 V$PGASTAT 视图可以查询 PGA 累计释放回 OS 的内存空间：

```
SQL> select name,value from v$pgastat where name like '%OS';
NAME                                                             VALUE
---------------------------------------- ----------------------
PGA memory freed back to OS                              39824916480
```

下图是 UGA 的结构示意图：



在 PGA 的示意图中，还涉及了另外一块内存区域被称为 CGA（Call Global Area）-调用

全局区。与其他的全局区不同，CGA 的存在是瞬间的，只存在于调用过程中，而且无论 UGA 存在于 PGA 还是 SGA，CGA 都是 PGA 的 SubHeap。对于实例的一些低层次的调用（Low-Level Call）需要 CGA，包括分析 SQL 语句、执行 SQL 语句以及获取查询结果都需要使用 CGA，在 SQL 执行过程中的每个递归调用需要一个独立的 CGA，在 SQL 的解析过程中，查询数据字典信息、对 SQL 进行语法以及语义的解析、SQL 的优化以及不同执行计划的评估都需要使用 CGA。

当然，调用并不是只通过 CGA 中的数据结构来工作，实际上调用所需要的大部分的重要数据结构都来自于 UGA（如 SQL AREA, PL/SQL AREA，Sort Area 都存放在 UGA 中，因为这些结构在调用期间需要一直可用），CGA 中只包含了那些调用结束后可以被释放的数据。例如，CGA 中包含了 Direct I/O BUFFER、递归调用信息、表达式评估的堆栈信息等，此外 Java 调用内存也在 CGA 中分配。

5.2.3 PGA 管理技术的变迁

在 Oracle 9i 以前的版本中，PGA 由一系列的内存区域组成，这些区域包括主要由 *_area_size 参数控制。在 Oracle8i 的环境中，这些参数主要有：sort_area_size、hash_area_size、bitmap_merge_size、create_bitmap_area_size。

可以从数据库中得到这些参数设置的当前值，在 Oracle8i 中可以通过手工修改 sort_area_size、hash_area_size 等参数值来控制 PGA 的使用：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle8i Enterprise Edition Release 8.1.7.4.0 - 64bit Production
SQL> show parameter area_size
NAME                                 TYPE    VALUE
------------------------------------ ------- ------------------------------
bitmap_merge_area_size               integer 1048576
create_bitmap_area_size              integer 8388608
hash_area_size                       integer 131072
sort_area_size                       integer 65536
```

这种独立管理的方式存在一个极大的弊端，以 SORT 操作为例，如果我们为了使特定的排序操作能够在内存中完成，可能需要设置较大的 sort_area_size，但是由于进程的独立 PGA 内存难于回收和共享，这样可能导致过度的 PGA 内存消耗，所以合理设置和调整 PGA 在 Oracle9i 之前是一件比较复杂的事情；从 Oracle9i 开始，Oracle 提供了一种新的 PGA 内存管理方法：自动化 SQL 执行内存管理（Automated SQL Execution Memory Management），也称为自动 PGA 管理，使用这个新特性，Oracle 可以在一个总体 PGA 使用限制下自动管理和调整 SQL 内存区，从而大大简化了 DBA 的工作，同时也提高了数据库的性能。

为实现自动的 PGA 管理，Oracle 引入了几个新的初始化参数：

◆ PGA_AGGREGATE_TARGET-此参数用来指定所有 session 总计可以使用最大 PGA

内存。这个参数可以被动态的更改,取值范围从 10M ～（4096G-1 ）bytes。

- WORKAREA_SIZE_POLICY-此参数用于开关 PGA 内存自动管理功能，该参数有两个选项：AUTO 和 MANUAL，当设置为 AUTO 时，数据库使用自动 PGA 管理功能，当设置为 MANUAL 时，则仍然使用之前手工管理的方式。

缺省的，WORKAREA_SIZE_POLICY 参数被设置为 AUTO。

此外需要注意的是，在不同版本中，自动 PGA 管理的范畴不同：

- 在 Oracle9i 中，PGA_AGGREGATE_TARGET 参数仅对专用服务器模式下（Dedicated Server）的专属连接有效，但是对共享服务器（Shared Server）连接无效
- 从 Oracle10g 开始 PGA_AGGREGATE_TARGET 对专用服务器连接和共享服务器连接同时生效

5.2.4 参数的设置与内存分配

PGA_AGGREGATE_TARGET （通常可以缩写为 P_A_T）参数同时限制全局 PGA 分配和私有工作区内存分配。

在 Oracle9i 以及 Oracle10gR1 中，单个 SQL 操作内存使用存在如下限制：

1．对于串行操作，单个 SQL 操作能够使用的 PGA 内存按照以下原则分配：

*MIN（5%*PGA_AGGREGATE_TARGET，100MB）*

此处 5%*P_A_T 实际上是由一个内部参数_smm_max_size 决定的，该参数限制自动工作区模式下最大的工作区使用（maximum work area size in auto mode -serial）

2．对于并行操作

30% PGA_AGGREGATE_TARGET /DOP （DOP=Degree Of Parallelism 并行度）

对于 Oracle10gr2 以及 Oracle11g 存在如下限制：

1.对于串行操作，单个 SQL 操作能够使用的 PGA 内存按照以下原则分配：

如果 P_A_T <= 500MB，则_smm_max_size = 20%*P_A_T

如果 P_A_T 在 500MB 和 1000MB 之间，_smm_max_size = 100M

如果 P_A_T 介于 1001MB 和 2560MB (2.5GB)之间，_smm_max_size = 10%* P_A_T

如果 P_A_T > 2560MB (2,5GB)则 _smm_max_size = 262,060 MB (~0,25GB)

2．对于并行操作

50% PGA_AGGREGATE_TARGET /DOP （DOP=Degree Of Parallelism 并行度）

但是注意，当 DOP <=5 时，_smm_max_size 限制生效，并行度超过 5 时另外一个限制并行的参数 _smm_px_max_size 才会生效。

从 Oracle10g 开始的新 PGA 管理算法受一个新增的隐含参数_newsort_enabled 影响，如果将该参数设置为 False，则数据库会使用之前 Oracle9iR2 中的算法规则：

```
SQL> @GetHidPar
Enter value for name: newsort_enabled
old   4:    AND x.ksppinm LIKE '%&name%'
```

```
new   4:    AND x.ksppinm LIKE '%newsort_enabled%'
NAME                                               VALUE
-------------------------------------------------- --------------------
_newsort_enabled                                   TRUE
```

要理解 PGA 的自动调整，还需要区分可调整内存（TUNABLE MEMORY SIZE）与不可调整内存（UNTUNABLE MEMORY SIZE）。可调整内存是由 SQL 工作区使用的，其余部分是不可调整内存。

启用了自动 PGA 调整之后， Oracle 仍然需要遵循以下原则：

UNTUNABLE MEMORY SIZE + TUNABLE MEMORY SIZE <= PGA_AGGREGATE_TARGET

数据库系统只能控制可调整部分的内存分配，如果可调整的部分过小，则 Oracle 永远也不会强制启用这个等式。

另外，PGA_AGGREGATE_TARGET 参数在 CBO 优化器模式下，对于 SQL 的执行计划会产生影响。Oracle 在评估执行计划时会根据 PGA_AGGREGATE_TARGET 参数评估在 Sort，HASH-JOIN 或 Bitmap 操作时能够使用的最大或最小内存，从而选择最优的执行计划。

对于 PGA_AGGREGATE_TARGET 参数的设置，Oracle 提供这样一个建议方案

1．对于 OLTP 系统

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

2．对于 DSS 系统

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

也就是说，对于一个单纯的数据库服务器，通常我们需要保留 20%的物理内存给操作系统使用，剩余 80%可以分配给 Oracle 使用。Oracle 使用的内存分为两部分 SGA 和 PGA，那么 PGA 可以占用 Oracle 消耗总内存的 20%（OLTP 系统）至 50%（DSS 系统）

> 注意：在某些 os 上单个进程使用的真实内存可能远大于在 Oracle 中看到的 PGA 大小，如 AIX。在 AIX 上通常建议 Oracle 使用内存不超过物理内存的 70%。

这只是一个建议设置，更进一步的应该根据数据库的具体性能指标来调整和优化 PGA 的使用。伴随这个新特性的引入 V$PROCESS 视图增加了相应字段用来记录进程的 PGA 耗用，选择一个 Oracle 用户进程：

```
SQL> ! ps -ef|grep LOCAL|head -1
oracle    2803     1  0 Jul13 ?        00:00:46 oracleeygle (LOCAL=NO)
```

其相关的 PGA 使用现在可以从 v$process 视图获得：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Prod
SQL> select pid,spid,username,pga_used_mem,pga_alloc_mem,pga_freeable_mem,pga_max_mem
  2  from v$process where spid=2803;
 PID SPID         USERNAME   PGA_USED_MEM PGA_ALLOC_MEM PGA_FREEABLE_MEM PGA_MAX_MEM
---- ------------ ---------- ------------ ------------- ---------------- -----------
```

```
  18 2803          oracle          6037917       7217777           917504     12460657
```

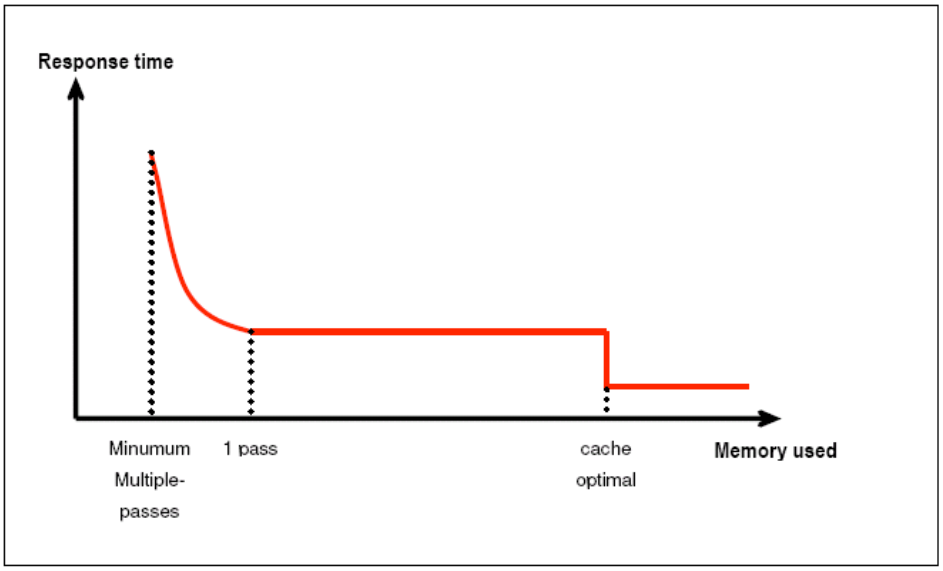
而通过 V$PROCESS_MEMORY 视图还可以进一步知道 PGA 内存消耗在什么地方：

```
SQL> SELECT p.program,p.spid,pm.category,pm.allocated,pm.used,pm.max_allocated
  2  FROM V$PROCESS p, V$PROCESS_MEMORY pm
  3  WHERE p.pid = pm.pid AND p.spid = 2803;
PROGRAM             SPID    CATEGORY         ALLOCATED       USED MAX_ALLOCATED
------------ ------ ------- ----------       ---------- ---------- -------------
oracleeygle@eygle   2803    SQL                137208      47028       1679336
oracleeygle@eygle   2803    PL/SQL              98528      33528        123196
oracleeygle@eygle   2803    Freeable           917504          0
oracleeygle@eygle   2803    Other             6064537                   9740621
```

SQL 在工作区中以三种方式执行：

- 优化方式（Optimal）-指所有处理可以在内存中完成
- onepass - 大部分操作可以在内存中完成，但是需要使用到磁盘操作
- multipass - 大量操作需要产生磁盘交互，性能极差

下图显示了在不同方式下响应时间与内存分配曲线：



通常我们对于 PGA 的优化目标就是使得 Optimal 的执行尽量高，也就是尽量在内存中完成所有排序等操作；同时使 multipass 操作尽量低，也就是要使磁盘交互尽量低。

工作区性能期望实现如下目标：

workarea execution - optimal >= 90%

workarea execution - multipass = 0%

以下是一个生产系统的 PGA 性能指标：

```
SQL> SELECT NAME, VALUE,
  2          100 * (  VALUE
  3           / DECODE ((SELECT SUM (VALUE) FROM v$sysstat
```

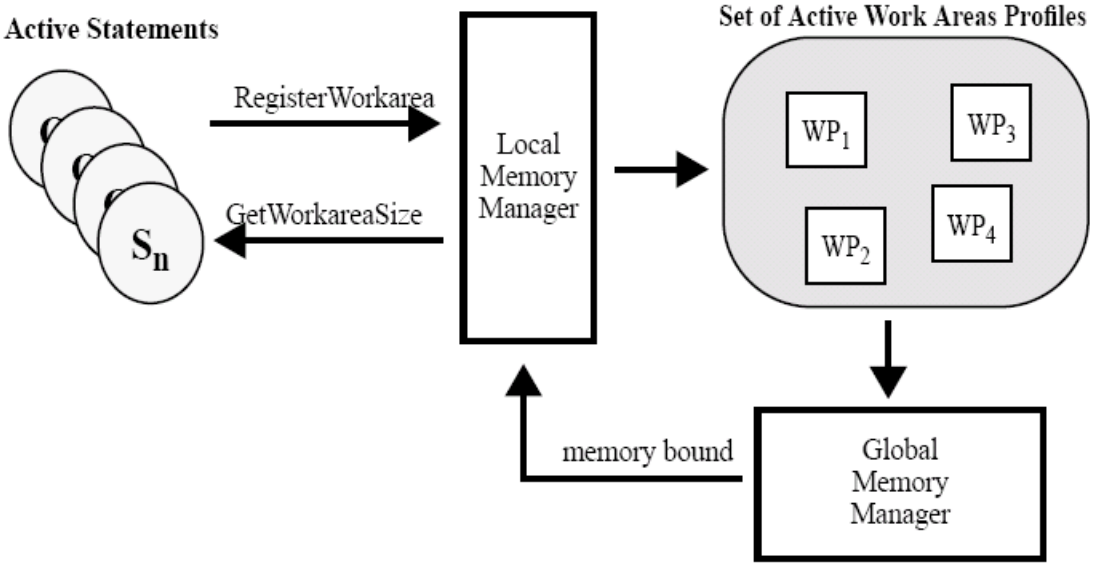
```
  4                       WHERE NAME LIKE 'workarea executions%'), 0, NULL,
  5                       (SELECT SUM (VALUE) FROM v$sysstat
  6                       WHERE NAME LIKE 'workarea executions%'))) pct
  7    FROM v$sysstat WHERE NAME LIKE 'workarea executions%';

NAME                                                      VALUE        PCT
--------------------------------                      ---------- -----------
workarea executions - optimal                              22478   97.9433551
workarea executions - onepass                                397   1.72984749
workarea executions - multipass                               75   .326797386
```

5.2.5 自动 PGA 管理实现原理

自动 PGA 管理采用反馈环（Feedback Loop）算法实现，其原理如下图所示。当活动进程开始执行 SQL 语句时，首先会通过 Local Memory Manager 注册一个 Active Workarea Profile，工作区 Profile 是进程与内存管理器之间通讯的唯一接口，Profile 是包含了这个 Workarea 的一系列属性（如类型、执行所需要的 minimum、one-pass and optimal 内存大小等）的元数据。

工作区活动Profile集通过Local Memory Manager维护，存储在SGA之中，由于Porfile经常被更新，所以所有Active Profile基本可以反应出当前PGA内存需要和当前正在使用的内存。有了这些Profile信息，后台的Global Memory Manager就可以计算出一个既能限制内存使用、又能提供较好性能的Global Memory Bound，这个值用于限制单个进程使用的PGA内存上限；Global Memory Manager每隔3秒更新一次Memory bound，Local Memory Manager得到Memory Bound后会计算出每个Active Statement所需要分配的PGA内存大小，在这里被称为Expect Size，然后每个Active Statement将会在自己所分配到的Expect Size内存中进行运算：



注意，在以上流程中，Global Memory Manager 并不直接参与 PGA 内存的分配，但是通过

其计算得出的 Global Memory Bound 将影响所有进程的 PGA 分配。

Global Memory Manager 由 CKPT 后台进程实现。通过底层表 x$messages 可以发现如下记录：

```
SQL> select description,dest from x$messages where description like 'SQL Memory%';
DESCRIPTION                              DEST
---------------------------------------- ----------
SQL Memory Management Calculation        CKPT
```

以下查询来自 Oracle11g 数据库，通过 x$trace 表同样可以发现这样的信息：

```
SQL> set linesize 120
SQL> column time format 99999999999999999
SQL> column data format a80
SQL> SELECT TIME time,data
  2  FROM x$trace WHERE data LIKE '%SQL Memory%' ORDER BY seq#;
              TIME DATA
------------------ ------------------------------------------------------------
  1215510948066330 KSBCTI: (CKPT) : (timeout action)   : acnum=[178] comment=[S
                   QL Memory Management Calculation]
  1215510951087639 KSBCTI: (CKPT) : (timeout action)   : acnum=[178] comment=[S
                   QL Memory Management Calculation]
  1215510954109207 KSBCTI: (CKPT) : (timeout action)   : acnum=[178] comment=[S
                   QL Memory Management Calculation]
```

在 Oracle10gR2 以及 Oracle11g 中，可以通过 v$sgastat 视图来查询工作区的管理内存分配，这部分内存在 Shared Pool 中分配：

```
SQL> select * from v$version where rownum <2;
BANNER
--------------------------------------------------------------------------------
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
SQL> select * from v$sgastat where name like 'work area%';
POOL         NAME                            BYTES
------------ -------------------------- ----------
shared pool  work area tab                   265320
```

伴随自动 PGA 调整新特性的引入，Oracle 随之引入了一系列新的视图,V$PGASTAT 就是其中之一，Global Memory Bound 就记录在该视图之中，以下可以从不同的 PGA 参数设置来观察一下 Oracle 运行的 PGA 上限（测试来自 Oracle9iR2 环境）：

```
SQL> alter system set pga_aggregate_target=10m;
System altered.
SQL> SELECT DISTINCT * FROM t WHERE ROWNUM < 500000;
20000 rows selected.
Elapsed: 00:03:04.12
```

```
SQL> SELECT sql_text, operation_type, POLICY, last_memory_used / 1024 / 1024,
  2         last_execution, last_tempseg_size
  3    FROM v$sql l, v$sql_workarea a  WHERE l.hash_value = a.hash_value
  4     AND sql_text = 'SELECT DISTINCT * FROM t WHERE ROWNUM < 500000';
OPERATION_TYPE     POLIC    LAST_MEMORY_USED/1024/1024 LAST_EXE LAST_TEMPSEG_SIZE
------------------ ------    ------------------------  -------- -----------------
GROUP BY (SORT)    AUTO                .548828125206      PASSES         62914560
SQL> SELECT NAME, VALUE / 1024 / 1024 MB FROM v$pgastat
  2   WHERE NAME IN ('aggregate PGA target parameter', 'global memory bound');
NAME                                                                  MB
---------------------------------------------------------------- ----------
aggregate PGA target parameter                                        10
global memory bound                                                   .5
SQL> alter system set pga_aggregate_target=30M;
System altered.
SQL> SELECT DISTINCT * FROM t WHERE ROWNUM < 500000;
20000 rows selected.
Elapsed: 00:00:53.30
SQL> SET autotrace off
SQL> SELECT operation_type, POLICY, last_memory_used / 1024 / 1024,
  2         last_execution, last_tempseg_size
  3    FROM v$sql l, v$sql_workarea a WHERE l.hash_value = a.hash_value
  4     AND sql_text = 'SELECT DISTINCT * FROM t WHERE ROWNUM < 500000';
OPERATION_TYPE     POLIC LAST_MEMORY_USED/1024/1024  LAST_EXECUTION     LAST_TEMPSEG_SIZE
------------------ ----- --------------------------- --------------     -----------------
GROUP BY (SORT)    AUTO                 1.480468756 PASSES                     57671680
SQL> SELECT NAME, VALUE / 1024 / 1024 MB FROM v$pgastat
  2  WHERE NAME IN ('aggregate PGA target parameter', 'global memory bound');
NAME                                                                  MB
---------------------------------------------------------------- ----------
aggregate PGA target parameter                                        30
global memory bound                                                   1.5
```

在 Oracle9iR2 中，通过以上测试可以注意到，PGA 的 global memory bound 会一直处在 5%的PGA_AGGREGATE_TARGET 参数设置，直到5%*PGA_AGGREGATE_TARGET 超过 100M,然后 global memory bound 被限制为 100M。

修改 PGA_AGGREGATE_TARGET 参数可以使用类似如下命令:

alter system set pga_aggregate_target=*4096M ;*

修改参数后,通常需要执行一些排序操作才能看到视图信息的变化，以下通过一些不同设置输出继续来看一下 PGA_AGGREGATE_TARGET 与 global memory bound 的关系：

```
SQL> SELECT NAME, VALUE / 1024 / 1024 MB FROM v$pgastat
 2   WHERE NAME IN ('aggregate PGA target parameter', 'global memory bound');
NAME                                                             MB
---------------------------------------------------------------- ----------
aggregate PGA target parameter                                    10
global memory bound                                              .5
SQL> SELECT NAME, VALUE / 1024 / 1024 MB FROM v$pgastat
  2  WHERE NAME IN ('aggregate PGA target parameter', 'global memory bound');
NAME                                                              MB
---------------------------------------------------------------- ----------
aggregate PGA target parameter                                    40
global memory bound                                               2
SQL> SELECT NAME, VALUE / 1024 / 1024 MB FROM v$pgastat
  2  WHERE NAME IN ('aggregate PGA target parameter', 'global memory bound');
NAME                                                             MB
---------------------------------------------------------------- ----------
aggregate PGA target parameter                                    1024
global memory bound                                              51.1992188
SQL> SELECT NAME, VALUE / 1024 / 1024 MB FROM v$pgastat
  2  WHERE NAME IN ('aggregate PGA target parameter', 'global memory bound');
NAME                                                              MB
---------------------------------------------------------------- ----------
aggregate PGA target parameter                                    4096
global memory bound                                               100
```

实际上这个 100M 的上限是受到了另外一个隐含参数的控制，该参数为_pga_max_size，在 Oracle9iR2 中该参数的缺省值为 200M，单进程串行操作 PGA 的上限不能超过该参数的 1/2.

```
SQL> @GetHidPar
Enter value for par: pga_max
old   6:    AND x.ksppinm LIKE '%&par%'
new   6:    AND x.ksppinm LIKE '%pga_max%'
NAME                           VALUE      DESCRIB
---------------------------------------------  ---------------  ---
_pga_max_size            209715200     Maximum size of the PGA memory for one
process
```

如果修改该参数， global memory bound 将可以突破 100M 的上限:

```
SQL> alter system set "_pga_max_size"=400M;
System altered.
……
SQL> SELECT NAME, VALUE / 1024 / 1024 MB FROM v$pgastat
```

```
  2  WHERE NAME IN ('aggregate PGA target parameter', 'global memory bound');
NAME                                                                MB
---------------------------------------------------------------- ----------
aggregate PGA target parameter                                    4096
global memory bound                                                200
```

从 Oracle10gR2 开始，_pga_max_size 的设置和 PGA_AGGREGATE_TARGET 相关，在不同 P_A_T 参数设置下，_pga_max_size 会自动调整以提供更好的性能需要。通过一些简单测试可以得出 Oracle10gR2 下两者的关系：

_pga_max_size	PGA_AGGREGATE_TARGET	比例关系
209715200	800M	0.25
214732800	1G	0.2
429486080	2G	0.2
644239360	3G	0.2
858992640	4G	0.2
939180032	5G	0.2
939180032	6G	0.15

注意到当 P_A_T 设置大于 5G 之后，_pga_max_size 不再变化，在 1G ~ 5G 范围，_pga_max_size 按照 20%*P_A_T 设置增长。当 PGA 增大，单进程能够使用的最大内存应当随之增加，这种增强是极其有益的。

对于 PGA 的控制,还有一系列的内部参数,列举如下,仅供参考:

```
NAME                      VALUE DESCRIB
------------------------- ----- ------------------------------------------------------------
_smm_auto_min_io_size     56    Minimum IO size (in KB) used by sort/hash-join in auto mode
_smm_auto_max_io_size     248   Maximum IO size (in KB) used by sort/hash-join in auto mode
_smm_auto_cost_enabled    TRUE  if TRUE, use the AUTO size policy cost functions
_smm_control              0     provides controls on the memory manager
_smm_trace                0     Turn on/off tracing for SQL memory manager
_smm_min_size             128   minimum work area size in auto mode
_smm_max_size             2560  maximum work area size in auto mode (serial)
_smm_px_max_size          15360 maximum work area size in auto mode (global)
_smm_bound                0     overwrites memory manager automatically computed bound
_smm_advice_log_size      0     overwrites default size of the PGA advice workarea history log
_smm_advice_enabled       TRUE  if TRUE, enable v$pga_advice
```

5.2.6 PGA 的调整建议

伴随自动 PGA 调整功能的引入，Oracle 同时引入相应的动态性能视图用于优化建议，PGA 的优化建议通过 v$pga_target_advice 和 v$pga_target_advice_histogram 提供。

v$pga_target_advice 视图通过对不同 PGA 设置进行评估，给出在不同设置下的 PGA 命中率和 OverAlloc 等信息。

```
SQL> select PGA_TARGET_FOR_ESTIMATE/1024/1024 PGAMB, PGA_TARGET_FACTOR,
```

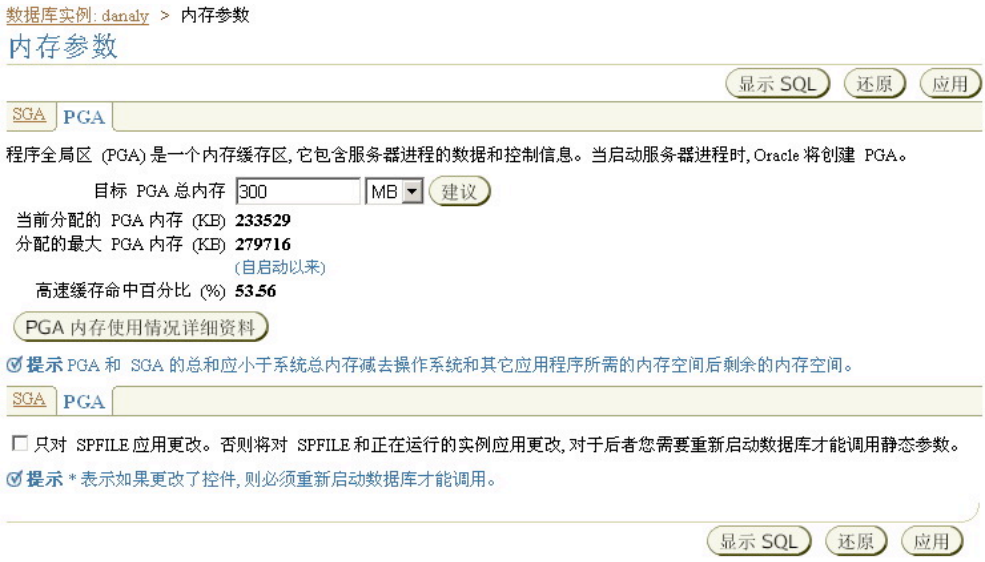
```
  2  ESTD_PGA_CACHE_HIT_PERCENTAGE, ESTD_OVERALLOC_COUNT
  3  from v$pga_target_advice;
```

PGAMB	PGA_TARGET_FACTOR	ESTD_PGA_CACHE_HIT_PERCENTAGE	ESTD_OVERALLOC_COUNT
37.5	0.125	35	391
75	0.25	43	58
150	0.5	61	0
225	0.75	70	0
300	1	70	0
360	1.2	71	0
420	**1.4**	**72**	**0**
480	1.6	72	0
540	1.8	72	0
600	2	72	0
900	3	72	0
1200	4	72	0
1800	6	72	0
2400	8	72	0

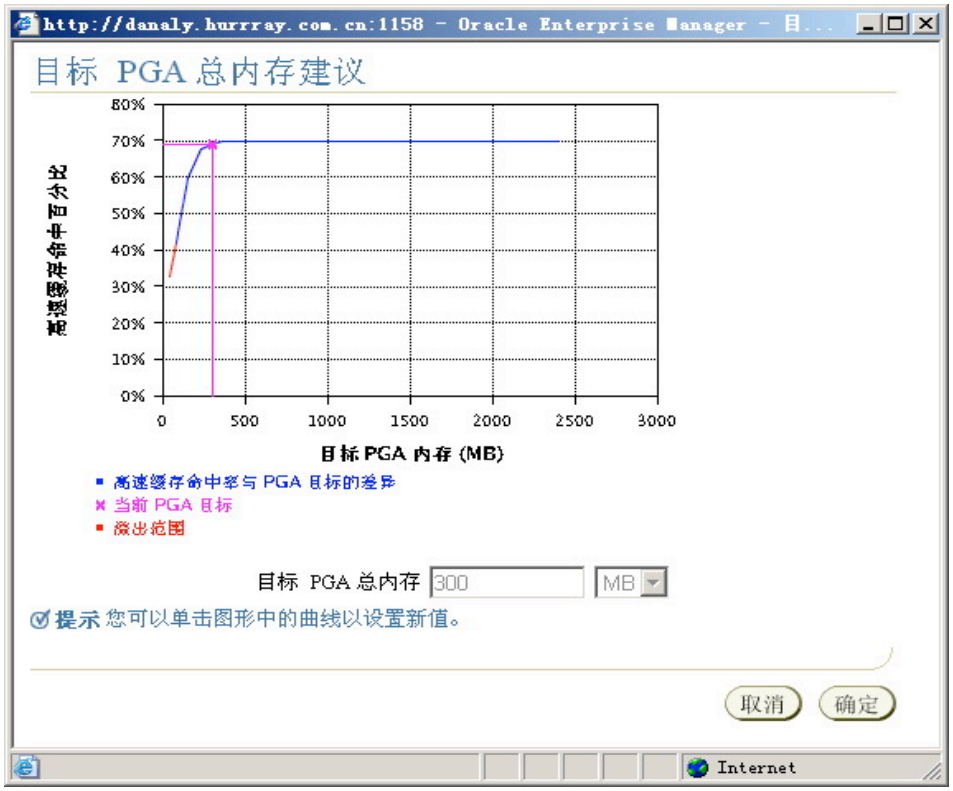
```
14 rows selected
```

可以看到，在以上输出中，当 PGA 设置为 150M 时，即可消除 PGA 过载；在 PGA 设置为 420M 时，可以达到最高命中率 72%；当前 PGA 设置为 300M。根据这些信息，我们可以调整 PGA 为 420M，从而提高性能。

从 Oracle9i 或者 Oracle10g 提供的 OEM 界面上，我们都可以直观的看到这些建议信息：下图是当前的 PGA 设置页：



点击建议按钮我们可以看到 Oracle 对于 PGA 设置的建议：



本例修改后数据库的 PGA 分配为 420M:

```
SQL> show parameter pga
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
pga_aggregate_target                 big integer 420M
```

该曲线既是根据 v$pga_target_advice 收集的信息绘制出来的，我们可以直接在图中拖动调整 PGA 的设置，确定后在 PGA 页应用该修改即可达到优化数据库的目的。

调整 PGA 后，建议信息会刷新，统计信息需要重新评估：

```
SQL> select PGA_TARGET_FOR_ESTIMATE/1024/1024 PGAMB,PGA_TARGET_FACTOR,
  2  ESTD_PGA_CACHE_HIT_PERCENTAGE,ESTD_OVERALLOC_COUNT
  3   from v$pga_target_advice;
     PGAMB PGA_TARGET_FACTOR ESTD_PGA_CACHE_HIT_PERCENTAGE ESTD_OVERALLOC_COUNT
---------- ----------------- ---------------------------- --------------------
      52.5             0.125                          100                    1
       105              0.25                          100                    0
       210               0.5                          100                    0
       315              0.75                          100                    0
       420                 1                          100                    0
       504               1.2                          100                    0
       588               1.4                          100                    0
```

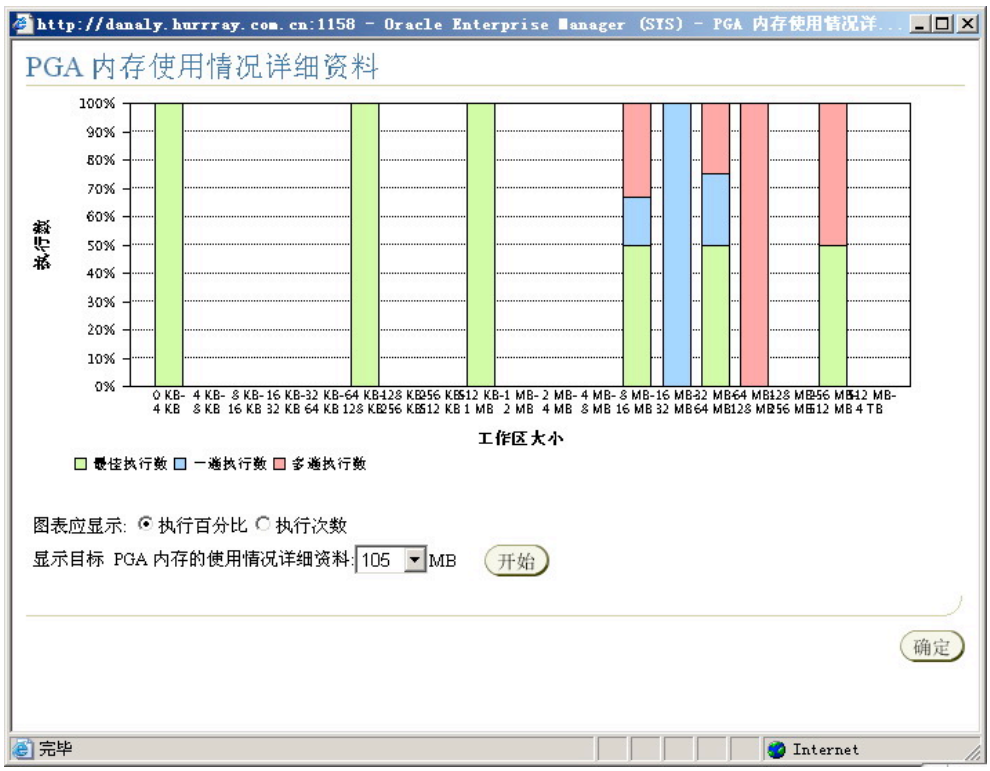
672	1.6	100	0
756	1.8	100	0
840	2	100	0
1260	3	100	0
1680	4	100	0
2520	6	100	0
3360	8	100	0

14 rows selected

v$pga_target_advice_histogram 视图可以通过对不同工作区大小的采样评估提供统计信息供分析使用。其中几个重要字段有：

- LOW_OPTIMAL_SIZE-Histogram 评估区间内 Optimal 下限 (bytes)
- HIGH_OPTIMAL_SIZE-Histogram 评估区间内 Optimal 上限 (bytes)
- ESTD_OPTIMAL_EXECUTIONS-Histogram 评估区间内估计 optimal 执行次数
- ESTD_ONEPASS_EXECUTIONS-Histogram 评估区间内估计 onepass 执行次数
- ESTD_MULTIPASSES_EXECUTIONS-Histogram 评估区间内估计 multipass 执行次数
- ESTD_TOTAL_EXECUTIONS-Histogram 评估区间内估计执行总次数

我们可以从 OEM 的直观柱状图中得到不同 PGA 设置下的评估数据：



同样的，这个柱状图表的信息来自动态性能视图 v$pga_target_advice_histogram，我们可以从数据库中查询得到这些数据，以上柱状图正是依据这些数据绘制得出的：

```
SQL> SELECT pga_target_factor factor, low_optimal_size / 1024 low,
```

```
  2         ROUND (high_optimal_size / 1024) high,
  3         estd_optimal_executions estd_opt, estd_onepass_executions estd_op,
  4         estd_multipasses_executions estd_mp, estd_total_executions estd_exec
  5    FROM v$pga_target_advice_histogram
  6   WHERE pga_target_factor = 0.25 AND estd_total_executions > 0;
    FACTOR        LOW       HIGH   ESTD_OPT    ESTD_OP    ESTD_MP  ESTD_EXEC
---------- ---------- ---------- ---------- ---------- ---------- ----------
      0.25     262144     524288          1          0          1          2
      0.25      65536     131072          0          0          2          2
      0.25      32768      65536          4          2          2          8
      0.25      16384      32768          0          2          0          2
      0.25       8192      16384          3          1          2          6
      0.25        512       1024       2038          0          0       2038
      0.25         64        128          3          0          0          3
      0.25          2          4       4540          0          0       4540
8 rows selected
```

PGA 的新特性使得 PGA 的管理得以极大简化。

5.2.7 PGA 的转储

对于 PGA 的进一步研究可以通过转储进程内存结构来实现，Oracle 提供如下命令用于将进程内存地址信息转储到跟踪文件：

alter session set events 'immediate trace name heapdump level n';

其中不同的 level 级别决定了将哪些内存堆转储到跟踪文件：

Level 1:　PGA 汇总信息
Level 2:　SGA 汇总信息
Level 4:　UGA 汇总信息
Level 8:　当前调用的汇总信息（CGA）
Level 16:　用户调用的汇总信息（CGA）
Level 32:　Large pool 的汇总信息（LGA）
Level 1025:　PGA 详细信息
Level 2050:　SGA 详细信息
Level 5000:　UGA 详细信息
Level 8200:　当前调用的详细信息
Level 16400:　用户调用的详细信息
Level 32800:　Large pool 的详细信息

看一个简单的范例，通过 Level 1 转储 PGA 概要信息：

```
SQL> alter session set events 'immediate trace name heapdump level 1';
Session altered.
```

检查跟踪文件可以发现和 PGA 相关的内容：

```
HEAP DUMP heap name="pga heap"  desc=0xc6a6580
 extent sz=0x206c alt=92 het=32767 rec=0 flg=2 opc=2
 parent=(nil) owner=(nil) nex=(nil) xsz=0xfff8
EXTENT 0 addr=0xb5f20008
  Chunk b5f20010 sz=    18972    perm      "perm           "  alo=7732
  Chunk b5f24a2c sz=     7076    free      "               "
  Chunk b5f265d0 sz=     4164    freeable  "Alloc environm "  ds=0xdca0e8c
  Chunk b5f27614 sz=    20568    freeable  "Fixed Uga      "
  Chunk b5f2c66c sz=      176    freeable  "ldm context    "
  Chunk b5f2c71c sz=      176    freeable  "ldm context    "
。。。。。。。。。。。。。。。
  Chunk  dcd37f0 sz=      220    freeable  "kopolal dvoid  "
  Chunk  dcd38cc sz=     1292    freeable  "kews sqlstat st"
  Chunk  dcd3dd8 sz=      260    freeable  "KFK PGA        "
  Chunk  dcd3edc sz=     4144    freeable  "qmtmInit       "  ds=0xdcd00c0
EXTENT 2 addr=0xdcd0e28
  Chunk  dcd0e30 sz=     4148    freeable  "qmtmInit       "  ds=0xdcd00c0
  Chunk  dcd1e64 sz=     4144    freeable  "qmtmInit       "  ds=0xdcd00c0
EXTENT 3 addr=0xdcced28
  Chunk  dcced30 sz=     6448    perm      "perm           "  alo=6448
  Chunk  dcd0660 sz=      144    free      "               "
  Chunk  dcd06f0 sz=       28    freeable  "ldm context    "
  Chunk  dcd070c sz=       72    freeable  "KFIO PGA struct"
  Chunk  dcd0754 sz=     1600    recreate  "qmtmInit       "  latch=(nil)
     ds  dcd00c0 sz=    14036 ct=        4
EXTENT 4 addr=0xdccccb0
  Chunk  dccccb8 sz=     6292    perm      "perm           "  alo=6292
  Chunk  dcce54c sz=     2000    recreate  "joxp heap      "  latch=(nil)
     ds  dcc7a98 sz=     2000 ct=        1
EXTENT 5 addr=0xdcc7c88
  Chunk  dcc7c90 sz=    20496    perm      "perm           "  alo=20496
EXTENT 6 addr=0xdcc5c10
  Chunk  dcc5c18 sz=     7224    perm      "perm           "  alo=7224
  Chunk  dcc7850 sz=      436    perm      "perm           "  alo=436
  Chunk  dcc7a04 sz=      136    perm      "perm           "  alo=124
  Chunk  dcc7a8c sz=      296    freeable  "joxp heap      "
```

进一步的，可以将某个具体的数据存储结构转储出来（DS），如以上的 ds dcd00c0 其空间使用的大小为 sz= 14036，首先对空间地址进行一下转换：

```
SQL> select to_number('dcd00c0','xxxxxxxxxx') from dual;
TO_NUMBER('DCD00C0','XXXXXXXXXX')
---------------------------------
                        231538880
```

然后在 Oracle9iR2 之后使用如下命令转储固定地址空间的内容：

alter session set events 'immediate trace name heapdump_addr level 1, addr n';

以上计算的地址空间可以通过如下命令转储：

```
SQL> ALTER SESSION SET EVENTS 'immediate trace name heapdump_addr level 1, addr 231538880';
Session altered.
```

其内容如下所示：

```
HEAP DUMP heap name="qmtmInit"  desc=0xdcd00c0
 extent sz=0x1024 alt=32767 het=32767 rec=0 flg=2 opc=2
 parent=0xc6a6580 owner=(nil) nex=(nil) xsz=0x1024
EXTENT 0 addr=0xdcd3ee8
  Chunk  dcd3ef0 sz=     2048    free      "               "
  Chunk  dcd46f0 sz=     2076    freeable  "qmushtCreate   "
EXTENT 1 addr=0xdcd0e3c
  Chunk  dcd0e44 sz=      932    free      "               "
  Chunk  dcd11e8 sz=       44    freeable  "qmtmltAlloc    "
……
  Chunk  dcd0d5c sz=       56    freeable  "qmtmltCreate   "
Total heap size    =    13940
FREE LISTS:
 Bucket 0 size=0
  Chunk  dcd3ef0 sz=     2048    free      "               "
  Chunk  dcd0e44 sz=      932    free      "               "
  Chunk  dcd1e78 sz=      928    free      "               "
  Chunk  dcd0778 sz=      444    free      "               "
Total free space   =     4352
UNPINNED RECREATABLE CHUNKS (lru first):
PERMANENT CHUNKS:
Permanent space    =        0
```

5.2.8 转储与 PGA 分析

通过上一节提到的 PGA 内存转储技术，可以对 PGA 的运行原理进行进一步的分析。下面来探讨一下 Oracle9i 中引入的_use_realfree_heap 在 PGA 管理中的作用。

```
SQL> connect eygle/eygle
Connected.
```

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Prod

SQL> ALTER SESSION SET EVENTS 'immediate trace name heapdump level 1';

Session altered.
SQL> SELECT    a.VALUE || b.symbol || c.instance_name || '_ora_' || d.spid || '.trc' trace_file
  2    FROM (SELECT VALUE FROM v$parameter WHERE NAME = 'user_dump_dest') a,
  3  (SELECT SUBSTR (VALUE, -6, 1) symbol FROM v$parameter WHERE NAME = 'user_dump_dest') b,
  4         (SELECT instance_name FROM v$instance) c,
  5         (SELECT spid FROM v$session s, v$process p, v$mystat m
  6           WHERE s.paddr = p.addr AND s.SID = m.SID AND m.statistic# = 0) d;
TRACE_FILE
--------------------------------------------------------------------------------
/opt/oracle/admin/mmstest/udump/mmstest_ora_5669.trc
```

检查这个进程转储文件，可以发现如下 Heap 地址信息及空间分配：

```
[oracle@test oracle]$ grep HEAP /opt/oracle/admin/mmstest/udump/mmstest_ora_5669.trc
HEAP DUMP heap name="pga heap"  desc=0xcd3a320
HEAP DUMP heap name="top call heap"  desc=0xcd3cb20
HEAP DUMP heap name="top uga heap"  desc=0xcd3cc40
```

根据输出可以看到，在自动管理模式下，PGA,CGA,UGA 都是独立分配的。

通过跟踪连接会话，可以获得操作系统上调用执行情况：

```
[oracle@test oracle]$ strace -p 5669 -o 5669.txt
```

在 SQL*Plus 中执行一个简单查询：

```
SQL> select count(*) from dba_users;
  COUNT(*)
----------
        16
```

摘录跟踪文件的主要内容如下：

```
[oracle@test oracle]$ grep -v get 5669.txt
read(8, "\0\313\0\0\6\0\0\0\0\21i\37p\265\226\10\1\0\0\0\3\0\0"..., 2064) = 203
times(NULL)                             = 32667140
times(NULL)                             = 32667141
times(NULL)                             = 32667141
times(NULL)                             = 32667141
times(NULL)                             = 32667141
mmap2(NULL, 524288, PROT_NONE, MAP_PRIVATE|MAP_NORESERVE, 7, 0x74) = 0xb5e84000
```

```
mmap2(0xb5e84000, 65536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 7, 0) =
0xb5e84000
mmap2(0xb5e94000, 65536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 7, 0) =
0xb5e94000
mmap2(0xb5ea4000, 131072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 7, 0) =
0xb5ea4000
mmap2(0xb5ec4000, 131072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 7, 0) =
0xb5ec4000
mmap2(0xb5ee4000, 131072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 7, 0) =
0xb5ee4000
mmap2(NULL, 1048576, PROT_NONE, MAP_PRIVATE|MAP_NORESERVE, 7, 0x1f4) = 0xb5c84000
mmap2(0xb5c84000, 131072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 7, 0) =
0xb5c84000
mmap2(0xb5ca4000, 131072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 7, 0) =
0xb5ca4000
mmap2(0xb5cc4000, 131072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 7, 0) =
0xb5cc4000
mmap2(0xb5ce4000, 131072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 7, 0) =
0xb5ce4000
munmap(0xb5c84000, 1048576)             = 0
munmap(0xb5e84000, 524288)              = 0
times(NULL)                             = 32667141
write(11, "\1+\0\0\6\0\0\0\0\0\20\27[\240\201\356\305\235\256B\246"..., 299) = 299
read(8, "\0\25\0\0\6\0\0\0\0\0\3\5!\2\0\0\0\17\0\0\0", 2064) = 21
times(NULL)                             = 32667142
times(NULL)                             = 32667142
write(11, "\0\204\0\0\6\0\0\0\0\0\4\1\0\0\0\37\0\1\1\0\0\0{\5\0\0"..., 132) = 132
read(8,  <unfinished ...>
```

注意以上输出，在 SQL 执行过程中，进行了两块内存分配，通过 mmap 调用实现，在执行完毕之后，通过 munmap 来释放了两块连续的内存。

而如果设置_use_realfree_heap 为 False，再来转储进程 HEAP 空间：

SQL> alter system set "_use_realfree_heap"=false;

System altered.

SQL> ALTER SESSION SET EVENTS 'immediate trace name heapdump level 1';

Session altered.

可以看到进程 HEAP 此次仅分配了一个 PGA HEAP：

```
[oracle@test udump]$ grep HEAP mmstest_ora_5929.trc
HEAP DUMP heap name="pga heap"  desc=0xcd3a320
```

这就是自动的 PGA 管理和 Oracle9i 之前手动的 PGA 管理的区别所在。

5.2 SGA 管理

SGA 指系统全局区（System Global Area）,是一块用于加载数据、对象并保存运行状态和数据库控制信息的一块内存区域，在数据库实例启动时分配，当实例关闭时释放，每个实例都拥有自己的 SGA 区。

在第一章曾经提到，当数据库启动到 nomount 状态时，SGA 已经分配，同时启动后台进程，在 SQL*Plus 中通过 show sga 命令可以看到 SGA 的分配情况：

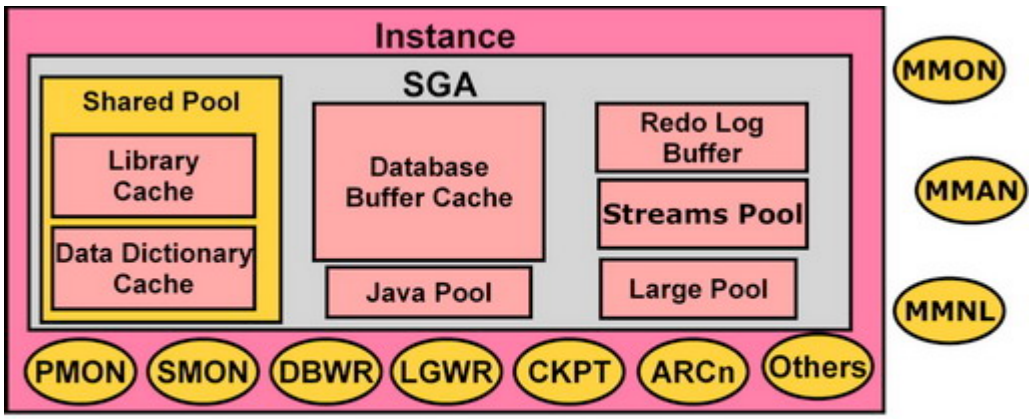
```
SQL> show sga

Total System Global Area  338390716 bytes
Fixed Size                   102076 bytes
Variable Size             133308416 bytes
Database Buffers          204800000 bytes
Redo Buffers                 180224 bytes
```

连接到 Oracle 数据库的用户都可以共享 SGA 中的数据，通常为了更优化的性能，我们总是期望在物理内存允许的情况下，设置更高的 SGA 区，以减少物理 I/O（SGA 中数据缓冲区的增大可以有效地减少物理读）。

5.1.1 SGA 的组成

下图是最常见的数据库实例体系结构图，展现了 SGA 的结构：



Oracle Instance简图

1. 固定区域 - Fixed Area

Fixed Size 部分是 SGA 中的固定部分，包含几千个变量和一些小的数据结构，如 Latch 或地址指针等，这部分内存分配和特定的数据库版本以及平台有关，不受用户控制，而且这些信息对于数据库来说非常重要，但是通常我们用户不需要关心。

固定部分只需要很小的内存，可以通过一个内部表 X$KSMFSV（[**K**]ernel [**S**]ervice Layer，[**M**]emory Management，Addresses of [**F**]ixed [**S**]GA [**V**]ariables）查询。此外 Oracle 的内部表 X$KSMMEM 记录了整个 SGA 的地址映射关系，通过 X$KSMFSV 和 X$KSMMEM 关联，可以找出 Fixed Area 中每个变量的设置。

在 32 位平台上，X$KSMMEM 表中每条记录代表 4 Bytes，在 64 位平台，每条记录代表 4 Bytes：

```
SQL> select banner from x$version where indx in (0,3);
BANNER
----------------------------------------------------------------
Oracle9i Enterprise Edition Release 9.2.0.4.0 - 64bit Production
TNS for Solaris: Version 9.2.0.4.0 - Production
SQL> show sga
Total System Global Area 2400687424 bytes
SQL> select count(*) from x$ksmmem;
  COUNT(*)
----------
 300085928
SQL> select 2400687424/300085928 from dual;
2400687424/300085928
--------------------
                   8
SQL> select * from x$ksmmem where rownum <5;
ADDR                   INDX    INST_ID KSMMMVAL
---------------- ---------- ---------- ----------------
0000000380000000          0          1 0000000000000EEE
0000000380000008          1          1 0920040000000001
0000000380000010          2          1 00
0000000380000018          3          1 00
```

以下是 32 位 Linux 平台的设置情况：

```
SQL> select banner from x$version where indx in (0,3);
BANNER
--------------------------------------------------------------------------------
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
TNS for Linux: Version 11.1.0.6.0 - Production
```

```
SQL> select count(*) from x$ksmmem;
  COUNT(*)
----------
 130777088
SQL> show sga
Total System Global Area  523108352 bytes
SQL> select 523108352/130777088 from dual;
523108352/130777088
-------------------
                  4
```

在修改某些相关参数及变量时，可以看到内存中相关变量的改变：

```
SQL> select a.ksmfsnam,b.ksmmmval,b.addr from x$ksmfsv a, x$ksmmem b
  2  where a.ksmfsadr=b.addr and a.ksmfsnam like 'sgaflg%';
KSMFSNAM             KSMMMVAL ADDR
-------------------- -------- --------
sgaflg_              0070021A 2001A3C0
SQL> alter system disable distributed recovery;
System altered.
SQL> select a.ksmfsnam,b.ksmmmval,b.addr from x$ksmfsv a, x$ksmmem b
  2  where a.ksmfsadr=b.addr and a.ksmfsnam like 'sgaflg%';
KSMFSNAM             KSMMMVAL ADDR
-------------------- -------- --------
sgaflg_              00700212 2001A3C0
SQL> alter system enable distributed recovery;
System altered.
SQL> select a.ksmfsnam,b.ksmmmval,b.addr from x$ksmfsv a, x$ksmmem b
  2  where a.ksmfsadr=b.addr and a.ksmfsnam like 'sgaflg%';
KSMFSNAM             KSMMMVAL ADDR
-------------------- -------- --------
sgaflg_              0070021A 2001A3C0
```

通过 oradebug 也可以看到类似变化：

```
SQL> oradebug setmypid
SQL> oradebug dumpvar sga sgaflg
SQL> alter system disable distributed recovery;
SQL> oradebug dumpvar sga sgaflg
SQL> alter system enable distributed recovery;
SQL> oradebug dumpvar sga sgaflg
```

此外我们知道 SCN 就记录在 Fixed Size 内存之中，Oracle 使用 32 位来存储 SCN 值，可以查询获得这个信息：

```
SYS@WAPDBS>select ksmfsnam,ksmfssiz from x$ksmfsv
  2  where ksmfsnam='kcsgscn_';
KSMFSNAM                                                         KSMFSSIZ
---------------------------------------------------------------- ----------
kcsgscn_                                                             32
```

通过 ORADEBUG 工具可以得到当前内存中的 SCN 值：

```
SYS@WAPDBS>oradebug setmypid
Statement processed.
SYS@WAPDBS>oradebug DUMPvar SGA kcsgscn_
kcslf kcsgscn_ [2000C848, 2000C868) = 00000000 000E23B5 00001322 00000000 00000000 00000000
00000000 2000C654
SYS@WAPDBS>select dbms_flashback.get_system_change_number from dual;
GET_SYSTEM_CHANGE_NUMBER
------------------------
                  926645
SYS@WAPDBS>select to_number('E23B5','xxxxxx') SCN from dual;
       SCN
----------
    926645
```

Fixed Area 包含很多控制信息，但是需要注意的是，查询 X$KSMFSV 视图可能会导致进程异常，需要谨慎使用。

1. Buffer Cache

Buffer Cache-缓冲区高速缓存，用于存储最近使用的数据块，这些数据块可能是被修改过的，也可能是未经修改的。我们知道，在 Oracle 对数据的处理过程中，代价最昂贵的就是物理 I/O（Physical I/O ）操作了，同样的数据从内存中得到要比从磁盘上读取快得多，所以将尽可能多的数据保存在内存中，可以减少磁盘 I/O 操作，从而提高数据库的性能。

在 Oracle9i 之前，Buffer Cache 的设置主要由两个参数决定：db_block_buffers 和 db_block_size。db_block_buffers 设置分配给 Buffer Cache 的缓冲区数量，这个数值乘以 db_block_size 得出的才是 Buffer Cache 的大小。

```
SQL> select * from v$version where rownum <2;
Oracle8i Enterprise Edition Release 8.1.7.4.0 - 64bit Production
SQL> select name,value from v$parameter where name in ('db_block_buffers','db_block_size');
NAME            VALUE
------------------- --------------------
db_block_buffers     25000
db_block_size        8192
SQL> select (select value from v$parameter where name='db_block_buffers')
```

```
  2  *(select value from v$parameter where name='db_block_size') / 1024 /1024 Buffer_Cache_MB
  3  from dual;
BUFFER_CACHE_MB
---------------
       195.3125
```

从 Oracle9i 开始，Oracle 引入了一个新的初始化参数 db_cache_size ,该参数用来定义主 Block Size（db_block_size 定义的块大小）的 Default 缓冲池的大小。

db_cache_size 最小值为一个粒度（Granule）。

粒度也是 Oracle9i 引入的一个新的概念，是连续虚拟内存分配的单位，其大小取决于估计的 SGA 的总大小（SGA 总大小由 SGA_MAX_SIZE 参数得到）：

- 如果估计的 SGA 大小< 128 MB 则值为 4 MB
- 否则值为 16 MB（32 位 Windows 为 8M）

在 Oracle10gR2 中，Granule 的分配算法有所改变，在大多数平台上：

- 如果 SGA<1G，Granule 通常为 4M
- 否则 Granule 为 16M（32 位 Windows 为 8M）。

这个 Granule 大小受到一个内部隐含参数_ksmg_granule_size 的控制,以下是 Oracle9iR2 中的测试输出(不同版本可能不同):

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle9i Enterprise Edition Release 9.2.0.4.0 - Production
SQL> show parameter sga_max_size
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
sga_max_size                         big integer      219223120
SQL> @GetParDescrb.sql
Enter value for par: _ksmg_granule_size
NAME                           VALUE                DESCRIB
------------------------------ -------------------- ----------------------------
_ksmg_granule_size             16777216             granule size in bytes
SQL> alter system set sga_max_size=120M scope=spfile;
System altered.
SQL> startup force;
SQL> show parameter sga_max_size
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
sga_max_size                         big integer 126948772
SQL> @GetParDescrb.sql
Enter value for par: _ksmg_granule_size
```

```
NAME                           VALUE              DESCRIB
------------------------------ ------------------ ------------------------------
_ksmg_granule_size             4194304             granule size in bytes
```

各内存组件所使用的 Granule 大小可以通过动态性能视图来查询：

```
SQL> select component,granule_size from v$sga_dynamic_components;
COMPONENT           GRANULE_SIZE
------------------- ------------
shared pool             16777216
large pool              16777216
buffer cache            16777216
```

大家知道内存空间总是有限的，Oracle 管理 Buffer Cache 使用的是 LRU 算法，但是这又带来另外一个问题，很多批处理的操作（比如全表扫描等）可能会导致 Buffer Cache 的刷新，将经常使用的数据“挤出”Buffer Cache，在不同版本中，Oracle 不停的改进 LRU 算法，以避免这类操作的过度影响。

但是在此之外，Oracle 提供了 Buffer Cache 的多缓冲池技术从另外一个方面来解决这个问题。所谓的多缓冲池技术是指，根据不同数据的不同访问方式，将 Buffer Cache 分为 Default、Keep 和 Recycle 池三个部分。对于经常使用的数据，我们可以在建表时就指定将其存放在 Keep 池中；对于经常一次性读取使用的数据，可以将其存放在 Recycle 池中；Keep 池中的数据倾向于一直保存，Recycle 池中的数据倾向于即时老化，而 Default 池则存放未指定存储池的数据，按照 LRU 算法管理。

默认情况下，所有表都使用 DEFAULT 池，它的大小就是数据缓冲区 Buffer Cache 的大小，由初始化参数 db_cache_size(8i 中是 db_block_size*db_block_buffers)决定。

如果我们在创建数据表或修改数据表时指定 STORAGE (BUFFER_POOL KEEP)或者 STROAGE(BUFFER_POOL RECYCLE)语句，就设置了这张表使用 KEEP 或者 RECYCLE 缓冲区。这两个缓冲区的大小分别由初始化参数 db_keep_cache_size 和 db_recycle_cache_size 来决定。

在 Oracle8i 中，只能修改参数文件，然后重新起动数据库，才能使对这两个参数的修改生效。在 Oracle9i 中，可以动态修改，下面看一下 Oracle9i 中这几个参数的设置:

```
SQL> show parameter cache_size
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_16k_cache_size                    big integer 0
db_2k_cache_size                     big integer 20971520
db_32k_cache_size                    big integer 0
db_4k_cache_size                     big integer 0
db_8k_cache_size                     big integer 0
db_cache_size                        big integer 4194304
db_keep_cache_size                   big integer 0
```

```
db_recycle_cache_size               big integer    0
SQL> alter system set db_keep_cache_size=4M;
System altered.
SQL> alter system set db_recycle_cache_size=4M;
System altered.
SQL> show parameter cache_size
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_16k_cache_size                    big integer 0
db_2k_cache_size                     big integer 20971520
db_32k_cache_size                    big integer 0
db_4k_cache_size                     big integer 0
db_8k_cache_size                     big integer 0
db_cache_size                        big integer 4194304
db_keep_cache_size                   big integer 4194304
db_recycle_cache_size                big integer 4194304
```

同时还可以看到，在 Oracle9i 中存在一系列的 db_*n*k_cache_size 参数，这是 Oracle9i 中引入的多块大小支持。Oracle9i 允许在同一个数据库中存在多种 Block_size 的表空间，分别支持：2k,4k,8k,16k 和 32k 的 Block_size,其中，由 db_block_size 定义的块大小被称为主 Block_size。

如果在数据库中创建不同 block_size 的表空间则需要分别设定 db_*n*k_cache_size 参数。

各缓冲池的设置，我们可以通过查询 v$buffer_pool 得到：

```
SQL> select id,name,block_size,current_size,target_size from v$buffer_pool;
        ID NAME                 BLOCK_SIZE CURRENT_SIZE TARGET_SIZE
---------- -------------------- ---------- ------------ -----------
         1 KEEP                       8192            4           4
         2 RECYCLE                    8192            4           4
         3 DEFAULT                    8192            4           4
         4 DEFAULT                    2048           20          20
```

2.Shared Pool

Shared Pool 通常被称为共享池，包含共享内存结构，如 SQL 区等。SQL 区包含 SQL 解析树、执行计划等信息，通过共享池，反复执行的 SQL 可以在不同 Session 间得到共享。

共享池的大小由参数 shared_pool_size 定义，在 Oracle9i 中，最小值为一个 Granule 大小。

关于共享池的设置和优化是非常重要和复杂的，我们将在下一章进行专题探讨。

3. Redo Log Buffer

Redo Log Buffer-日志缓冲区存储重做日志条目（redo entries）,日志记录数据库变更，最终将被写出到重做日志文件中，在数据库崩溃或故障时用于恢复；如果数据库运行在归档模式下，最终日志文件还会被写出到归档日志中，这些归档可以在介质恢复时用于进行数据恢复。

日志缓冲区的大小由初始化参数 log_buffer 决定。这是一个静态参数，不能动态调整。

4. 其他内存组件

Large Pool-大池是 SGA 的一个可选组件，通常用于共享服务器模式（MTS）、并行计算或 RMAN 的备份恢复等操作。

Java Pool-Java 池主要用于 JVM 等 Java 选件。

Streams Pool-Streams pool 是 Oracle10g 引入的概念，为 Oracle 的 Streams 功能所使用，如果不定义该参数，这部分内存将从 Shread Pool 中分配。

从下图我们可以看一下 Oracle SGA 的组成：

对于 SGA 各部分内存分配，可以从数据库的视图中查询得到（在 SQL*Plus 中通过命令 show sga 也可以看到同样结果）：

```
SQL> select * from v$sga;
NAME                 VALUE
-------------------  ----------
Fixed Size              731632
Variable Size         268435456
Database Buffers      117440512
Redo Buffers             811008
```

在 Oracle9i 中，Variable Size 包括 shared_pool_size，java_pool_size 和 large_pool_size 部分，SGA_MAX_SIZE 去除 db_cache_size 部分也被归入可变部分，所以很多时候我们看到的可变部分内存要远高于可变内存组件大小：

```
SQL> select
  2  (select value from v$parameter where name='large_pool_size') +
  3  (select value from v$parameter where name='shared_pool_size') +
  4  (select value from v$parameter where name='java_pool_size') Vsize
  5  from dual;
     VSIZE
----------
 134217728
```

Database Buffers 指 Buffer Cache 的设置：

```
SQL> show parameter db_cache_size
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_cache_size                        big integer       117440512
```

Redo Buffers 指日志缓冲区分配的内存大小，这个参数值通常比 log_buffers 参数设置略大：

```
SQL> show parameter log_buffer
```

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
log_buffer                           integer     524288
```

这是因为 Log Buffer 并非按照数据块大小分配，在内存中通常需要设置保护页对 Log Buffer 进行保护。

当前 SGA 的分配和使用具体信息我们还可以通过 V$SGASTAT 视图查询得到。大家可能也会注意到，在 V$SGASTAT 中显示的 Shared Pool 大小和 shared_pool_size 设置的仍然不同，这是因为在共享池内存的分配和使用过程中会存在一定量的额外消耗，这部分内存在 Oracle10g 中被单独列出：

```
[oracle@danaly ~]$ sqlplus "/ as sysdba"

SQL*Plus: Release 10.2.0.1.0 - Production on Wed Apr 12 18:37:41 2006
Copyright (c) 1982, 2005, Oracle.  All rights reserved.
SQL> select * from v$sgainfo;
NAME                                  BYTES RESIZEABLE
------------------------------ ---------- ----------
Fixed SGA Size                     1222744 No
Redo Buffers                       7163904 No
Buffer Cache Size                830472192 Yes
Shared Pool Size                  96468992 Yes
Large Pool Size                    4194304 Yes
Java Pool Size                     4194304 Yes
Streams Pool Size                        0 Yes
Granule Size                       4194304 No
Maximum SGA Size                 943718400 No
Startup overhead in Shared Pool   46137344 No
Free SGA Memory Available                0
```

此外，由于一些 Bug 的影响，可能 v$sgastat 视图显示的 Shared pool 会出现一些异常的情况，以下是曾经收到的一则异常报告。

数据库版本：

```
select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle9i Enterprise Edition Release 9.2.0.8.0 - Production
```

当前的 Shared_pool_size 设置为 80M：

```
shared_pool_reserved_size            integer  4194304
shared_pool_size                     integer  83886080
```

但是 v$sgastat 的显示出现了极度异常：

```
SELECT *
```

```
  FROM (SELECT   *
          FROM v$sgastat
         WHERE pool = 'shared pool'
      ORDER BY BYTES DESC)
 WHERE ROWNUM <= 5
POOL        NAME                            BYTES
----------- -------------------------- ----------
shared pool session param values       4293530440
shared pool free memory                  67550256
shared pool miscellaneous                27355660
shared pool XDB Schema Cac                3889744
shared pool library cache                 3709176
5 rows selected.
```

这里的 session param values 消耗达到了 4G，当然这不可能是真正的内存消耗，Oracle 的某个 Bug 导致了显示异常，这个 Bug 在 Oracle 10.2 中被修正。所以通常 v$sgastat 不能用来作为计算 Shared Pool 的依据。

5.1.2 SGA 与共享内存

SGA 的设置在 Linux/Unix 上和一个操作系统内核参数有关，这个参数是：shmmax。不同操作系统，该参数设置的位置不同，在 Solaris 上，该参数由/etc/system 文件中 shmsys:shminfo_shmmax 定义；在 Linux 上，该参数由/proc/sys/kernel/shmmax 参数定义。

很多人将该参数理解为共享内存的大小，这是不对的。实际上 shmmax 内核参数定义的是系统允许的单个共享内存段的最大值，如果该参数设置小于 Oracle SGA 设置，那么 SGA 仍然可以创建成功，但是会被分配多个共享内存段。我们通常推荐通过调整 shmmax 设置，将 SGA 限制在一个共享内存段中。

在 Windows 系统中，由于系统采用多线程服务器（所有 oracle server process 实际上都是一个进程中的线程），所以不存在共享内存的问题，无需进行特殊设置。

以 32 位 Linux 平台为例来看一下 shmmax 参数对于数据库的影响。

Linux 上该参数的缺省值通常为 32M。

```
[root@neirong root]# more /proc/sys/kernel/shmmax
33554432
```

本例的操作系统版本为：

```
[root@neirong root]# cat /etc/redhat-release
Red Hat Enterprise Linux AS release 3 (Taroon Update 2)
[root@neirong root]# uname -r
2.4.21-15.ELsmp
```

可以通过 ipcs 命令查看此设置下共享内存的分配，我们可以看到 Oracle 分配了多个共享内存段以满足 SGA 设置的需要:

```
[root@neirong root]# ipcs -sa

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 884736     oracle     640        4194304    14
0x00000000 917505     oracle     640        33554432   14
0x00000000 950274     oracle     640        33554432   14
0x00000000 983043     oracle     640        33554432   14
0x00000000 1015812    oracle     640        33554432   14
0x00000000 1048581    oracle     640        33554432   14
0x00000000 1081350    oracle     640        33554432   14
0x00000000 1114119    oracle     640        33554432   14
0x00000000 1146888    oracle     640        33554432   14
0x00000000 1179657    oracle     640        33554432   14
0x00000000 1212426    oracle     640        33554432   14
0x00000000 1245195    oracle     640        33554432   14
0x00000000 1277964    oracle     640        33554432   14
0x00000000 1310733    oracle     640        33554432   14
0x00000000 1343502    oracle     640        33554432   14
0x00000000 1376271    oracle     640        33554432   14
0x00000000 1409040    oracle     640        33554432   14
0x00000000 1441809    oracle     640        33554432   14
0x00000000 1474578    oracle     640        33554432   14
0x00000000 1507347    oracle     640        33554432   14
0x00000000 1540116    oracle     640        33554432   14
0x00000000 1572885    oracle     640        33554432   14
0x00000000 1605654    oracle     640        33554432   14
0x00000000 1638423    oracle     640        33554432   14
0x00000000 1671192    oracle     640        33554432   14
0x00000000 1703961    oracle     640        33554432   14
0x7a9c9900 1736730    oracle     640        4194304    56

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0xfcf02e10 229376     oracle     640        154

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```

通过以上输出看到为了创建 Oracle 的 SGA，系统共分配了 27 个共享内存段。接下来针对

一个后台进程，使用 pmap 工具查看一下每个共享内存段的地址空间：

```
[root@neirong root]# ps -ef|grep dbw
oracle    3102     1  0 09:27 ?        00:00:26 ora_dbw0_hsmkt
root      7018  6923  0 15:48 pts/1    00:00:00 grep dbw

[root@neirong root]# pmap 3102
ora_dbw0_hsmkt[3102]
08048000 (37308 KB)    r-xp (68:06 1525072)  /opt/oracle/product/9.2.0/bin/oracle
0a4b7000 (8804 KB)     rw-p (68:06 1525072)  /opt/oracle/product/9.2.0/bin/oracle
0ad50000 (380 KB)      rw-p (00:00 0)
50000000 (4096 KB)     rw-s (00:04 884736)   /SYSV00000000
51000000 (32768 KB)    rw-s (00:04 917505)   /SYSV00000000
```

注意：这里的 884736、917505 等就是 ipcs 里所看到的共享内存 ID（shmid）。

```
53000000 (32768 KB)    rw-s (00:04 950274)   /SYSV00000000
55000000 (32768 KB)    rw-s (00:04 983043)   /SYSV00000000
57000000 (32768 KB)    rw-s (00:04 1015812)  /SYSV00000000
59000000 (32768 KB)    rw-s (00:04 1048581)  /SYSV00000000
5b000000 (32768 KB)    rw-s (00:04 1081350)  /SYSV00000000
5d000000 (32768 KB)    rw-s (00:04 1114119)  /SYSV00000000
5f000000 (32768 KB)    rw-s (00:04 1146888)  /SYSV00000000
61000000 (32768 KB)    rw-s (00:04 1179657)  /SYSV00000000
63000000 (32768 KB)    rw-s (00:04 1212426)  /SYSV00000000
65000000 (32768 KB)    rw-s (00:04 1245195)  /SYSV00000000
67000000 (32768 KB)    rw-s (00:04 1277964)  /SYSV00000000
69000000 (32768 KB)    rw-s (00:04 1310733)  /SYSV00000000
6b000000 (32768 KB)    rw-s (00:04 1343502)  /SYSV00000000
6d000000 (32768 KB)    rw-s (00:04 1376271)  /SYSV00000000
6f000000 (32768 KB)    rw-s (00:04 1409040)  /SYSV00000000
71000000 (32768 KB)    rw-s (00:04 1441809)  /SYSV00000000
73000000 (32768 KB)    rw-s (00:04 1474578)  /SYSV00000000
75000000 (32768 KB)    rw-s (00:04 1507347)  /SYSV00000000
77000000 (32768 KB)    rw-s (00:04 1540116)  /SYSV00000000
79000000 (32768 KB)    rw-s (00:04 1572885)  /SYSV00000000
7b000000 (32768 KB)    rw-s (00:04 1605654)  /SYSV00000000
7d000000 (32768 KB)    rw-s (00:04 1638423)  /SYSV00000000
7f000000 (32768 KB)    rw-s (00:04 1671192)  /SYSV00000000
81000000 (32768 KB)    rw-s (00:04 1703961)  /SYSV00000000
83000000 (4 KB)        r--s (00:04 1736730)  /SYSV7a9c9900
83001000 (644 KB)      rw-s (00:04 1736730)  /SYSV7a9c9900
```

```
830a2000 (4 KB)        r--s (00:04 1736730)  /SYSV7a9c9900
830a3000 (3444 KB)     rw-s (00:04 1736730)  /SYSV7a9c9900
。。。。。。。。。。。。。。。。。。
mapped: 881332 KB       writable/private: 12056 KB      shared: 827392 KB
```

为了避免多个共享内存段，可以修改 shmmax 内核参数，使 SGA 存在于一个共享内存段中。通过修改/proc/sys/kernel/shmmax 参数可以达到此目的。

以下是一个生产环境中的设置：

```
[root@danaly root]# echo 1610612736 > /proc/sys/kernel/shmmax
[root@danaly root]# more /proc/sys/kernel/shmmax
1610612736
```

这里修改为 1.5G。对于 shmmax 文件的修改，系统重新启动后会复位。可以通过修改/etc/sysctl.conf 文件使更改永久化。在该文件内添加以下一行，这个更改在系统重新启动后生效：

```
kernel.shmmax = 1610612736
```

修改 shmmax 之后，需要重起数据库使更改生效。

查看一下这个 Oracle10g 生产环境的共享内存分配情况，Oracle 用户主要分配了两个共享内存段，这两个内存段分别为 ASM 实例和数据库实例所使用：

```
[root@danaly ~]# ipcs -sa
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x4fb9f94c 98304      oracle    640        85983232   12
0x6e9259ec 196609     oracle    640        945815552  30
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0xb1ab1ddc 491520     oracle    640        44
0x975a71dc 884737     oracle    640        154
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```

可以找到两个实例的一个后台进程用于分析：

```
[root@danaly ~]# ps -ef|grep dbw
oracle    7091     1  0 Aug14 ?        00:00:01 asm_dbw0_+ASM
oracle    8201     1  0 Aug14 ?        01:01:06 ora_dbw0_danaly
root     17309 16630  0 16:19 pts/2    00:00:00 grep dbw
```

数据库实例 DBWR 的进程地址空间使用如下所示：

```
[root@danaly ~]# more /proc/8201/maps
0041c000-0042e000 r-xp 00000000 68:03 75361      /lib/libnsl-2.3.4.so
0042e000-00430000 rw-p 00011000 68:03 75361      /lib/libnsl-2.3.4.so
.................
009c2000-009c4000 rw-p 009c2000 00:00 0
```

```
08048000-0cb72000 r-xp 00000000 68:06 968117     /opt/oracle/product/10.2.0/bin/oracle
0cb72000-0cbc3000 rw-p 04b2a000 68:06 968117     /opt/oracle/product/10.2.0/bin/oracle
0cbc3000-0cc53000 rw-p 0cbc3000 00:00 0
20000000-58600000 rw-s 00000000 00:06 196609     /SYSV6e9259ec (deleted)
.....................
bfff8000-c0000000 rwxp bfff8000 00:00 0
ffffe000-fffff000 ---p 00000000 00:00 0
```

这里的 196609 正是 ipcs 中看到的一个共享内存段，也就是数据库实例所使用的共享内存。这段进程地址空间从 **20000000-58600000**，大约是 902M 的空间，这正好是数据库的 SGA 使用的内存空间：

```
SQL> show sga
Total System Global Area  943718400 bytes
Fixed Size                  1222744 bytes
Variable Size             109053864 bytes
Database Buffers          826277888 bytes
Redo Buffers                7163904 bytes
SQL> select 943718400/1024/1024 from dual;
943718400/1024/1024
-------------------
                900
```

另外一块共享内存分配给了 ASM 实例：

```
[root@danaly ~]# more /proc/7091/maps|grep deleted
20000000-25200000 rw-s 00000000 00:06 98304      /SYSV4fb9f94c (deleted)
```

这段地址空间是 82M，也正是 ASM 实例所使用的内存空间：

```
[oracle@danaly ~]$ export ORACLE_SID=+ASM
[oracle@danaly ~]$ sqlplus "/ as sysdba"
SQL*Plus: Release 10.2.0.1.0 - Production on Mon Sep 17 16:33:29 2007
Copyright (c) 1982, 2005, Oracle.  All rights reserved.
SQL> show sga
Total System Global Area   83886080 bytes
Fixed Size                  1217836 bytes
Variable Size              57502420 bytes
ASM Cache                  25165824 bytes
SQL> select 83886080/1024/1024 from dual;
83886080/1024/1024
------------------
                80
```

通常，如果没有修改 shmmax 参数，Oracle 在启动过程中就会在告警日志文件中记录如下警告：

```
Starting ORACLE instance (normal)
Thu Nov 17 09:27:29 2005
WARNING: EINVAL creating segment of size 0x0000000033400000
fix shm parameters in /etc/system or equivalent
```

这是一个 WARNING 的提示，说明是建议修正，但并非强制的内容。在 Solaris 平台上，有时候也会看到类似的警报：

```
Sun Apr 30 05:35:20 2006
Starting ORACLE instance (normal)
Sun Apr 30 05:35:20 2006
WARNING: Not enough physical memory for SHM_SHARE_MMU segment of size 0x000000006d400000
[flag=0x4000]
```

这通常是因为 SGA 设置过大，超过了物理内存而导致的，这种情况通过修正参数即可解决。有时候这类警告也可能是因为数据库异常关闭，后台进程未正常退出，共享内存未及时释放引起的，对于这种情况，可以通过 ipcs 命令找到共享内存段 id（shared memory id），然后通过 ipcrm 命令可以强制释放该共享内存段，完成这些特殊处理后，数据库通常就可以正常启动了。

5.1.3 SGA 管理的变迁

在 Oracle9i 之前，SGA 一直是静态分配内存的。SGA 分配的内存空间可以被所有的 Oracle 进程/线程所共享，内存的大小是根据 init.ora 参数文件中的值计算得来的，一旦分配完毕，可用共享内存的大小就不能增大或缩小，如果 DBA 要增加数据库块缓冲区的数量，必须首先关闭例程修改初始化参数文件，然后重新启动该例程。

让我们从 Oracle8i 开始，研究一下 Oracle SGA 管理的变迁。

1. Oracle8i 中静态 SGA 的管理

我们知道在 Oracle8i 之中,当我们需要修改 SGA 参数时(例如:SHARED_POOL_SIZE),必需修改参数文件，重新启动数据库之后,修改才能生效.

我们可以从参数文件中找到这些重要参数，下面是一个 Oracle8i 数据库的设置示例：

```
db_block_buffers = 25000
shared_pool_size = 104857600
large_pool_size = 8388608
java_pool_size = 10485760
log_buffer = 163840
```

当然这些参数的设置要受物理内存的限制，需要全面考虑。

2. Oracle9i 动态 SGA 管理

从 Oracle9i 开始,Oracle 推出了动态 SGA 调整,也就是说,允许我们不重新启动数据库而使得 SGA 的修改生效。

在 Oracle9i 中,我们可以设置参数 SGA_MAX_SIZE,该参数用以控制各缓冲池使用的内存总和,本质上是在进程中预先分配一段虚拟地址备用而不分配物理内存，目的是防止和进程私有地址段的冲突：

```
SQL> show parameter sga_max_size
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
sga_max_size                         big integer      387418608
```

设置了该参数之后,我们可以通过在线方式修改 Oracle SGA 各内存组件的内存分配,经常我们可能用到类似如下命令(关于 Scope 参数等说明可以参考上一章的内容):

```
alter system set db_cache_size = 2g scope=memory;
alter system set large_pool_size = 200m scope=memory;
alter system set java_pool_size = 200m scope=memory;
```

只要总的 SGA 内存设置不超过 SGA_MAX_SIZE 的设置,更改都可以立即生效（但是需要注意的是，在 Oracle9iR1 中，动态减小内存设置可能会触发一些 Bug，在繁忙的生产系统中，缩减各组件的内存使用应该是相当慎重的）:

```
SQL> alter system set db_cache_size=100M;
System altered.
```

如果内存不足,则 Oracle 会给出错误,提示内存缺乏:

```
SQL> alter system set db_cache_size=500M;
alter system set db_cache_size=500M
*
ERROR at line 1:
ORA-02097: parameter cannot be modified because specified value is invalid
ORA-00384: Insufficient memory to grow cache
```

当然在动态修改这些参数时，存在一些常见的限制：

1．修改的内存大小必须是粒度（granule）大小的整数倍，否则会自动向上取整

2．SGA 总大小不能超过 SGA_MAX_SIZE

3．SGA 最低配置为三个粒度（granule）一个粒度用于固定的 SGA（包括重做缓冲区），一个粒度用于缓冲区高速缓存，一个粒度用于共享池。

通过 OEM，我们可以直观的了解一下 Oracle9i SGA 各部分参赛设置：

伴随动态 SGA 管理的新特性，Oracle 推出了一系列内存设置建议功能，同时引入了一系列动态性能视图：

```
SQL> select tname from tab where tname like '%ADVICE%';
TNAME
------------------------------
GV_$DB_CACHE_ADVICE
GV_$MTTR_TARGET_ADVICE
GV_$PGATARGET_ADVICE_HISTOGRAM
GV_$PGA_TARGET_ADVICE
GV_$SHARED_POOL_ADVICE
V_$DB_CACHE_ADVICE
V_$MTTR_TARGET_ADVICE
V_$PGA_TARGET_ADVICE
V_$PGA_TARGET_ADVICE_HISTOGRAM
V_$SHARED_POOL_ADVICE
10 rows selected.
```

其中和 SGA 相关的是 V$DB_CACHE_ADVICE 和 V$SHARED_POOL_ADVICE，这些新功能通过在数据库运行时持续不断的收集信息，从而对内存的设置提供建议。

缓冲区高速缓存建议(buffer cache advisory) 受初始化参数 DB_CACHE_ADVICE 控制。该参数为动态参数，可用的值有三个 OFF、ON 和 READY。

DB_CACHE_ADVICE 不同参数值的含义分别如下：

OFF-关闭建议并且不为建议分配内存

ON-开启建议并且 CPU 和内存开销都会发生

READY-关闭建议但是仍保留为建议分配的内存

在某些版本中，如果在参数为 OFF 状态时尝试将其设置为 ON 可能会出现 ORA-4031 错误，无法从共享池中分配内存；如果参数处于 READY 状态则可以将其设置为 ON 而不会发生错误，这是因为需要的内存已经分配。

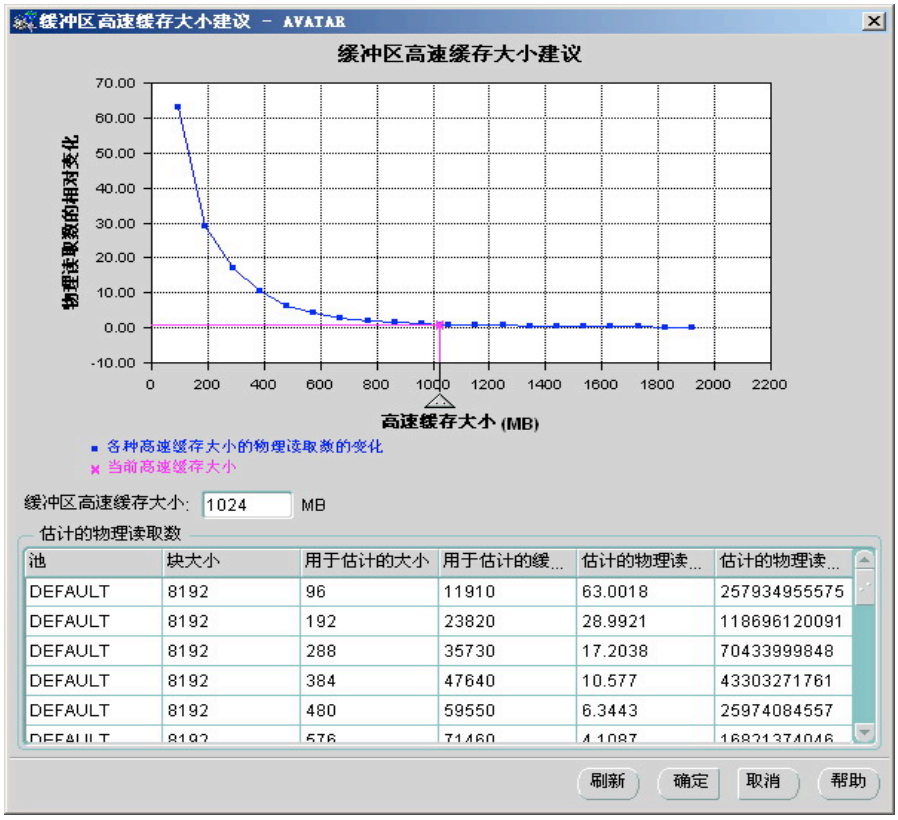
我们看一下一个生产数据库中，Oracle 收集的 buffer cache 建议信息：

```
SQL> show parameter db_cache_ad
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_cache_advice                      string      ON
SQL>  select id,name,block_size,size_for_estimate sfe,size_factor sf,
  2  estd_physical_read_factor eprf,estd_physical_reads epr
  3  from v$db_cache_advice;
        ID NAME                  BLOCK_SIZE        SFE         SF    EPRF     EPR
---------- -------------------- ---------- ---------- ---------- ---------- ----------
         3 DEFAULT                    8192         96     0.0938    63.0018 2579371363
         3 DEFAULT                    8192        192     0.1875    28.9921 1186971632
         3 DEFAULT                    8192        288     0.2813    17.2038 7043452876
         3 DEFAULT                    8192        384      0.375     10.577 4330360330
         3 DEFAULT                    8192        480     0.4688     6.3443 2597429002
         3 DEFAULT                    8192        576     0.5625     4.1087 1682151293
         3 DEFAULT                    8192        672     0.6563      2.897 1186065595
         3 DEFAULT                    8192        768       0.75     2.1604 8844976015
         3 DEFAULT                    8192        864     0.8438     1.5728 6439364907
         3 DEFAULT                    8192        960     0.9375     1.1605 4751342947
         3 DEFAULT                    8192       1024          1          1
4094120994
         3 DEFAULT                    8192       1056     1.0313      0.937 3836222877
         3 DEFAULT                    8192       1152      1.125     0.7783 3186337904
         3 DEFAULT                    8192       1248     1.2188     0.6519 2669051919
         3 DEFAULT                    8192       1344     1.3125      0.534 2186279350
         3 DEFAULT                    8192       1440     1.4063     0.4296 1758909643
         3 DEFAULT                    8192       1536        1.5     0.3529 1444916209
```

```
         3 DEFAULT                8192       1632       1.5938     0.2946 1206059096
         3 DEFAULT                8192       1728       1.6875     0.2416  989138472
         3 DEFAULT                8192       1824       1.7813     0.2031  831457344
         3 DEFAULT                8192       1920        1.875     0.1712  700889664
21 rows selected
```

我们可以看到，伴随 db_cache_size 的增大，估计的物理读（estd_physical_reads）在逐渐减少，我们的选择就在于在 db_cache_size 的设置和 physical_reads 之间寻找一个边际效益最高点，使用可以接受的内存设置,获得尽量低的物理读。

在 Oracle9i 中,OEM 中提供了对以上数据的图形化展现,我们可以清楚的看到趋势曲线:



而对于 Shred Pool 的建议则受到另外一个初始化参数的影响，这个参数是:STATISTICS_LEVEL。STATISTICS_LEVEL 控制数据库收集的统计信息的级别,该参数有三个选项：

- BASIC-收集基本的统计信息
- TYPICAL-收集大部分的统计信息,这是系统的缺省设置,为了从 Oracle 不断增加的新特性中受益,始终应该将该参数设置为典型.
- ALL-收集全部的统计信息.

可以通过 v$statistics_level 视图来查看该参数的影响范围，在 Oracle11g 中该视图中的条目已经增加到 24 个：

```
SQL>select STATISTICS_NAME,SESSION_STATUS,
```

```
  2 SYSTEM_STATUS,ACTIVATION_LEVEL,SESSION_SETTABLE from v$statistics_level;
STATISTICS_NAME                          SESSION_ SYSTEM_S ACTIVAT SES
---------------------------------------- -------- -------- ------- ---
Buffer Cache Advice                      ENABLED  ENABLED  TYPICAL NO
MTTR Advice                              ENABLED  ENABLED  TYPICAL NO
Timed Statistics                         ENABLED  ENABLED  TYPICAL YES
Timed OS Statistics                      DISABLED DISABLED ALL     YES
Segment Level Statistics                 ENABLED  ENABLED  TYPICAL NO
PGA Advice                               ENABLED  ENABLED  TYPICAL NO
Plan Execution Statistics                DISABLED DISABLED ALL     YES
Shared Pool Advice                       ENABLED  ENABLED  TYPICAL NO
Modification Monitoring                  ENABLED  ENABLED  TYPICAL NO
Longops Statistics                       ENABLED  ENABLED  TYPICAL NO
Bind Data Capture                        ENABLED  ENABLED  TYPICAL NO
Ultrafast Latch Statistics               ENABLED  ENABLED  TYPICAL NO
Threshold-based Alerts                   ENABLED  ENABLED  TYPICAL NO
Global Cache Statistics                  ENABLED  ENABLED  TYPICAL NO
Active Session History                   ENABLED  ENABLED  TYPICAL NO
Undo Advisor, Alerts and Fast Ramp up    ENABLED  ENABLED  TYPICAL NO
Streams Pool Advice                      ENABLED  ENABLED  TYPICAL NO
Time Model Events                        ENABLED  ENABLED  TYPICAL YES
Plan Execution Sampling                  ENABLED  ENABLED  TYPICAL YES
Automated Maintenance Tasks              ENABLED  ENABLED  TYPICAL NO
SQL Monitoring                           ENABLED  ENABLED  TYPICAL YES
Adaptive Thresholds Enabled              ENABLED  ENABLED  TYPICAL NO
V$IOSTAT_* statistics                    ENABLED  ENABLED  TYPICAL NO
Session Wait Stack                       ENABLED  ENABLED  TYPICAL NO

24 rows selected.
```

可以看到在 TYPICAL 设置下，除 Timed OS Statistics 和 Plan Execution Statistics 信息不收集外,其他信息都被收集。其中, Buffer Cache Advice 受 db_cache_advice 参数独立控制, Timed Statistics 受 timed_statistics 参数独立控制。其他统计信息的收集都受到 STATISTICS_LEVEL 参数的控制。当 STATISTICS_LEVEL 为 Basic 时，除受独立参数影响的 **Buffer Cache Advice** 和 **Timed Statistics** 外，其他信息收集都将被禁止。

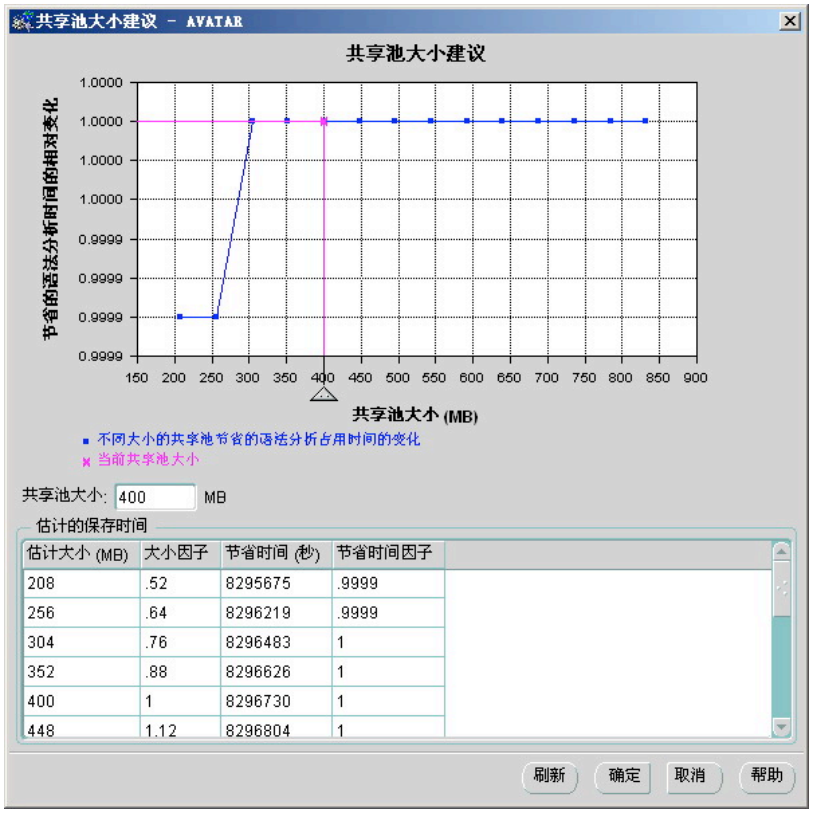
我们可以通过查询 V$SHARED_POOL_ADVICE 视图获得关于 Shared Pool 的建议信息：

```
SQL> select SHARED_POOL_SIZE_FOR_ESTIMATE SPSFE,SHARED_POOL_SIZE_FACTOR SPSF,
  2  ESTD_LC_SIZE,ESTD_LC_MEMORY_OBJECTS ELMO,ESTD_LC_TIME_SAVED ELTS,
  3  ESTD_LC_TIME_SAVED_FACTOR ELTSF,ESTD_LC_MEMORY_OBJECT_HITS ELMOH
  4  from v$shared_pool_advice;
```

SPSFE	SPSF	ESTD_LC_SIZE	ELMO	ELTS	ELTSF	ELMOH
208	0.52	193	17972	8295638	0.9999	359670096
256	0.64	240	22083	8296182	0.9999	359708386
304	0.76	289	26310	8296446	1	359725903
352	0.88	336	30330	8296589	1	359735662
400	1	383	34632	8296693	1	359745443
448	1.12	430	42078	8296767	1	359752957
496	1.24	479	50650	8296816	1	359757737
544	1.36	526	59499	8296855	1	359761517
592	1.48	573	68609	8296891	1	359764717
640	1.6	620	77522	8296930	1	359767727
688	1.72	667	85969	8296953	1	359769641
736	1.84	716	96051	8296968	1	359770920
784	1.96	764	105940	8296987	1	359772425
832	2.08	795	112434	8296992	1	359772998

14 rows selected

通过以上数据可以得知，当前的 shared_pool_size 大小为 400M（大小因子为 1 的是当前设置），通过 OEM，也可以直观的看到图形显示：



通过以上统计数据分析，当 shared_pool_size 设置为 304M 时即可达到和现在相同的效果，目前的 shared_pool_size 设置浪费了部分内存，那么我们就可以动态调整 shared_pool_size 参数，释放这部分内存，留给其他内存组件使用。

```
SQL> alter system set shared_pool_size=304M;
```

当进行动态参数修改时，修改 Session 会处于等待状态，等待事件为 background parameter adjustment：

```
SQL> select sid,seq#,event,SECONDS_IN_WAIT,state
  2  from v$session_wait where sid=80;
     SID  SEQ#  EVENT                                          SECONDS_IN_WAIT STATE
---------- ---------- ----------------------------------------------------------------
80   46479   background parameter adjustment                              928  WAITING
```

这个调整的时间可能极其漫长，从 v$lock 视图中，我们还可以获得相关锁定信息：

```
SQL> select * from v$lock where sid=80;
ADDR             KADDR                    SID TYPE         ID1         ID2       LMODE
REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- ---- ---------- ---------- ---------- ----------
---------- ----------
00000003CF3D6048 00000003CF3D6068          80 PE             44          0          4
0       1437          0
```

锁定类型为 PE，即 Kernel Service system **P**arameters **E**NQUEUE，在修改系统参数时需要获取该锁定。

需要提醒的是，虽然 Oracle9i 中，Oracle 提供了动态内存修改的功能，但是仍然建议在系统规划时做好设置，尽量避免运行时的动态调整。动态调整某些系统参数(如 undo_retention 等)在繁忙的系统中可能触发 bug 而造成系统挂起。

3. Oracle10g 自动共享内存管理

很多人可能注意到,Oracle9i 的动态 SGA 调整的新特性虽然方便，但是仍然需要 DBA 去观察并修改这些设置,如果 Oracle 能够自动完成这个调整,那将是一个划时代的进步了.

很快,在 Oracle10g 中,这个梦想得以实现.

在 Oracle10g 之前，你可能面对过这样的情况，你的数据库在白天需要处理大量的 OLTP 任务，这些任务需要大量的 Buffer Cache 内存，而在夜间系统可能需要运行大量的并行批处理任务，这些任务又需要大量的 Large Pool 内存，为了让这样一个系统在有限的资源下高效率运行，可能你需要在各类峰值业务来临之前对数据库进行不断的调整。

那么在 Oracle10g 的自动共享内存管理(Automatic Shared Memory Management-ASMM)下，这些动作不再需要人工介入，当运行 OLTP 任务时，Buffer Cache 会获取大部分内存以达到良好的 I/O 性能。当系统需要运行 DSS 批处理任务时，内存会自动转移给 Large Pool，以便并行查询等可以获得更多的内存，更快的执行。

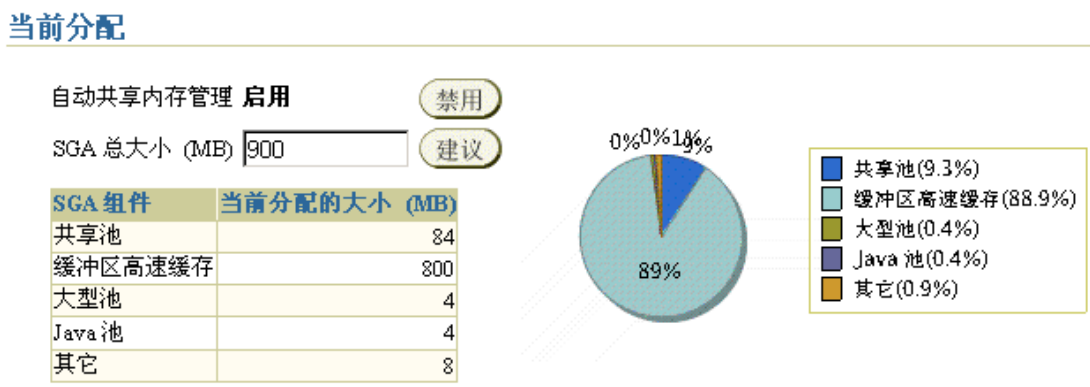
Oracle10g 使用了一个新的初始化参数 SGA_TARGET，通过指定这个参数，让 Oracle 自

动管理 SGA 中以下大多数的内存分配。SGA_TARGET 是个动态参数,但是该参数不能超过 SGA_MAX_SIZE 参数的设置，如果试图修改 SGA_TARGET 超越 SGA_MAX_SIZE 的限制,系统会给出错误信息:

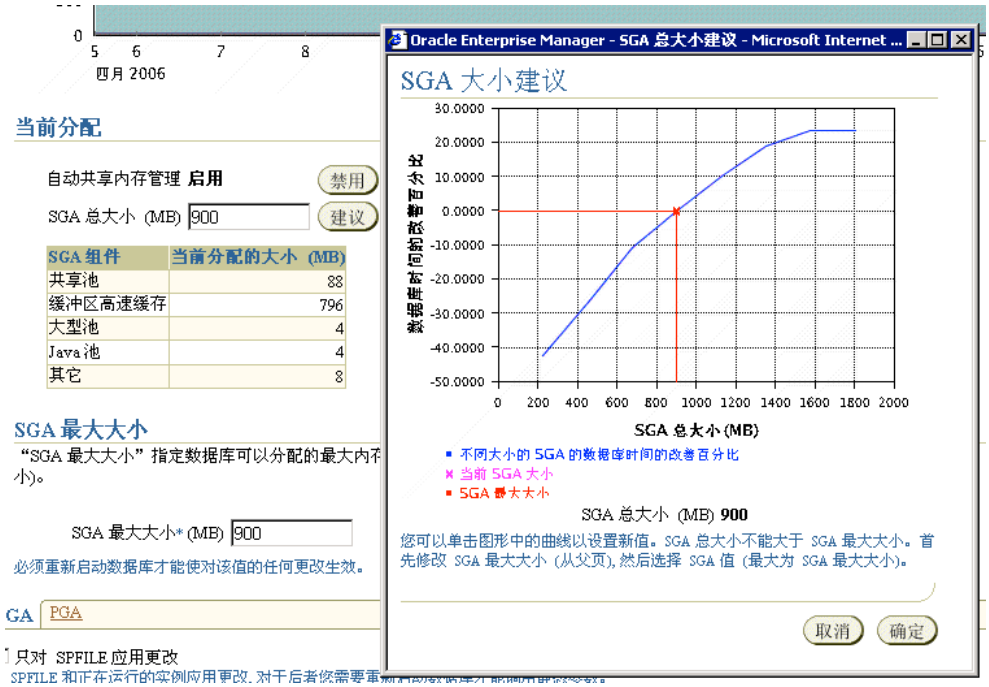
```
SQL> show parameter sga_max
NAME                                 TYPE        VALUE
------------------------------------ ----------- -----
sga_max_size                         big integer 300M
SQL> alter system set sga_target=400M;
alter system set sga_target=400M
*
ERROR at line 1:
ORA-02097: parameter cannot be modified because specified value is invalid
ORA-00823: Specified value of sga_target greater than sga_max_size
```

并非所有 SGA 组件都可以自动调整,可以自动分配的内存包括 Buffer Cache、Shared Pool、Java Pool、 Large Pool

启用自动共享内存管理,我们可以估算一个 SGA 的总大小，然后设置 SGA_TARGET 参数为非零值, Oaracle 将启用自动共享内存管理。自动共享内存管理需要 STATISTICS_LEVEL 参数设置为 TYPICAL 或者 ALL.。可以通过 Oracle10g 的 Database Control 来启用或禁用自动共享内存管理：



为了更加直观，Oracle 还在 Database Control 中提供了图形展现,提供建议数据：



Oracle 服务器根据系统运行的情况自动调整这些内存的大小，并记录在 SPFILE 中，数据库实例重新启动时，不会丢失之前的调整结果。

但是注意以下相关初始化参数还是需要手工配置的： 非标准 BLOCK_SIZE 的 Cache、Keep/Recycle Buffer Cache、Redo Log Buffer、Streem Pool

自动的共享内存管理引入了一个新的后台进程:MMAN(Memory Manager)。该进程用以动态调整内存组件。动态调整的依据来自系统不间断收集的内存建议。

从告警日志文件中，可以看到该进程启动顺序为 3,进程号为 4：

```
PMON started with pid=2, OS id=4464
PSP0 started with pid=3, OS id=4466
MMAN started with pid=4, OS id=4468
DBW0 started with pid=5, OS id=4470
LGWR started with pid=6, OS id=4472
CKPT started with pid=7, OS id=4474
SMON started with pid=8, OS id=4476
RECO started with pid=9, OS id=4478
CJQ0 started with pid=10, OS id=4480
MMON started with pid=11, OS id=4482
```

如果不想使用自动共享内存管理的新特性，Oracle 也允许使用手工管理，只需要简单的将 SGA_TARGET 参数设置为 0,Oracle 就会回到手工管理的模式,当前的各内存组件值会被计入 spfile,做为手工管理的初始值使用.

随着自动共享内存管理新特性的引入，许多相关参数的使用也发生了改变。当我们从开始就设置了 SGA_MAX_SIZE 参数，启用了自动共享内存管理之后，相关内存参数值会处于未设置状态：

```
SQL> select name,value from v$parameter
  2  where name in  ('large_pool_size','java_pool_size',
  3                  'shared_pool_size','streams_pool_size','db_cache_size');
NAME                                 VALUE
------------------------------------ ------------------------------
shared_pool_size                     0
large_pool_size                      0
java_pool_size                       0
streams_pool_size                    0
db_cache_size                        0
```

而真正决定各组件大小的，是由一组新引入的参数决定：

```
SQL> SELECT x.ksppinm NAME, y.ksppstvl VALUE, x.ksppdesc describ
  2    FROM SYS.x$ksppi x, SYS.x$ksppcv y
  3   WHERE x.inst_id = USERENV ('Instance') AND y.inst_id = USERENV ('Instance')
  4     AND x.indx = y.indx AND x.ksppinm like '%pool_size%'z;
NAME                           VALUE          DESCRIB
------------------------------ -------------- --------------------------------
_NUMA_pool_size             Not specified   aggregate size in bytes of NUMA pool
__shared_pool_size             96468992        Actual size in bytes of shared pool
shared_pool_size               0               size in bytes of shared pool
__large_pool_size              4194304         Actual size in bytes of large pool
large_pool_size                0               size in bytes of large pool
__java_pool_size               4194304         Actual size in bytes of java pool
java_pool_size                 0               size in bytes of java pool
__streams_pool_size             0               Actual size in bytes of streams
pool
streams_pool_size              0               size in bytes of the streams pool
```

这些由两个下划线开头的参数决定了当前 SGA 的分配，也是动态内存管理调整的参数。这些参数的更改会被记录到 spfile 文件当中，在下一次数据库启动时仍然有效.

我们转储一下 spfile 文件:

```
SQL> show parameter spfile
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
spfile                               string      +ORADG/danaly/spfiledanaly.ora
SQL> create pfile from spfile;
File created.
```

检查其参数设置，可以清晰地看到这些参数及其参数值：

```
[oracle@danaly dbs]$ more initdanaly.ora
danaly.__db_cache_size=830472192
danaly.__java_pool_size=4194304
danaly.__large_pool_size=4194304
danaly.__shared_pool_size=96468992
danaly.__streams_pool_size=0
*.audit_file_dest='/opt/oracle/admin/danaly/adump'
*.background_dump_dest='/opt/oracle/admin/danaly/bdump'
*.compatible='10.2.0.1.0'
*.control_files='+ORADG/danaly/controlfile/current.256.600173845','+ORADG/danaly/controlf
ile/current.257.600173845'
。。。。。。。。。。。。。
*.user_dump_dest='/opt/oracle/admin/danaly/udump'
```

这是 Oracle10g 在参数上的一些变化，通过动态视图 v$sga_dynamic_components，可以看到各动态组件调整的时间和调整类型等信息：

```
SQL> select COMPONENT,CURRENT_SIZE,MIN_SIZE,LAST_OPER_TYPE,
  2  LAST_OPER_MODE,to_char(LAST_OPER_TIME,'yyyy-mm-dd hh24:mi:ss') LOT
  3  from v$sga_dynamic_components where LAST_OPER_TIME is not null;
COMPONENT            CURRENT_SIZE   MIN_SIZE LAST_OPER_TYP LAST_OPER LOT
-------------------- ------------ ---------- ------------- --------- -------------------
shared pool             838860800  805306368 GROW          DEFERRED  2008-01-29 13:06:23
DEFAULT buffer cache   2315255808 2315255808 SHRINK        DEFERRED  2008-01-29 13:06:23
```

4. Oracle11g 自动内存管理

在 Oracle11g 中，Oracle 将内存管理的自动化更进了一步，引入了自动内存管理（Automatic Memory Management-AMM）的新特性，现在 Oracle 将 SGA 和 PGA 都纳入了自动管理的范畴。

在 Oracle10g 中，Oracle 对 SGA 引入了自动管理，相关的参数为 SGA_TARGET 和 SGA_MAX_SIZE；在 Oracle9i 中，Oracle 已经引入了自动的 PGA 管理，相关的初始化参数为 pga_aggregate_target。

综合这两个自动化特性，Oracle11g 引入了一个新的参数 MEMORY_TARGET。现在只要设置了这个参数，可以不需要设置 SGA_TARGET 和 PGA_AGGREATE_TARGET 两个参数，使用自动内存管理，Oracle 数据库将自行决定 SGA 以及 PGA 的分配和使用，这极大的简化了 DBA 对于内存的调整和管理工作。

```
SQL> show parameter memory_target
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
memory_target                        big integer 352M
```

```
SQL> show parameter memory_max_
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
memory_max_target                    big integer 352M
```

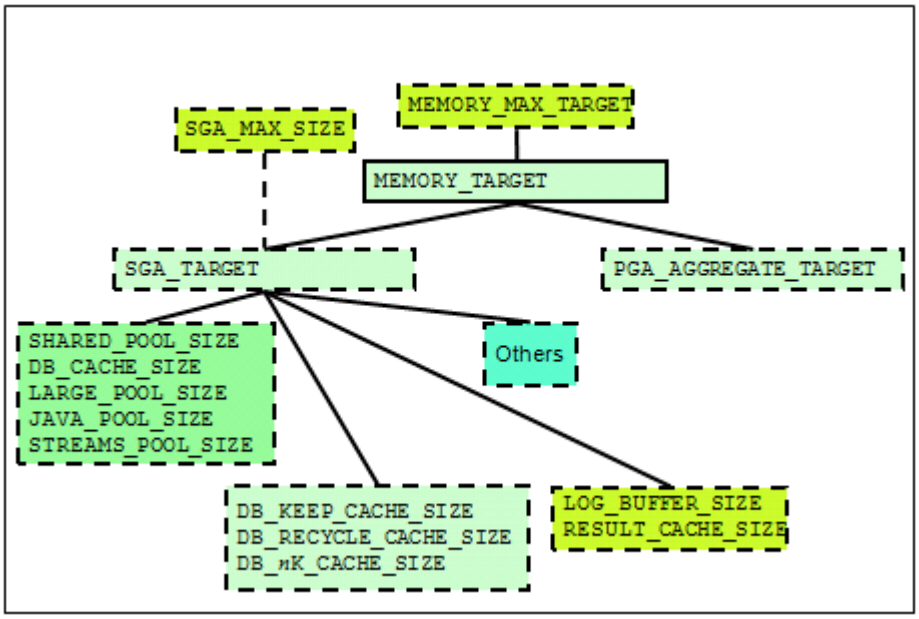
此时 PGA 和 SGA 参数可以无需设置：

```
SQL> show parameter pga
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
pga_aggregate_target                 big integer 0
SQL> show parameter sga_target
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
sga_target                           big integer 0
```

在参数上，自动内存管理和之前的自动 SGA 调整是类似的，Oracle 引入了带有两个下划线开头的参数用于实际控制内存设置，原有参数被设置为 0，这些参数在使用 SPFILE 时，可以动态在参数文件中改写，从而可以使得参数调整可以跨越数据库启动继续有效：

```
SQL> @GetHidPar
Enter value for par: sga_target
old   6:    AND x.ksppinm LIKE '%&par%'
new   6:    AND x.ksppinm LIKE '%sga_target%'
NAME                           VALUE                DESCRIB
------------------------------ -------------------- --------------------
sga_target                     0                    Target size of SGA
__sga_target                   222298112            Actual size of SGA
SQL> /
Enter value for par: pga_agg
old   6:    AND x.ksppinm LIKE '%&par%'
new   6:    AND x.ksppinm LIKE '%pga_agg%'
NAME                    VALUE     DESCRIB
------------------- --------- ----------------------------------------------------------------
pga_aggregate_target   0         Target size for the aggregate PGA memory consumed by the
                                 instance
__pga_aggregate_target 146800640 Current target size for the aggregate PGA memory consumed
```

随 MEMORY_TARGET 参数一起引入的还有 MEMORY_MAX_TARGET 参数，现在 ORACLE 的内存管理参数下图所示：



当 MEMORY_TARGET 参数设置超过了系统允许的限制后，会出现 ORA-00845 错误：

```
SYS@WAPDBS>alter system set memory_max_target=2048M scope=spfile;
System altered.
SYS@WAPDBS>alter system set memory_target=2048M scope=spfile;
System altered.
SYS@WAPDBS>startup force;
ORA-00845: MEMORY_TARGET not supported on this system
```

这个错误经常使人误解，实际上系统并非不支持 MEMORY_TARGET，而是不支持当前的参数设置。检查告警日志文件，在 Linux 上可以看到如下的提示信息：

```
Starting ORACLE instance (normal)
WARNING: You are trying to use the MEMORY_TARGET feature. This feature requires the /dev/shm
file system to be mounted for at least 2063597568 bytes. /dev/shm is either not mounted or
is mounted with available space less than this size. Please fix this so that MEMORY_TARGET
can work as expected. Current available is 1073741824 and used is 0 bytes.
memory_target needs larger /dev/shm
```

这是因为在 Linux 的 VLM（Very Large Memory）支持中，使用了 shmfs/tmpfs 选项（另外一种实现方式是使用（ramfs），如果/dev/shm 不足够，则会出现如上错误。

检查一下/dev/shm 设置：

```
[root@test126 /]# df -k /dev/shm
Filesystem           1K-blocks      Used Available Use% Mounted on
tmpfs                  1048576         0   1048576   0% /dev/shm
```

重新设置和加载一下/dev/shm：

```
[root@test126 /]# umount /dev/shm
```

```
[root@test126 /]# mount -t tmpfs shmfs -o size=3192M /dev/shm
[root@test126 /]# df -k /dev/shm
Filesystem           1K-blocks      Used Available Use% Mounted on
shmfs                  3268608         0   3268608   0% /dev/shm
[root@test126 /]# ls -l /dev/shm
total 0
```

设置完成之后，数据库可以正常启动：

```
[oracle@test126 admin]$ sqlplus "/ as sysdba"
SQL*Plus: Release 11.1.0.6.0 - Production on Tue Sep 18 11:31:53 2007
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
Connected to an idle instance.

SQL> startup
ORACLE instance started.

Total System Global Area 2058981376 bytes
Fixed Size                  1300968 bytes
Variable Size             939525656 bytes
Database Buffers         1107296256 bytes
Redo Buffers               10858496 bytes
Database mounted.
Database opened.
```

检查一下现在的 shm 内存使用：

```
[oracle@test126 admin]$ df -k /dev/shm
Filesystem           1K-blocks      Used Available Use% Mounted on
shmfs                  3268608   1244436   2024172  39% /dev/shm
```

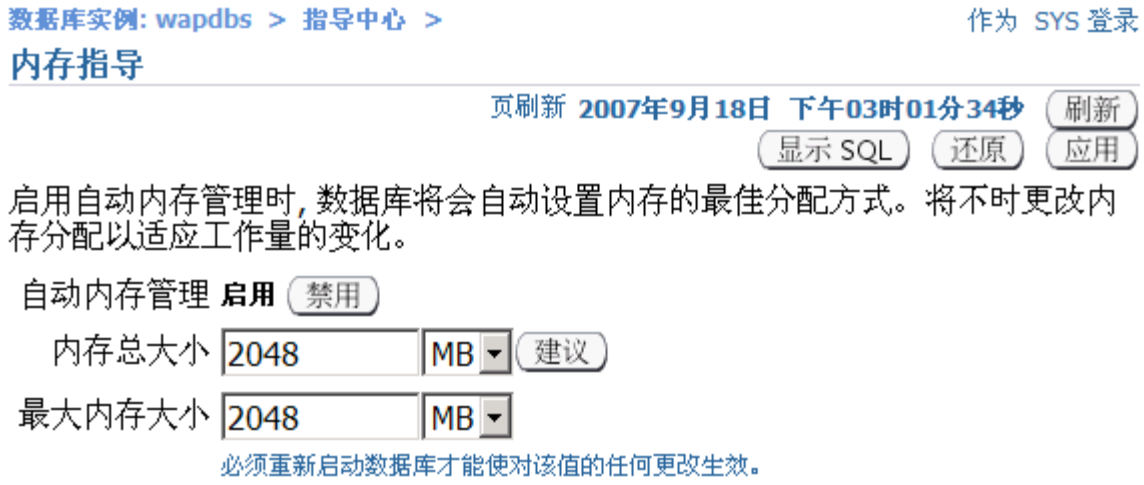
再看看内存段的使用情况：

```
[oracle@test126 admin]$ ls -l /dev/shm
total 1244436
-rw-r----- 1 oracle dba 16777216 Sep 18 11:31 ora_wapdbs_2457602_0
……
-rw-r----- 1 oracle dba 16777216 Sep 18 11:31 ora_wapdbs_2752517_30
-rw-r----- 1 oracle dba 16777216 Sep 18 11:32 ora_wapdbs_2752517_4
-rw-r----- 1 oracle dba 16777216 Sep 18 11:32 ora_wapdbs_2752517_5
-rw-r----- 1 oracle dba 16777216 Sep 18 11:32 ora_wapdbs_2752517_6
-rw-r----- 1 oracle dba 16777216 Sep 18 11:32 ora_wapdbs_2752517_7
-rw-r----- 1 oracle dba 16777216 Sep 18 11:32 ora_wapdbs_2752517_8
-rw-r----- 1 oracle dba 16777216 Sep 18 11:32 ora_wapdbs_2752517_9
```

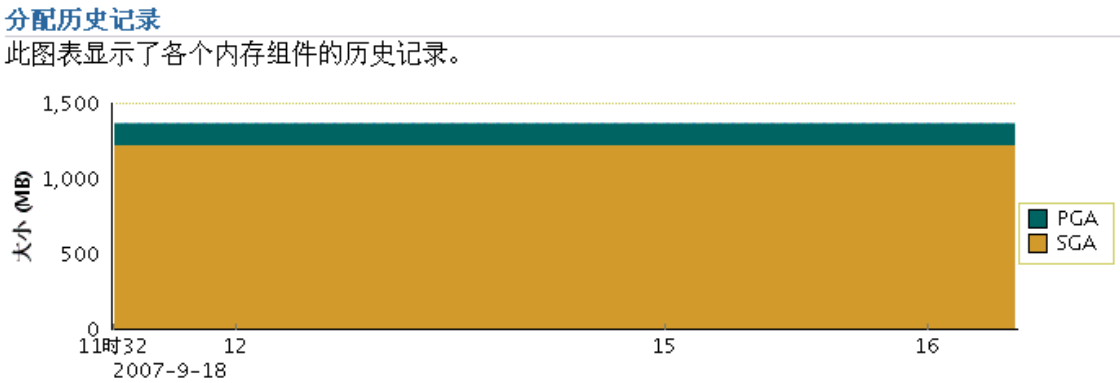
注意以上输出，Oracle 在系统上，以 Granule 为单位分配了内存映射文件（此时 Granule

的大小为 16M）。

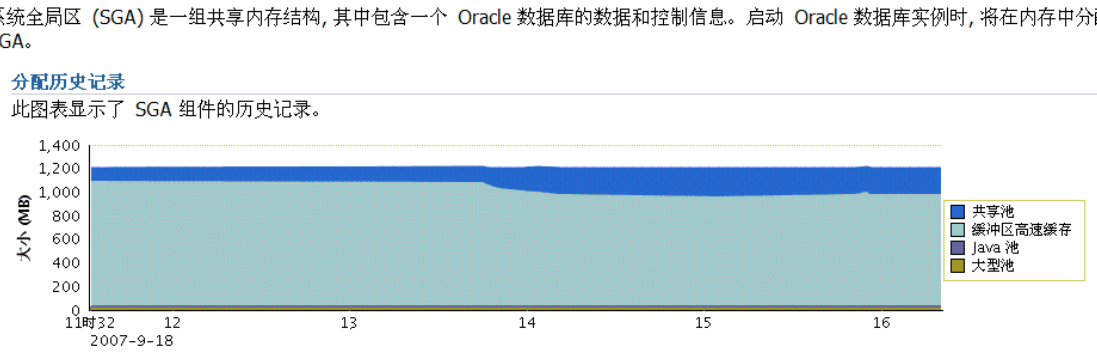
内存设置也可以通过 Database Control 来进行修改，下图是 Oracle Database 11g 中自动内存管理下图形显示：



SGA 和 PGA 的分配和变化可以通过图形界面直观的显示出来：



SGA 的各组件的内存分配和使用情况：



5.3 Oracle 的内存分配和使用

Oracle 数据库在系统占用的内存分内两个部分: SGA 和 PGA. 如何设置和规划 Oracle 的内存分配一直以来是一个广为争论的话题。

在 Oracle9i 之前，PGA 的内存使用估算一直比较复杂;从 Oracle9i 开始,Oracle 的自动 PGA 管理新特性使得 Oracle 的内存规划得以简化。

根据 Oracle 的建议,Oracle 最多可以使用 80% 的物理内存,其余 20% 保留给操作系统使用,在这 80%的内存中,对于 OLTP 系统,Oracle 建议分配 20%给 PGA 使用;对于 DSS 系统,可以分配 50%给 PGA 使用,再引述一下前文的等式:

1．对于 OLTP 系统

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

2．对于 DSS 系统

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

我们进一步归纳一下就是：

SGA + PGA <= <Total Physical Memory > * 80%

也就是：

SGA_MAX_SIZE + PGA_AGGREGATE_TARGET <= <Total Physical Memory > * 80%

现在 Oracle Database 11g 引入了自动内存管理，这个公式得以进一步简化：

MEMORY_TARGET<=MEMORY_MAX_TARGET<= <Total Physical Memory > * 80%

这是一个可以参考的数值，在为 Oracle 规划内存使用时，我们必须清楚，如果 Oracle 耗用的内存过高，甚至超过了系统的物理内存，那么系统的性能就会受到严重的影响，当系统执行任务时，如果没有足够的内存，那么系统就会进行分页或交换，以完成当前活动事务。

当系统执行分页时，会将当前没有使用的信息从内存转移到硬盘上，这样就可以为当前需要内存的程序分配内存。如果频繁发生分页，系统性能就会严重降低，从而导致很多程序的执行时间变长。

当系统执行交换时，会将某些进程所分配的不活跃内存页(根据 LRU 算法)从内存转移到硬盘上，这样另一个活动进程就可以得到所需要的内存。交换基于系统循环时间。如果交换太过频繁，系统甚至会出现当机。

接下来让我们通过实际生产中的案例，看一下内存分配不当会带来的问题。

5.3.1 诊断案例一-SGA 与 Swap

案例描述:用户报告，服务器启动一段时间以后，无法建立数据库连接。重新启动几分钟以后，再次无法连接。

操作系统：SunOS 5.8，系统无法正常使用.

1. 登陆数据库，检查系统进程

登陆系统，检查系统进程，发现后台进程正常，有一定量的用户连接：

```
wapplatform:/>su - oracle
Sun Microsystems Inc. SunOS 5.8 Generic Patch October 2001
You have new mail.
/export/home1/oracle/admin>ps -ef|grep ora
oracle 25269 25258 0 13:58:36 pts/3 0:00 grep ora
oracle 25267 1 1 13:58:34 ? 0:00 oracleHSWAPDB (LOCAL=NO)
oracle 25193 1 0 13:57:03 ? 0:01 oracleHSWAPDB (LOCAL=NO)
oracle 25209 1 0 13:57:09 ? 0:00 oracleHSWAPDB (LOCAL=NO)
oracle 25244 1 1 13:58:23 ? 0:00 oracleHSWAPDB (LOCAL=NO)
oracle 25218 1 0 13:57:23 ? 0:00 oracleHSWAPDB (LOCAL=NO)
.................
oracle 25149 1 0 13:56:41 ? 0:01 ora_lgwr_HSWAPDB
oracle 25153 1 0 13:56:42 ? 0:01 ora_smon_HSWAPDB
oracle 25155 1 0 13:56:42 ? 0:00 ora_reco_HSWAPDB
oracle 25151 1 0 13:56:41 ? 0:00 ora_ckpt_HSWAPDB
oracle 25145 1 0 13:56:41 ? 0:00 ora_dbw0_HSWAPDB
oracle 25143 1 0 13:56:41 ? 0:00 ora_pmon_HSWAPDB
.......
```

2. 检查警报日志文件

发现如下大量提示信息：

```
Tue Mar 23 13:40:45 2004
skgpspawn failed:category = 27142, depinfo = 12, op = fork, loc = skgpspawn3
skgpspawn failed:category = 27142, depinfo = 12, op = fork, loc = skgpspawn3
skgpspawn failed:category = 27142, depinfo = 12, op = fork, loc = skgpspawn3
skgpspawn failed:category = 27142, depinfo = 12, op = fork, loc = skgpspawn3
skgpspawn failed:category = 27142, depinfo = 12, op = fork, loc = skgpspawn3
skgpspawn failed:category = 27142, depinfo = 12, op = fork, loc = skgpspawn3
skgpspawn failed:category = 27142, depinfo = 11, op = fork, loc = skgpspawn5
skgpspawn failed:category = 27142, depinfo = 12, op = fork, loc = skgpspawn3
skgpspawn failed:category = 27142, depinfo = 12, op = fork, loc = skgpspawn3
Tue Mar 23 13:42:02 2004
skgpspawn failed:category = 27142, depinfo = 12, op = fork, loc = skgpspawn3
```

该提示说明系统无法 fork 新的数据库进程，数据库无法 spawn a new session:

而且这里 "skgpspawn failed:category = 27142"实际上应该是 Oracle 的错误号，我们可以通过 Oracle 的手册查询到这个错误的具体内容，在 Unix/Linux 上我们可以通过 oerr 工具获得相

关信息：

```
$ oerr ora 27142
27142, 0000, "could not create new process"
// *Cause: OS system call
// *Action: check errno and if possible increase the number of processes
```

3. 尝试连接数据库

当再次尝试连接数据库时，收到如下错误信息，无法连接数据库：

```
$ sqlplus "/ as sysdba"
SQL*Plus: Release 9.2.0.3.0 - Production on 星期二 3月 23 14:14:06 2004
Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.
ERROR:
ORA-12540: TNS: 超出内部限制
请输入用户名:
ERROR:
ORA-12540: TNS: 超出内部限制
请输入用户名:
ERROR:
ORA-12540: TNS: 超出内部限制
SP2-0157: 在 3 次尝试之后无法 CONNECT 到 ORACLE, 退出 SQL*Plus
```

内部限制超过，通常说明某些系统资源不足.

4. 检查系统日志

检查系统日志信息，发现大量失败的 su 操作，有 swap 区不足的报告：

```
wapplatform:/>dmesg
2004 年 03 月 23 日 星期二 14 时 00 分 32 秒 CST
Mar 23 13:40:56 wapplatform su: [ID 810491 auth.crit] 'su root' failed for root on /dev/pts/2
Mar 23 13:46:26 wapplatform genunix: [ID 470503 kern.warning] WARNING: Sorry, no swap space
to grow stack for pid 24888
(sqlplus)
Mar 23 13:49:18 wapplatform su: [ID 810491 auth.crit] 'su oracle' failed for root on /dev/pts/6
Mar 23 13:54:03 wapplatform genunix: [ID 470503 kern.warning] WARNING: Sorry, no swap space
to grow stack for pid 25035 (su)
Mar 23 13:54:08 wapplatform genunix: [ID 470503 kern.warning] WARNING: Sorry, no swap space
to grow stack for pid 25036 (su)
```

现在基本可以判断是交换区的问题，当然和 Oracle SGA 设置有关.

5. 检查系统内存及交换区使用

通过 TOP 工具检查系统内存及 Swap 使用情况：

```
$ top
last pid: 25456; load averages: 0.67, 0.70, 0.69            14:10:03
93 processes: 91 sleeping, 2 on cpu
CPU states: 72.7% idle, 14.9% user, 2.7% kernel, 9.7% iowait, 0.0% swap
Memory: 1024M real, 34M free, 752M swap in use, 10M swap free

PID   USERNAME THR PRI NICE SIZE RES STATE TIME CPU COMMAND
25199   oracle 1 40 0 674M 631M cpu/2 8:03 16.32% oracle
25209   oracle 1 30 0 675M 630M sleep 0:03 0.13% oracle
25159   oracle 1 48 0 674M 628M sleep 0:03 0.06% oracle
25384   oracle 1 58 0 2632K 1736K cpu/0 0:01 0.05% top
25145   oracle 143 58 0 682M 630M sleep 0:01 0.03% oracle
```

发现物理内存仅为 1G，free 部分为 34M，交换区使用了 752M，仅 Swap Free 部分仅余 10M 。由此我们知道系统内存严重不足，Swap 区不足。

6. 检查数据库的 SGA 设置

发现 SGA 设置为: 622299344 bytes，接近 600M，这个 SGA 设置过高：

```
/export/home1/oracle>sqlplus "/ as sysdba"
SQL*Plus: Release 9.2.0.3.0 - Production on 星期二 3月 23 14:02:30 2004
Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.
连接到:
Oracle9i Enterprise Edition Release 9.2.0.3.0 - 64bit Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.3.0 - Production
SQL> show sga
Total System Global Area 622299344 bytes
Fixed Size               731344 bytes
Variable Size          268435456 bytes
Database Buffers       352321536 bytes
Redo Buffers             811008 bytes
```

对于 RAM 小于 1G 的系统，Dedicated 模式下,通常我们建议 Oracle 的 SGA 一般不应超过 1/2 物理内存，SGA 之外我们还应考虑到 PGA 及操作系统的内存分配。

7. 调整内容

第一步调整，减小 SGA，为系统保留足够的内存。

第二步调整，为系统增加 swap 区。

```
wapplatform:/var/swap>cd /export/home1
wapplatform:/export/home1>mkdir swap
wapplatform:/export/home1>cd swap
```

```
wapplatform:/export/home1/swap>mkfile -v 1g swapfile1
swapfile1 1073741824 bytes
wapplatform:/export/home1/swap>swap -a /export/home1/swap/swapfile1
wapplatform:/export/home1/swap>swap -s
总数：分配了 623160k 字节 + 保留 162704k = 已使用 785864k，1010936k 可用
```

至此系统恢复正常，问题解决

7. **问题总结**:

Oracle 数据库问题的解决从来就离不开操作系统，很多时候我们必须通过操作系统一级的手段来诊断并解决问题。

关于操作系统，一般 Swap 区的推荐值为 2*RAM。如果物理内存（RAM）很大，不一定非要把 Swap 设置为 2xSwap，通常可以设置 Swap = Ram 或者小于物理内存（如内存超过 32G 则完全可以设置 Swap 为 16G）。如果物理内存（RAM）过小，在系统繁忙期间，产生大量交换无法换到磁盘，就会出现问题，如本案例就是这样。

另外，如果系统物理内存较小，通常设置 SGA < 1/2 Ram，要考虑为 Server process 的 PGA 消耗及 OS 保留足够的内存空间.

5.3.2 诊断案例二-SGA 设置过高导致的系统故障

案例描述:这是一个大型生产系统，问题出现时系统累计大量用户进程，用户请求得不到及时响应，新的进程不断尝试建立连接，连接数很快被用完。

最后系统处于挂起状态，无法进行服务响应。

数据库版本:9.2.0.3

操作系统:Solaris8

1. **登陆数据库，检查警告日志文件**

接到问题报告之后，马上登陆数据库服务器，检查 alert 文件。

日志中记录如下错误信息，说明系统异步 IO 出现问题:

```
WARNING: aiowait timed out 2 times
Tue Aug 26 15:33:32 2003
WARNING: aiowait timed out 2 times
Tue Aug 26 15:33:34 2003
WARNING: aiowait timed out 2 times
Tue Aug 26 15:33:36 2003
WARNING: aiowait timed out 2 times
Tue Aug 26 15:33:38 2003
WARNING: aiowait timed out 2 times
```

后来在其他平台上也发现类似的错误提示：

```
Tue Nov 11 14:08:24 2003
WARNING: aiowait timed out 1 times
Tue Nov 11 14:18:24 2003
WARNING: aiowait timed out 2 times
Tue Nov 11 14:28:24 2003
WARNING: aiowait timed out 3 times
Tue Nov 11 14:38:24 2003
WARNING: aiowait timed out 4 times
Tue Nov 11 14:38:24 2003
WARNING: aiowait timed out 5 times
Tue Nov 11 14:48:24 2003
WARNING: aiowait timed out 5 times
Tue Nov 11 14:58:24 2003
WARNING: aiowait timed out 6 times
Tue Nov 11 15:08:24 2003
WARNING: aiowait timed out 7 times
Tue Nov 11 15:08:24 2003
WARNING: aiowait timed out 7 times
```

注意到每次 WARNING 的间隔时间为 10 分钟。

> 注意：由于警报日志文件（alert_<sid>.log）中会记录数据库出现故障时的错误信息等，所以我们处理数据库问题时，通常应该首先检查该文件，看是否可以从中发现问题线索。

我们知道在 SUN 的某些版本上异步 IO 存在问题，而异步 IO 缺省是打开的

```
SQL> show parameter disk_a
NAME          TYPE               VALUE
------------------------------------ ----------- ------------------------------
disk_asynch_io    Boolean             TRUE
```

针对此问题，我们暂时停用了数据库的异步 IO 写入。

2. 检查共享内存设置

alert 文件中还记录了以下警告信息:

```
  Tue Aug 26 21:37:40 2003
  WARNING: EINVAL creating segment of size 0x0000000190400000
  fix shm parameters in /etc/system or equivalent
```

该信息说明系统内核参数设置不合理或者和 SGA 不匹配，检查 system 配置文件：

```
$ cat /etc/system
.......................
set shmsys:shminfo_shmmax=4096000000
```

```
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmni=200
set shmsys:shminfo_shmseg=200
set semsys:seminfo_semmap=1024
set semsys:seminfo_semmni=2048
set semsys:seminfo_semmns=2048
set semsys:seminfo_semmnu=2048
set semsys:seminfo_semume=200
set semsys:seminfo_semmsl=2048
```

发现最大共享内存段设置为 4G

3. 检查 SGA 设置

察看内核参数之后，我们检查 SGA 设置：

```
SQL> show sga
Total System Global Area   6695660272 bytes
Fixed Size                   740080 bytes
Variable Size             2399141888 bytes
Database Buffers          4294967296 bytes
Redo Buffers                 811008 bytes
```

发现 SGA 设置接近 7G（超过了 4G，Oracle 将分配多个共享内存段），这也就是步骤 2 中警告提示出现的原因。

4. 交换区问题

我们用 top 工具检查系统运行状况

```
# /usr/local/bin/top

last pid: 16899; load averages: 0.82, 0.81, 0.83 21:49:05
1230 processes:1228 sleeping, 1 running, 1 on cpu
CPU states: 50.1% idle, 7.4% user, 8.6% kernel, 33.9% iowait, 0.0% swap
Memory: 8192M real, 118M free, 12G swap in use, 11G swap free

PID USERNAME THR PRI NICE SIZE RES STATE TIME CPU COMMAND
15751 oracle 11 44 0 6456M 6408M sleep 0:02 0.49% oracle
15725 oracle 11 58 0 6458M 6410M sleep 0:02 0.46% oracle
251 root 12 48 0 7096K 1944K sleep 126:00 0.45% picld
16540 oracle 11 58 0 6458M 6411M sleep 0:01 0.45% oracle
16766 root 1 43 0 3744K 2248K cpu/1 0:01 0.41% top
16408 oracle 11 58 0 6457M 6410M sleep 0:01 0.34% oracle
15989 oracle 11 58 0 6458M 6409M sleep 0:01 0.34% oracle
```

```
15919 oracle 11 58 0 6457M 6409M sleep 0:02 0.30% oracle
16404 oracle 11 58 0 6457M 6409M sleep 0:00 0.28% oracle
16327 oracle 11 55 0 6457M 6410M sleep 0:00 0.27% oracle
14870 oracle 11 58 0 6457M 6412M sleep 0:05 0.24% oracle
16851 oracle 11 35 0 6457M 6411M sleep 0:00 0.22% oracle
16467 oracle 11 58 0 6457M 6409M sleep 0:00 0.21% oracle
16163 oracle 11 58 0 6457M 6408M sleep 0:03 0.21% oracle
15159 oracle 11 58 0 6457M 6408M sleep 0:05 0.21% oracle
```

发现在 Top 输出中，使用了 12G 的 Swap，而物理内存几乎耗尽：

```
Memory: 8192M real, 118M free, 12G swap in use, 11G swap free
```

至此我们可以初步作出以下判断:

由于 SGA 设置过大(将近 7G)导致运行时产生大量交换，大量 SWAP 交换进而引发磁盘 I/O 问题。这也就应该是我们第一步看到异步 I/O 错误的原因：

```
WARNING: aiowait timed out 1 times
```

大量交换导致数据库性能急剧下降，进而导致用户请求得不到快速响应，堵塞、累积，直至数据库失去响应。

5. 解决方案

此问题主要是由于 SGA 设置不当引起，我们马上缩小了 SGA 设置:

```
SQL> show sga
Total System Global Area 3591870848 bytes
Fixed Size                  735616 bytes
Variable Size           1442840576 bytes
Database Buffers        2147483648 bytes
Redo Buffers                811008 bytes
```

此时，数据库减少了交换,达到了稳定运行,用户请求可以得到快速响应。至此问题解决完成。

6. 系统调整后状态

调整后系统运行状况如下 TOP 输出所示，内存使用已经降低到合理范围，系统得以稳定运行：

```
$ top

last pid: 12745;  load averages:  0.46,  0.79,  0.65          22:22:49
228 processes: 227 sleeping, 1 on cpu
CPU states: 92.3% idle,  5.0% user,  1.6% kernel,  1.1% iowait,  0.0% swap
Memory: 8192M real,  3817M free, 4015M swap in use, 15G swap free

PID USERNAME THR PRI NICE  SIZE   RES STATE    TIME    CPU COMMAND
```

```
 12610 oracle     1  51    0 3511M   22M sleep    0:04  1.96% oracle
 12595 oracle     1  48    0 3511M   22M sleep    0:03  0.92% oracle
 12630 oracle     1  38    0 3511M   21M sleep    0:01  0.84% oracle
 12614 oracle     1  46    0 3511M   22M sleep    0:01  0.64% oracle
 12620 oracle     1  58    0 3511M   22M sleep    0:01  0.53% oracle
 12709 oracle     1  48    0 3511M   21M sleep    0:00  0.45% oracle
   265 root      11  38    0 7032K 1920K sleep    3:16  0.42% picld
 12729 oracle     1   0    0 3511M   20M sleep    0:00  0.26% oracle
 12741 oracle     1  58    0 2768K 1760K cpu/3    0:00  0.19% top
 12745 oracle     1  44    0 3506M   16M sleep    0:00  0.17% oracle
 12711 oracle     1  48    0 3506M   16M sleep    0:00  0.11% oracle
 12738 oracle     1  43    0 3506M   16M sleep    0:00  0.06% oracle
  7606 oracle     1  45    0   17M 6928K sleep    0:07  0.05% tnslsnr
 12721 oracle     1  34    0 3506M   16M sleep    0:00  0.05% oracle
 12723 oracle     1  53    0 3506M   16M sleep    0:00  0.05% oracle
```

7. 一点总结

这个案例和前面我提到的另外一个极其相似，同样都是 SGA 设置不当引起的数据库问题。

这些问题本身并不复杂，应该在数据库规划和建设阶段就避免掉。良好的规划和设计是系统稳定运行的基础。如果在生产环境中遇到这类问题，更重要的是快速判断，准确定位，及时解决问题，减少故障对于业务系统的影响。

所以，在这些案例处理的过程中，最重要的其实只是一个思路和想法。

8. 后续研究

在故障处理之后，进一步研究发现，这一问题在 Oracle9.2.0.3 的 Solaris 平台上广泛存在，Oracle 为此记录了 Bug（Bug No：2086687）并做出了改进。

在 Oracle9.2.0.3 版本中，缺省的，Oracle 在异步 I/O 出现问题时，会连续 WARNING 100 次，每次间隔 10 分钟，也就是在 1000 分钟之后会给出 ORA-27083 错误：

```
[oracle@jumper oracle]$ oerr ORA 27083
27083, 00000, "skgfrliopo: waiting for async I/Os failed"
// *Cause:  The aio_waitn() library call returned an error.
// *Action: Check errno.
```

这一设置让很多用户不满，因为连续 100 次的 I/O 超时可能已经给系统带来的严重的影响，所以在 Oracle9.2.0.4 版本中，Oracle 引入了一个新的隐含参数用以控制在报告 ORA-27083 错误前 WARNING 的次数，这个参数是：_aiowait_timeouts，该参数的缺省值为 100：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
```

```
Oracle9i Enterprise Edition Release 9.2.0.4.0 - Production
SQL> @GetParDescrb.sql
Enter value for par: aiowait
old   6:    AND x.ksppinm LIKE '%&par%'
new   6:    AND x.ksppinm LIKE '%aiowait%'
NAME               VALUE       DESCRIB
---------------- ------------ ------------------------------------------------------------
_aiowait_timeouts       100    Number of aiowait timeouts before error is reported
```

到这里，这个问题可以告一段落了。

5.3.3 诊断案例三-如何诊断和解决 CPU 高度消耗(100%)问题

很多时候服务器都可能会经历 CPU 消耗 100%的性能问题，排除系统的异常，这类问题通常都是因为系统中存在性能低下甚至存在错误的 SQL 语句, 消耗了大量的 CPU 所致。本节将通过一个案例就如何捕获这样的 SQL 给出一个通用的方法.

问题描述：系统 CPU 高度消耗,系统运行缓慢

操作系统：Sun Solaris8

数据库版本：Oracle9.2.0.3

1. 通过 Top 命令查看

通过 Top 工具可以观察到，在进程列表里，存在两个高 CPU 耗用的 Oracle 进程，分别消耗了 47.77%和 40.98%的 CPU 资源：

```
$ top
load averages:  1.61,  1.28,  1.25                    HSWAPJSDB             10:50:44
172 processes: 160 sleeping, 1 running, 3 zombie, 6 stopped, 2 on cpu
CPU states:  0.2% idle, 98.5% user,  1.3% kernel,  0.0% iowait,  0.0% swap
Memory: 4.0G real, 1.4G free, 1.9G swap in use, 8.9G swap free
   PID USERNAME THR PR NCE  SIZE   RES STATE   TIME FLTS    CPU COMMAND
 20521 oracle     1 40   0  1.8G  1.7G run     6:37    0 47.77% oracle
 20845 oracle     1 40   0  1.8G  1.7G cpu02   0:41    0 40.98% oracle
 20847 oracle     1 58   0  1.8G  1.7G sleep   0:00    0  0.84% oracle
 20780 oracle     1 48   0  1.8G  1.7G sleep   0:02    0  0.83% oracle
 15828 oracle     1 58   0  1.8G  1.7G sleep   0:58    0  0.53% oracle
 20493 oracle     1 58   0  1.8G  1.7G sleep   0:03    0  0.29% oracle
 20887 oracle     1 48   0  1.8G  1.7G sleep   0:00    0  0.13% oracle
 20851 oracle     1 58   0  1.8G  1.7G sleep   0:00    0  0.10% oracle
 20483 oracle     1 48   0  1.8G  1.7G sleep   0:00    0  0.09% oracle
```

2. 找到存在问题的进程信息

通过 PID，使用操作系统工具可以找到这两个有问题的进程。确认这是两个远程连接的用户进程：

```
$ ps -ef|grep 20521
  oracle 20909 20875  0 10:50:53 pts/10   0:00 grep 20521
  oracle 20521     1 47 10:43:59 ?        6:45 oraclejshs (LOCAL=NO)
$ ps -ef|grep 20845
  oracle 20845     1 44 10:50:00 ?        0:55 oraclejshs (LOCAL=NO)
  oracle 20918 20875  0 10:50:59 pts/10   0:00 grep 20845
```

3. 捕获存在问题的 SQL 语句

通过如下 getsql.sql 脚本，我们可以获取相关 SQL 语句：

```
SELECT   /*+ ORDERED */ sql_text FROM v$sqltext a
   WHERE (a.hash_value, a.address) IN (
            SELECT DECODE (sql_hash_value,0, prev_hash_value,sql_hash_value),
                   DECODE (sql_hash_value, 0, prev_sql_addr, sql_address)
              FROM v$session b
             WHERE b.paddr = (SELECT addr FROM v$process c WHERE c.spid = '&pid'))
ORDER BY piece ASC;
```

注意这里我们涉及了 3 个视图,并应用其关联进行数据获取。

首先需要输入一个 pid,这个 pid 即 process id,也就是在 Top 或 ps 中我们看到的 PID。

通过 pid 和 v$process.spid 相关联我们可以获得 Process 的相关信息，进而通过 v$process.addr 和 v$session.paddr 相关联,我们就可以获得和 session 相关的所有信息。

再结合 v$sqltext,我们即可获得当前 session 正在执行的 SQL 语句。通过 v$process 视图，我们得以把操作系统和数据库关联了起来。

4. 连接数据库,找到问题 sql 及进程

通过 Top 中我们观察到的 PID,进而应用我的 getsql 脚本,我们得到以下结果输出.

```
SQL> @getsql
Enter value for spid: 20521
old  10: where c.spid = '&pid'
new  10: where c.spid = '20521'
SQL_TEXT
------------------------------------------------------------------
select * from (select VC2URL,VC2PVDID,VC2MOBILE,VC2ENCRYPTFLAG,S
ERVICEID,VC2SUB_TYPE,CISORDER,NUMGUID,VC2KEY1, VC2NEEDDISORDER,V
C2PACKFLAG,datopertime from hsv_2cpsync where datopertime<=sysda
te and numguid>70000000000308 order by NUMGUid) where rownum<=20
```

那么这段代码就是当前正在疯狂消耗 CPU 的罪魁祸首.

接下来需要进行的工作就是找出这段代码的问题，看是否可以通过优化提高其效率，减少资源消耗.

5. 进一步的跟踪

如果需要进一步的跟踪详细信息，可以通过 dbms_system 包来进行：

```
SQL> @getsid
Enter value for spid: 20521
old 3: select addr from v$process where spid = &spid)
new 3: select addr from v$process where spid = 20521)

SID SERIAL# USERNAME MACHINE
------------------------------------------------------------
45 38991 HSUSER_V51 hswapjsptl1.hurray.com.cn
SQL> exec dbms_system.set_sql_trace_in_session(45,38991,true);
PL/SQL procedure successfully completed.
```

关于 dbms_system 包的使用，在后面的章节将会有详细的介绍。

6.一点说明

很多时候，高 CPU 消耗都是由于问题 SQL 导致的,所以找到这些 SQL 通常也就找到了问题所在，通过优化调整通常就可以解决问题。

但是有时候你可能会发现，这些最消耗 CPU 的进程是后台进程，这一般是由于异常、BUG 或者恢复后的异常导致的，需要具体问题具体分析了.

参考文献：

Oracle9i Database Administrator's Guide Release 2 (9.2)	Part Number A96521-01
ADVANCED MANAGEMENT OF WORKING AREAS IN ORACLE 9I/10G	By Joze Senegacnik
SQL Memory Management in Oracle9i	Benoit Dageville,Mohamed Zait
Oracle8i Internal Services for Waits, Latches, Locks and Memory	Steve Adams