



Oracle Tuning (Oracle 性能调整)的一些总结频道提供数据库,企业信息化,开发技术,oracle,db2,sybase,Microsoft,sap,erp,scm,用友,金蝶,java,ibm信息服务。

网站首页 | 编程语言 | 站长之家 | 图形图像 | 网页制作 | 操作系统 | 软件教学 | 网络安全

天天网络技术资讯 通告：☆ 免责声明：天天网络技术资讯 所有内容均来自互联网，如有侵犯版权，请速联系站长将会以最快的速度将其删除！ 投递文章 投稿指南 ☆

您的位置： 首页>编程语言>数据库教程>Oracle教程>Oracle Tuning (Oracle 性能调整)的一些总结

搜索标题



Oracle Tuning (Oracle 性能调整)的一些总结

2008-07-30 来源:ItYin.Com 作者:ItYin.Com 【大 中 小】

关于Oracle的性能调整，一般包括两个方面，一是指Oracle数据库本身的调整，比如SGA、PGA的优化设置，二是连接Oracle的应用程序以及SQL语句的优化。做好这两个方面的优化，就可以使一套完整的Oracle应用系统处于良好的运行状态。

本文主要是把一些Oracle Tuning的文章作了一个简单的总结，力求以实际可操作为目的，配合讲解部分理论知识，使大部分具有一般Oracle知识的使用者能够对Oracle Tuning有所了解，并且能够根据实际情况对某些参数进行调整。关于更加具体的知识，请参见本文结束部分所提及的推荐书籍，同时由于该话题内容太多且复杂，本文必定有失之偏颇甚至错误的地方，请不吝赐教，并共同进步。

1. SGA的设置

在Oracle Tuning中，对SGA的设置是要害。SGA，是指Shared Global Area，或者是 System Global Area，称为共享全局区或者系统全局区，结构如下图所示。

对于SGA区域内的内存来说，是共享的、全局的，在UNIX上，必须为oracle 设置共享内存段（可以是一个或者多个），因为oracle 在UNIX上是多进程；而在WINDOWS上oracle是单进程（多个线程），所以不用设置共享内存段。

1.1 SGA的各个组成部分

下面用 sqlplus 查询举例看一下 SGA 各个组成部分的情况：

```
SQL#gt; select * from v$sga;
NAME VALUE
-----
Fixed Size 104936
Variable Size 823164928
Database Buffers 1073741824
Redo Buffers 172032 或者
SQL#gt; show sga
Total System Global Area 1897183720 bytes
Fixed Size 104936 bytes
Variable Size 823164928 bytes
Database Buffers 1073741824 bytes
Redo Buffers 172032 bytes
```

Fixed Size

oracle 的不同平台和不同版本下可能不一样，但对于确定环境是一个固定的值，里面存储了SGA各部分组件的信息，可以看作引导建立SGA的区域。

Variable Size

包含了shared_pool_size、Java_pool_size、large_pool_size 等内存设置

Database Buffers

指数据缓冲区，在8i 中包

含db_block_buffer*db_block_size、buffer_pool_keep、buffer_pool_recycle 三部分内存。在9i 中包含db_cache_size、db_keep_cache_size、db_recycle_cache_size、 db_nk_cache_size。

Redo Buffers

推荐文章

- 经典的问题与解答(2)
- 搞定Sendmail邮件权限控制
- vb 调用 Oracle 函数返回数据集的
- 在UNIX下让ORACLE定时执行*.sql文
- Oracle安装（linux）总结一下
- [转载]关于SGA设置的一点总结
- oracle9i日常操作总结
- Common Performance Tuning Issues
- Oracle8 优化技术:输入/输出
- Oracle数据库中LONG类型字段的存取
- Linux 中 x86 的内联汇编
- sql学习

热门文章

- Oracle 数据库唯一约束中的NULL的
- Linux As3 U8环境下Oracle 9i 9208
- Oracle数据库PL/SQL过程调试的输出
- UNIX系统环境下设置自动开关数据库
- Oracle Developer 2000中的一些实
- 详细讲解Oracle SQL*Loader的使用
- 经验总结：一次Oracle数据库冷备份
- 全面解析Oracle developer的异常处
- Oracle listener静态注册和动态注
- 10g即时客户端在不同系统环境下的
- Redhat Linux AS4 安装Oracle 10g(
- 在64位Linux环境下安装Oracle数据
- 揭开Oracle 10G手工创建数据库的神
- Oracle 9i中OCCL在VC6下不能DEBUG
- 讲解往表中顺序插入N条记录的简易
- 如何使用exp以传输表空间的方式将
- 手动制作Oracle9i/10g客户端的实用
- 个人经验总结：Oracle数据库SCN号
- 一些基础的Oracle DBA笔试题和面试
- Oracle全局数据库名、环境变量和si

指日志缓冲区，log_buffer。在这里要额外说明一点的是，对于v\$parameter、v\$sghastat、v\$sga查询值可能不一样。v\$ parameter 里面的值，是指用户在初始化参数文件里面设置的值，v\$sghastat是oracle 实际分配的日志缓冲区大小（因为缓冲区的分配值实际上是离散的，也不是以block 为最小单位进行分配的），v\$sga 里面查询的值，是在oracle 分配了日志缓冲区后，为了保护日志缓冲区，设置了一些保护页，通常我们会发现保护页大小是8k(不同环境可能不一样)。参考如下内容

```
SQL#gt; select substr(name,1,10) name,substr(value,1,10) value
2 from v$parameter where name = 'log_buffer';
NAME VALUE
-----
log_buffer 163840 SQL#gt; select * from v$sghastat where pool is null; POOL NAME BYTES
-----
fixed_sga 104936
db_block_buffers 1073741824
log_buffer 163840 SQL#gt; select * from v$sga; NAME VALUE
-----
Fixed Size 104936
Variable Size 823164928
Database Buffers 1073741824
Redo Buffers 172032 172032 – 163840 = 8192 （以上试验数据是在 HP B.11.11 + Oracle
```

8.1.7.4 环境下得到的)

1.2 SGA的大小设置

在对SGA的结构进行简单分析以后，下面是关于如何根据系统的情况正确设置SGA大小的问题。SGA是一块内存区域，占用的是系统物理内存，因此对于一个Oracle应用系统来说，SGA决不是越大越好，这就需要寻找一个系统优化的平衡点。

1.2.1 设置参数前的预备

在设置SGA的内存参数之前，我们首先要问自己几个问题

- 一：物理内存多大
- 二：操作系统估计需要使用多少内存
- 三：数据库是使用文件系统还是裸设备
- 四：有多少并发连接
- 五：应用是OLTP 类型还是OLAP 类型 根据这几个问题的答案，我们可以粗略地为系统估计一下内存设置。那我们现在来逐问题地讨论，首先物理内存多大是最轻易回答的一个问题，然后操作系统估计使用多少内存呢？从经验上看，不会太多，通常应该在200M 以内（不包含大量进程PCB）。

接下来我们要探讨一个重要的问题，那就是关于文件系统和裸设备的问题，这往往轻易被我们所忽略。操作系统对于文件系统，使用了大量的buffer 来缓存操作系统块。这样当数据库获取数据块的时候，虽然SGA 中没有命中，但却实际上可能是从操作系统的文件缓存中获取的。而假如数据库和操作系统支持异步IO，则实际上当数据库写进程DBWR写磁盘时，操作系统在文件缓存中标记该块为延迟写，等到真正地写入磁盘之后，操作系统才通知DBWR写磁盘完成。对于这部分文件缓存，所需要的内存可能比较大，作为保守的估计，我们应该考虑在 0.2——0.3 倍内存大小。但是假如我们使用的是裸设备，则不考虑这部分缓存的问题。这样的情况下SGA就有调大的机会。

关于数据库有多少并发连接，这实际上关系到PGA 的大小（MTS 下还有large_pool_size）。事实上这个问题应该说还跟OLTP 类型或者OLAP 类型相关。对于OLTP类型oracle 倾向于可使用MTS,对于OLAP 类型使用独立模式，同时OLAP 还可能涉及到大量的排序操作的查询，这些都影响到我们内存的使用。那么所有的问题综合起来，实际上主要反映在UGA的大小上。UGA主要包含以下部分内存设置

SQL#gt; show parameters 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

SQL#gt; 在这部分内存中我们最关注的通常是sort_area_size，这是当查询需要排序的时候，数据库会话将使用这部分内存进行排序，当内存大小不足的时候，使用临时表空间进行磁盘排序。由于磁盘排序效率和内存排序效率相差好几个数量级，所以这个参数的设置很重要。

当出现大量排序时的磁盘I/O操作时，可以考虑增加sort_area_size的

值。sort_area_size是Oracle用于一次排序所需的最大内存数，在排序结束但是结果列返回之前，Oracle会释放sort_area_size大小的内存，但是会保留 sort_area_retained_size大小的内存，知道最后一行结果列返回以后，才释放所有的内存。

会导致排序的语句有 SELECT DISTINCT，MINUS，INTERSECT，UNION 和 min()、max()、count() 操作；而不会导致排序的语句有 UPDATE，带BETWEEN子句的SELECT 等等。

这四个参数都是针对会话进行设置的，是单个会话使用的内存的大小，而不是整个数据库使用的。偶然会看见有人误解了这个参数以为是整个数据库使用的大小，这是极其严重的错误。假如设置了MTS，则UGA被分配在large_pool_size，也就是说放在了共享内存里面，不同进程（线程）之间可以共享这部分内存。在这个基础上，我们假设数据库存在并发执行server process 为100 个，根据上面我们4 个参数在oracle8.1.7 下的默认值，我们来计算独立模式下PGA 的大致大小。由于会话并不会经常使用create_bitmap_area_size、bitmap_merge_area_size，所以我们通常不对四个参数求和。在考虑到除这四个参数外会话所保存的变量、堆栈等信息，我们估计为 2M，则200 个进程最大可能使用200M 的PGA。 1.2.2 一个经验公式

现在，根据上面这些假定，我们来看SGA 实际能达到多少内存。在1G 的内存的服务器上，我们能分配给SGA 的内存大约为400—500M。若是2G 的内存，大约可以分到1G的内存给SGA，8G 的内存可以分到5G的内存给SGA。当然我们这里是以默认的排序部分内存sort_area_size=64k进行衡量的，假如我们需要调大该参数和 hash_area_size等参数，然后我们应该根据并发的进程的数量，来衡量考虑这个问题。事实上，通常我们更习惯通过直观的公式化来表达这样的问题：

OS 使用内存+SGA+并发执行进程数*(sort_area_size+hash_ara_size+2M) #lt; 0.7*总内存 (公式是死的，系统是活的，实际应用的调整不必框公式，这不过是一个参考建议) 在我们的实际应用中，假如采用的是裸设备，我们可适当的增大SGA(假如需要的话)。由于目前几乎所有的操作系统都使用虚拟缓存，所以实际上假如就算SGA 设置的比较大也不会导致错误，而是可能出现频繁的内存页的换入与换出(page in/out)。在操作系统一级假如观察到这个现象，那么我们就需要调整内存的设置。

1.2.3 各个参数的设置

那么SGA中的各个参数具体应该按照什么样的原则来设置呢，下面进行讨论：

log_buffer

对于日志缓冲区的大小设置，通常我觉得没有过多的建议，因为参考LGWR写的触发条件之后，我们会发现通常超过3M意义不是很大。作为一个正式系统，可能考虑先设置这部分为log_buffer=1—3M 大小，然后针对具体情况再调整。

large_pool_size

对于大缓冲池的设置，假如不使用MTS，建议在20—30M 足够了。这部分主要用来保存并行查询时候的一些信息，还有就是RMAN 在备份的时候可能会使用到。假如设置了MTS，则由于UGA部分要移入这里，则需要具体根据session最大数量和 sort_ares_size 等相关会话内存参数的设置来综合考虑这部分大小的设置，一般可以考虑为 session * (sort_area_size + 2M)。这里要提醒一点，不是必须使用MTS，我们都不主张使用MTS，尤其同时在线用户数小于500的情况下。。

java_pool_size

假如数据库没有使用JAVA，我们通常认为保留10—20M 大小足够了。事实上可以更少，甚至最少

只需要32k，但具体跟安装数据库的时候的组件相关(比如http server)。

shared_pool_size

这是迄今为止最具有争议的一部分内存设置。按照很多文档的描述，这部分内容应该几乎和数据缓冲区差不多大小。但实际上情况却不是这样的。首先我们要考究一个问题，那就是这部分内存的作用，它是为了缓存已经被解析过的SQL，而使其能被重用，不再解析。这样做的原因是因为，对于一个新的SQL（shared_pool 里面不存在已经解析的可用的相同的SQL），数据库将执行硬解析，这是一个很消耗资源的过程。而若已经存在，则进行的仅仅是软分析（在共享池中寻找相同 SQL），这样消耗的资源大大减少。所以我们期望能多共享一些SQL，并且假如该参数设置不够大，经常会出现ora-04031错误，表示为了解析新的 SQL，没有可用的足够大的连续空闲空间，这样自然我们期望该参数能大一些。但是该参数的增大，却也有负面的影响，因为需要维护共享的结构，内存的增大也会使得SQL 的老化的代价更高，带来大量的治理的开销，所有这些可能会导致CPU 的严重问题。 在一个充分使用绑定变量的比较大的系统中，shared_pool_size 的开销通常应该维持在300M 以内。除非系统使用了大量的存储过程、函数、包，比如oracle erp 这样的应用，可能会达到500M甚至更高。于是我们假定一个1G内存的系统，可能考虑设置该参数为100M，2G 的系统考虑设置为150M,8G 的系统可以考虑设置为200—300M。

对于一个没有充分使用或者没有使用绑定变量系统，这可能给我们带来一个严重的问题。所谓没有使用bind var 的SQL，我们称为Literal SQL。也就是比如这样的两句SQL我们认为是不同的SQL,需要进行2 次硬解析：

select * from EMP where name = 'TOM';

select * from EMP where name = 'JERRY';

假如把 'TOM' 和 'JERRY' 换做变量V，那就是使用了bind var，我们可以认为是同样的SQL 从而能很好地共享。共享SQL 本来就是shared_pool_size 这部分内存存在的本意，oracle的目的也在于此，而我们不使用bind var 就是违反了oracle 的初衷，这样将给我们的系统带来严重的问题。当然，假如通过在操作系统监控，没有发现严重的cpu问题，我们假如发现该共享池命中率不高可以适当的增加 shred_pool_size。但是通常我们不主张这部分内存超过800M（非凡情况下可以更大）。

事实上，可能的话我们甚至要想办法避免软分析，这在不同的程序语言中实现方式有差异。我们也可能通过设置session_cached_cursors 参数来获得帮助（这将增大PGA）

关于使用绑定变量的话题，在下面的应用优化中继续讨论。 Data buffer

现在我们来谈数据缓冲区，在确定了SGA 的大小并分配完了前面部分的内存后，其余的，都分配给这部分内存。通常，在答应的前提下，我们都尝试使得这部分内存更大。这部分内存的作用主要是缓存 DB BLOCK，减少甚至避免从磁盘上获取数据，在8i中通常是

由db_block_buffers*db_block_size 来决定大小的。假如我们设置了buffer_pool_keep 和buffer_pool_recycle，则应该加上后面这两部分内存的大小。 可以看出，设置SGA时基本上应该把握的原则是：

data buffer 一般可以尽可能的大

shared_pool_size 应该适度

log buffer 在 1MB 以内就可以了 假定oracle是 32 bit ,服务器RAM大于2G ，注重你的PGA的情况，,则建议

shared_pool_size + data buffer +large_pool_size + java_pool_size #lt; 1.6G 再具体化，假如512M RAM

建议 shared_pool_size = 50M, data buffer = 200M 假如1G RAM

shared_pool_size = 100M , data buffer = 500M 假如2G RAM

shared_pool_size = 150M ,data buffer = 1.2G 物理内存再大已经跟参数没有关系了 假定64 bit ORACLE 内存4G

shared_pool_size = 200M , data buffer = 2.5G 内存8G

shared_pool_size = 300M , data buffer = 5G 内存 12G

shared_pool_size = 300M-----800M , data buffer = 8G 1.3 32bit 与 64bit 对SGA的影响 为什么在上面SGA大小设置的经验规则中要分 32bit Oracle 和 64bit Oracle 呢，是因为这关系

到SGA大小的上限问题。在32bit的数据库下，通常oracle只能使用不超过1.7G的内存，即使我们拥有12G的内存，但是我们却只能使用1.7G，这是一个莫大的遗憾。假如我们安装64bit的数据库,我们就可以使用很大的内存，几乎不可能达到上限。但是64bit 的数据库必须安装在64bit 的操作系统上，可惜目前windows上只能安装32bit的数据库，我们通过下面的方式可以查看数据库是 32bit 还是 64bit：

```
SQL#> select * from v$version;
```

BANNER

```
-----  
Oracle8i Enterprise Edition Release 8.1.7.0.0 - ProdUction  
PL/SQL Release 8.1.7.0.0 - Production  
CORE 8.1.7.0.0 Production  
TNS for 32-bit Windows: Version 8.1.7.0.0 - Production  
NLSRTL Version 3.4.1.0.0 – Production 在UNIX平台下的显示有所不同，明显可以看出是 64bit  
Oracle，比如在HP-UX平台上：
```

```
SQL#> select * from v$version; BANNER
```

```
-----  
Oracle8i Enterprise Edition Release 8.1.7.4.0 - 64bit Production  
PL/SQL Release 8.1.7.4.0 - Production  
CORE 8.1.7.0.0 Production  
TNS for HPUX: Version 8.1.7.4.0 - Production  
NLSRTL Version 3.4.1.0.0 – Production 32bit的oracle无论跑在32bit或者64bit的平台都有SGA的限制的，而对于32bit的平台只能跑32bit的oracle，但是在特定的操作系统下，可能提供了一定的手段，使得我们可以使用超过1.7G 的内存，达到2G 以上甚至更多。由于我们现在一般都使用64bit Oracle，因此关于如何在32bit平台上扩展SGA大小的问题不再赘述。
```

1.4 9i中相关参数的变化

oracle的版本的更新，总是伴随着参数的变化，并且越来越趋向于使得参数的设置更简单，因为复杂的参数设置使得DBA们经常焦头烂额。关于内存这部分的变化，我们可以考察下面的参数。事实上在9i中数据库本身可以给出一组适合当前运行系统的SGA相关部分的参数调整值（参考V\$DB_CACHE_ADVICE、V\$SHARED_POOL_ADVICE），关于PGA也有相关视图V\$PGA_TARGET_ADVICE 等。Data buffer

9i 中保留了8i中的参数，如设置了新的参数，则忽略旧的参数。9i中用db_cache_size来取代db_block_buffers，用db_keep_cache_size 取代buffer_pool_keep, 用db_recycle_cache_size 取代buffer_pool_recycle；这里要注重9i 中设置的是实际的缓存大小而不再是块的数量。另外9i新增加了db_nk_cache_size，这是为了支持在同一个数据库中使用不同的块大小而设置的。对于不同的表空间，可以定义不同的数据块的大小，而缓冲区的定义则依靠该参数的支持。其中n 可以为2、4、6、8、16 等不同的值。在这里顺便提及的一个参数就是db_block_lru_latches，该参数在9i中已经成为了保留参数，不推荐手工设置。PGA

在9i 里面这部分也有了很大的变化。在独立模式下，9i已经不再主张使用原来的UGA相关的参数设置，而代之以新的参数。假如 workarea_size_policy=AUTO（缺省），则所有的会话的UGA 共用一大块内存，该内存由 pga_aggregate_target 设置。在我们根据前面介绍的方法评估了所有进程可能使用的最大PGA 内存之后，我们可以通过在初始化参数中设置这个参数，从而不再关心其他 “*_area_size” 参数。SGA_MAX_SIZE

在9i中若设置了SGA_MAX_SIZE，则在总和小于等于这个值内，可以动态的调整数据缓冲区和共享池的大小

```
SQL#> show parameters sga_max_size
```

```
NAME TYPE VALUE
```

```
-----  
sga_max_size unknown 193752940
```

```
SQL#>
```

```
SQL#> alter system set db_cache_size = 30000000;
System altered.
SQL#> alter system set shared_pool_size = 20480000;
System altered.
```

1.5 lock_sga = true 的问题

由于几乎所有的操作系统都支持虚拟内存，所以即使我们使用的内存小于物理内存，也不能避免操作系统将SGA 换到虚拟内存（SWAP）。所以我们可以尝试使得SGA 锁定在物理内存中不被换到虚拟内存中，这样减少页面的换入和换出，从而提高性能。但在这里遗憾的是，windows 是无法避免这种情况的。下面我们来参考在不同的几个系统下怎么实现lock_sga

AIX 5L（AIX 4.3.3 以上）

```
logon aix as root
cd /usr/samples/kernel
./vmtune (信息如下) v_pingshm已经是1
./vmtune -S 1
然后oracle用户修改initSID.ora 中 lock_sga = true
重新启动数据库 HP UNIX
Root身份登陆
Create the file "/etc/privgroup": vi /etc/privgroup
Add line "dba MLOCK" to file
As root, run the command "/etc/setprivgrp -f /etc/privgroup":
```

```
$/etc/setprivgrp -f /etc/privgroup
oracle用户修改initSID.ora中lock_sga=true
重新启动数据库 SOLARIS (solaris2.6以上)
8i版本以上数据库默认使用隐藏参数 use_ism = true ，自动锁定SGA于内存中,不用设置lock_sga, 假如设置 lock_sga =true 使用非 root 用户启动数据库将返回错误。WINDOWS 不能设置lock_sga=true,可以通过设置pre_page_sga=true,使得数据库启动的时候就把所有内存页装载，这样可能起到一定的作用。
```

2. 应用优化

下面我们从技术的角度入手，来探讨数据库优化方面的问题。通常作为优化Oracle系统的人，或者是DBA，其实很多时候对应用并不很了解甚至可以说是完全不了解，更不要说对应用程序代码的了解。事实上呢，一个系统运行的快或者慢相信大家都明白，第一重要的是数据库的设计，然后是应用的设计， SQL语句的编写，最后才是数据库参数的调整和硬件、网络的问题，等等。所以在我们不了解一个系统的时候来优化数据库应用不是一个轻松的轻易的事情。那么我们第一步应该怎么做呢？

通常有两类方法：
其中一个方法就是我们常用的，使用statspack来进行诊断系统的瓶颈所在。
在statspack中oracle给出了几乎涵盖oracle大部分重要内容的信息。
另外一种方式，就是trace session。假如某个session运行很慢或者某个用户的某个查询很慢，那么这个时候我们可以通过trace session的方式来诊断到底是慢在哪里，看究竟执行计划是怎样的，然后在user_dump_dest下根据该session的进程号或者线程号可以找到一个产生的trace文件。通过使用tkprof格式化文件之后我们就可以看见很多的统计信息，这里包括了执行计划、parse/fetch等步骤消耗cpu的时间。通常我们是观察query模式下的consistent gets来首先看sql是否使用了索引，然后看执行计划是不是正常，是不是有调整的余地。当然假如您没有实际做过的话，这些内容说起来很抽象。这是在不了解应用和程序下针对特定session的诊断和调整过程。

trace session的方式是一种自下而上的方法，从sql入手；而statspack是自顶向下的方法，也就是从宏观上先诊断数据库的瓶颈在哪里，然后从瓶颈入手来做调整，这个习惯上又可以称为通过等待事件（wait event）入手的方法。
2.1 使用statspack
statspack是一个性能诊断工具，首先发布于Oracle8.1.6版本，在8.1.7版本中功能得到加

强。Statspack除了查找实例中的性能问题外，还可以查找应用程序中高负荷的SQL语句，很轻易确定Oracle 数据库的瓶颈所在，并且记录数据库性能状态。

在数据库中Statspack 的脚本位于\$ORACLE_HOME/RDBMS/ADMIN 目录下，对于ORACLE8.1.6,是一组以stat 开头的文件；对于ORACLE8.1.7,是一组以sp 开头的文件。

在Statspack 发布之前，我们通常能够使用诊断数据库的工具是两个脚本UTLBSTAT.SQL 和UTLESTAT.SQL，BSTAT/ESTAT 是一个非常简单的性能诊断工具。UTLBSTAT 获得开始时很多V\$视图的快照，UTLESTAT 通过先前的快照和当前视图生成一个报表。

该报表实际上相当于statspack 中的两个采样点。

Statspack 通过连续的采样，能够给我们提供至关重要的趋势分析数据。这是一个巨大的进步。能够使用Statspack 的环境我们就尽量不要使用BSTAT/ESTAT 的方式来诊断数据库问题。 2.1.1 安装statapack

§ 步骤一：

为了能够顺利安装和运行Statspack ，首先需要设置以下两个系统参数：

1. job_queue_processes

为了能够建立自动任务，执行数据收集，该参数需要大于0。你可以在初试化参数文件中修改该参数(使该参数在重起后以然有效)。

该参数可以在系统级动态修改(重起后失效)。 SQL#gt; alter system set job_queue_processes = 6;

System altered 在Oracle9i 当中，可以指定范围，如 both,这样该修改在当前及之后保持有效(仅当你使用spfile 时，假如在9i 中仍然使用pfile，那么更改方法同8i 相同): SQL#gt; alter system set job_queue_processes = 6 scope=both;

System altered

2. timed_statistics

收集操作系统的计时信息，这些信息可被用来显示时间等统计信息、优化数据库和 SQL 语句。要防止因从操作系统请求时间而引起的开销，请将该值设置为False。

使用statspack 收集统计信息时建议将该值设置为 TRUE，否则收集的统计信息大约只能起到10%的作用，将timed_statistics 设置为True 所带来的性能影响与好处相比是微不足道的。

该参数使收集的时间信息存储在在V\$SESSTATS 和V\$SYSSTATS 等动态性能视图中。

timed_statistics 参数也可以在实例级进行更改 SQL#gt; alter system set timed_statistics = true;

System altered 假如你担心一直启用timed_statistics 对于性能的影响，你可以在使用statspack 之前在system 更改，采样过后把该参数动态修改成false。 § 步骤二：

需要单独为statspack创建一个存储数据的表空间，假如采样间隔较短，周期较长，打算长期使用，那么可能需要一个大一点的表空间，假如每个半个小时采样一次，连续采样一周，数据量是很大的。下面的例子中创建了一个500M 的测试表空间。

注重: 这里创建的表空间不能太小，假如太小的话创建对象会失败，建议至少建立100M 表空间。 SQL#gt; create tablespace perfstat

2 datafile '/oracle/oradata/oradata/res/perfstat.dbf'

3 size 500M;

Tablespace created。 § 步骤三：

在 sqlplus 中用internal 身份登陆，或者拥有SYSDBA(connect / as sysdba)权限的用户登陆。

注: 在Oracle9i 中，不存在internal 用户，可以使用sys 用户以sysdba 身份连接。

先转到\$ORACLE_HOME/RDBMS/ADMIN 目录，检查安装脚本是否存在，同时我们执行脚本也可以方便些。 \$ cd \$ORACLE_HOME/rdbms/admin

\$ ls -l sp*.sql

-rw-r--r-- 1 oracle other 1774 Feb 18 2000 spauto.sql

-rw-r--r-- 1 oracle other 62545 Jun 15 2000 spcpkg.sql

-rw-r--r-- 1 oracle other 877 Feb 18 2000 spcreate.sql

-rw-r--r-- 1 oracle other 31193 Jun 15 2000 spctab.sql

-rw-r--r-- 1 oracle other 6414 Jun 15 2000 spcusr.sql

```
-rw-r--r-- 1 oracle other 758 Jun 15 2000 spdrops.sql
-rw-r--r-- 1 oracle other 3615 Jun 15 2000 spdtabs.sql
-rw-r--r-- 1 oracle other 1274 Jun 15 2000 spdusr.sql
-rw-r--r-- 1 oracle other 6760 Jun 15 2000 sppurge.sql
-rw-r--r-- 1 oracle other 71034 Jul 12 2000 spreports.sql
-rw-r--r-- 1 oracle other 2191 Jun 15 2000 sprunc.sql
-rw-r--r-- 1 oracle other 30133 Jun 15 2000 spup816.sql
$ 接下来我们就可以开始安装Statspack 了。在Oracle8.1.6 版本中运行statscre.sql;
在Oracle8.1.7 版本中运行spcreate.sql。
这期间会提示你输入缺省表空间和临时表空间的位置,输入我们为 perfstat 用户创建的表空间和
你的临时表空间。安装脚本会自动创建perfstat 用户。 $ sqlplus SQL*Plus: Release 8.1.7.0.0 -
Production on Sat Jul 26 16:27:31 2003 (c) Copyright 2000 Oracle Corporation. All rights
reserved. Enter user-name: internal Connected to:
Oracle8i Enterprise Edition Release 8.1.7.0.0 - Production
With the Partitioning option
JServer Release 8.1.7.0.0 - Production SQL#gt;
SQL#gt; @spcreate
... Installing Required Packages Package created. Grant succeeded. View created. Package
body created. Package created. Synonym dropped. Synonym created.
..... Specify PERFSTAT user's default tablespace
Enter value for default_tablespace: perfstat
Using perfstat for the default tablespace User altered. User altered. Specify PERFSTAT user's
temporary tablespace
Enter value for temporary_tablespace: temp
Using temp for the temporary tablespace User altered. NOTE:
SPCUSR complete. Please check spcusr.lis for any errors. .... 假如安装成功, 你可以接着看到
如下的输出信息:

....
Creating Package STATSPACK... Package created. No errors.
Creating Package Body STATSPACK... Package body created. No errors. NOTE:
SPCPKG complete. Please check spcpkg.lis for any errors. 可以查看.lis 文件查看安装时的错误信
息。 § 步骤四:
假如安装过程中出现错误, 那么可以运行spdrops.sql 脚本来删除这些安装脚本建立的对象。然后
重新运行spcreate.sql来创建这些对象。 SQL#gt; @spdrops
Dropping old versions (if any) Synonym dropped. Sequence dropped. Synonym dropped. Table
dropped. Synonym dropped. View dropped.
.....
NOTE:
SPDUSR complete. Please check spdusr.lis for any errors. (以上的安装过程描述是在 HP 11.11
+ Oracle 8.1.7 平台上得到的) 2.1.2 测试statspack
运行statspack.snap 可以产生系统快照, 运行两次, 然后执行spreports.sql 就可以生成一个基于
两个时间点的报告。
假如一切正常, 说明安装成功。 SQL#gt;execute statspack.snap
PL/SQL procedure successfully completed.
SQL#gt;execute statspack.snap
PL/SQL procedure successfully completed.
SQL#gt;@spreports.sql 可是有可能你会得到以下错误: SQL#gt; exec statspack.snap;
BEGIN statspack.snap; END;
*
```


ERROR at line 1:

ORA-01401: inserted value too large for column

ORA-06512: at "PERFSTAT.STATSPACK", line 978

ORA-06512: at "PERFSTAT.STATSPACK", line 1612

ORA-06512: at "PERFSTAT.STATSPACK", line 71

ORA-06512: at line 1 这是Oracle 的一个Bug, Bug 号1940915。

该Bug 自8.1.7.3 后修正。

这个问题只会出现在多位的字符集, 需要修改spcpkg.sql 脚本。

本, \$ORACLE_HOME/rdbms/admin/spcpkg.sql, 将"substr" 修改为 "substrb", 然后重新运行该脚本。

该脚本错误部分:

```
select l_snap_id
, p_dbid
, p_instance_number
, substr(sql_text,1,31)
. . . . .
```

substr 会将多位的字符, 当作一个byte.substrb 则会当作多个byte。在收集数据时, statpack 会将 top10 的 sql 前 31 个字节 存入数据表中,若在SQL 的前31 个字有中文, 就会出现此错误。

注重: 运行 spcpkg.sql 也需要以 internal 用户登录 sqlplus

2.1.3 生成statspack报告

调用spreport.sql 可以生成分析报告:

当调用spreprot.sql 时, 系统首先会查询快照列表, 然后要求你选择生成报告的开始快照ID(begin_snap)和结束快照ID(end_snap),生成一个报告。

为了生成一个report,我们至少需要两次采样:

SQL#gt; @spreport DB Id DB Name Inst Num Instance

2749170756	RES	1	res	Completed	Snapshots	Snap	Snap
Instance	DB Name	Id	Snap	Started	Level	Comment	

res	RES	1	26	Jul	2003	16:36	5
2	26	Jul	2003	16:37	5		
3	26	Jul	2003	17:03	5		

Specify the Begin and End Snapshot Ids

~~~~~

Enter value for begin\_snap:2

Begin Snapshot Id specified: 2 Enter value for end\_snap: 3

End Snapshot Id specified: 3

Specify the Report Name

~~~~~

The default report file name is sp_2_3. To use this name, press to continue, otherwise enter an alternative.

Enter value for report_name: rep0726.txt End of Report

在运行 spreport.sql 生成 statspack 报告的过程中, 会有三个地方提示用户输入:

- 1、 开始快照ID;
- 2、 结束快照ID;
- 3、 输出报告文件的文件名, 缺省的文件名是sp__

上面输入的开始快照ID是2, 开始快照ID是3, 输出报告文件的文件名是rep0726.txt

成功运行一次 statspack.snap 就会产生一个 snapshot , 在生成 statspack 报告的时候就可以看到这个 snap id 和 snap 运行的时间。运行 statspack.snap , 就是上面所说的采样, statspack 报

告是分析两个采样点之间各种情况。

2.1.4 删除历史快照数据

前面讲过，成功运行一次 `statspack.snap` 就会产生一个 `snapshot`，这个 `snapshot` 的基本信息是存放在 `PERFSTAT.stats$snapshot` 表中的，生成 `statspack` 报告时会查询该表的数据，供用户选择预备分析的 `snapshot`。假如运行 `statspack.snap` 次数多了以后，该表的数据也会增加，历史数据会影响正常运行的效果，因此需要定时清理一下历史快照数据。

删除 `stats$snapshot` 数据表中的相应数据，其他表中的数据会相应的级连删除：`SQL#gt; select max(snap_id) from stats$snapshot;`

```
MAX(SNAP_ID)
-----
166 SQL#gt; delete from stats$snapshot where snap_id #lt; = 166;
143 rows deleted 你可以更改snap_id 的范围以保留你需要的数据。
在以上删除过程中，你可以看到所有相关的表都被锁定。
SQL#gt; select a.object_id,a.oracle_username ,b.object_name
from v$locked_object a,dba_objects b
where a.object_id = b.object_id
/
OBJECT_ID ORACLE_USERNAME OBJECT_NAME
-----
-----
156 PERFSTAT SNAP$
39700 PERFSTAT STATS$LIBRARYCACHE
39706 PERFSTAT STATS$ROLLSTAT
39712 PERFSTAT STATS$SGA
39754 PERFSTAT STATS$PARAMETER
39745 PERFSTAT STATS$SQL_STATISTICS
39739 PERFSTAT STATS$SQL_SUMMARY
39736 PERFSTAT STATS$ENQUEUEESTAT
39733 PERFSTAT STATS$WAITSTAT
39730 PERFSTAT STATS$BG_EVENT_SUMMARY
39724 PERFSTAT STATS$SYSTEM_EVENT
39718 PERFSTAT STATS$SYSSTAT
39715 PERFSTAT STATS$SGASTAT
39709 PERFSTAT STATS$ROWCACHE_SUMMARY
39703 PERFSTAT STATS$BUFFER_POOL_STATISTICS
39697 PERFSTAT STATS$LATCH_MISSES_SUMMARY
39679 PERFSTAT STATS$SNAPSHOT
39682 PERFSTAT STATS$FILESTATXS
39688 PERFSTAT STATS$LATCH
174 PERFSTAT JOB$
20 rows selected Oracle 还提供了系统脚本用于Truncate 这些统计信息表，这个脚本名字是：
sptrunc.sql (8i、9i 都相同)
该脚本主要内容如下，里面看到的就是statspack 相关的所有系统表：
truncate table STATS$FILESTATXS;
truncate table STATS$LATCH;

truncate table STATS$LATCH_CHILDREN;
truncate table STATS$LATCH_MISSES_SUMMARY;
truncate table STATS$LATCH_PARENT;
truncate table STATS$LIBRARYCACHE;
truncate table STATS$BUFFER_POOL_STATISTICS;
```

```
truncate table STATS$ROLLSTAT;  
truncate table STATS$ROWCACHE_SUMMARY;  
truncate table STATS$SGA;  
truncate table STATS$SGASTAT;  
truncate table STATS$SYSSTAT;  
truncate table STATS$SESSTAT;  
truncate table STATS$SYSTEM_EVENT;  
truncate table STATS$SESSION_EVENT;  
truncate table STATS$BG_EVENT_SUMMARY;  
truncate table STATS$WAITSTAT;  
truncate table STATS$ENQUEUESTAT;  
truncate table STATS$SQL_SUMMARY;  
truncate table STATS$SQL_STATISTICS;  
truncate table STATS$SQLTEXT;  
truncate table STATS$PARAMETER;  
delete from STATS$SNAPSHOT;  
delete from STATS$DATABASE_INSTANCE;  
commit;
```

2.1.5 一些重要脚本

1. 通过导出保存及共享数据

在诊断系统问题时，可能需要向专业人士提供原始数据，这时我们可以导出Statspack 表数据，其中我们可能用到：spueXP.par

其内容主要为：

```
file=spuexp.dmp log=spuexp.log compress=y grants=y indexes=y rows=y constraints=y  
owner=PERFSTAT consistent=y
```

我们可以导出如下：

```
exp userid=perfstat/my_perfstat_passWord parfile=spuexp.par
```

2. 删除数据
spdrop.sql 在执行时主要调用两个脚本：spdtab.sql、spdusr.sql

前者删除表及同义词等数据，后者删除用户

3. Oracle92 中新增加的脚本

1) 用于升级statspack 对象的脚本,这些脚本需要以具有SYSDBA 权限的用户运行, 升级前请先备份存在的Schema 数据:

spup90.sql: 用于升级9.0 版本的模式至9.2 版本。

spup817.sql: 假如从Statspack 8.1.7 升级,需要运行这个脚本

spup816.sql: 从Statspack 8.1.6 升级,需要运行这个脚本，然后运行spup817.sql

2) sprep.sql 用于根据给定的SQL Hash 值生成SQL 报告

2.1.6 调整statspack的收集门限

Statspack 有两种类型的收集选项： 1. 级别（level）：控制收集数据的类型

Statspack 共有三种快照级别，默认值是5

- a. level 0: 一般性能统计。包括等待事件、系统事件、系统统计、回滚段统计、行缓存、SGA、会话、锁、缓冲池统计等等。
- b. level 5: 增加SQL 语句。除了包括level0 的所有内容，还包括SQL 语句的收集，收集结果记录在stats\$sql_summary 中。
- c. level 10: 增加子锁存统计。包括level5 的所有内容。并且还会将附加的子锁存存入stats\$lathc_children 中。在使用这个级别时需要慎重，建议在Oracle support 的指导下进行。

可以通过statspack 包修改缺省的级别设置

```
SQL>execute statspack.snap(i_snap_level=>0,i_modify_parameter=>'true');
```

通过这样的设置，以后的收集级别都将是0 级。

假如你只是想本次改变收集级别，可以忽略i_modify_parameter 参数。

SQL>execute statspack.snap(i_snap_level=>10);

2. 快照门限：设置收集的数据的阈值。
快照门限只应用于stats\$sql_summary 表中获取的SQL 语句。
因为每一个快照都会收集很多数据，每一行都代表获取快照时数据库中的一个SQL 语句，所

以stats\$sql_summary 很快就会成为Statspack 中最大的表。

门限存储在stats\$statspack_parameter 表中。让我们了结一下各种门限：

- a. executions_th 这是SQL 语句执行的数量(默认值是100)
- b. disk_reads_tn 这是SQL 语句执行的磁盘读入数量（默认值是1000）
- c. parse_calls_th 这是SQL 语句执行的解析调用的数量（默认值是1000）
- d. buffer_gets_th 这是SQL 语句执行的缓冲区获取的数量（默认值是10000）

任何一个门限值超过以上参数就会产生一条记录。

通过调用statspack.modify_statspack_parameter 函数我们可以改变门限的默认值。

例如：

```
SQL#gt;execute
statspack.modify_statspack_parameter(i_buffer_gets_th=#gt;100000,i_disk_reads_th=#gt;100000)
```

2.2 对statspack报告的分析

从上面的描述可以看出，产生一个statspack报告是比较简单的，但是如何读懂statspack报告却不是那么轻易，需要对Oracle的体系架构、内存结构、等待事件以及应用系统有充分的了解，加上不断的实践，才能基本读懂statspack报告并且从报告中找到调整优化Oracle的途径。

下面接合一个实际的statspack报告，大致分析一下。 2.2.1 基本信息分析

```
DB Name DB Id Instance Inst Num Release OPS Host
-----
RES 2749170756 res 1 8.1.7.0.0 NO res Snap Id Snap Time Sessions
-----
Begin Snap: 2 26-Jul-03 16:37:08 38
End Snap: 3 26-Jul-03 17:03:23 38
Elapsed: 26.25 (mins) Statspack报告首先描述了数据库的基本情况，比如数据库名、实例名、实例个数、oracle版本号等等；然后是该报告的开始快照和结束快照的信息，包括 snap id , snap time 等等；最后是该报告经过的时间跨度，单位是分钟(mins)。 Cache Sizes
```

```
~~~~~
db_block_buffers: 61440 log_buffer: 163840
db_block_size: 8192 shared_pool_size: 52428800 然后描述了Oracle内存结构中几个重要的参数。 2.2.2 内存信息分析
```

```
Load Profile
~~~~~ Per Second Per Transaction
-----
Redo size: 4,834.87 11,116.67
Logical reads: 405.53 932.43
Block changes: 60.03 138.02

Physical reads: 138.63 318.75
Physical writes: 54.27 124.79
User calls: 62.69 144.13
Parses: 19.14 44.00
Hard parses: 2.26 5.20
Sorts: 1.83 4.20
Logons: 0.21 0.47
Executes: 21.10 48.50
Transactions: 0.43 % Blocks changed per Read: 14.80 Recursive Call %: 34.45
Rollback per transaction %: 0.00 Rows per Sort: 20.57 Redo size: 是日志的生成量，分为每秒和每事务所产生的，通常在很繁忙的系统中日志生成量可能达到上百k，甚至几百k； Logical reads: 逻辑读实际上就是logical IO=buffer gets表示的含义，我们可以这样认为，block在内存中，我们每一次读一块内存，就相当于一次逻辑读； Parses 和 Hard parses: Parse 和 hard
```


parse通常是很轻易出问题的部分，80%的系统的慢都是由于这个原因所导致的。

所谓parse分soft parse 和hard parse，soft parse是当一条sql传进来后，需要在shared pool中找是否有相同的sql，假如找到了，那就是soft parse，假如没有找着，那就开始hard parse，实际上hard parse主要是检查该sql所涉及到的所有的对象是否有效以及权限等关系，hard parse之后才根据rule/cost模式生成执行计划，再执行sql。

而hard parse的根源，基本都是由于不使用bind var所导致的，不使用bind var违反了oracle的shared pool的设计的原则，违反了这个设计用来共享的思想，这样导致shared_pool_size里面命中率下降。因此不使用bind var，将导致cpu使用率的问题，极有使得性能急剧下降。

还有就是为了维护internal structure，需要使用latch，latch是一种Oracle低级结构,用于保护内存资源，是一种内部生命周期很短的lock，大量使用latch将消耗大量的cpu资源。 Sorts: 表示排序的数量； Executes: 表示执行次数； Transactions: 表示事务数量； Rollback per transaction %: 表示数据库中事务的回退率。假如不是因为业务本身的原因，通常应该小于10%为好，回退是一个很消耗资源的操作。

Instance Efficiency Percentages (Target 100%)

~~~~~

Buffer Nowait %: 100.00 Redo NoWait %: 99.98

Buffer Hit %: 65.82 In-memory Sort %: 99.65

Library Hit %: 91.32 Soft Parse %: 88.18

Execute to Parse %: 9.28 Latch Hit %: 99.99

Parse CPU to Parse Elapsed %: 94.61 % Non-Parse CPU: 99.90 Buffer Hit %: 数据缓冲区命中率，通常应该大于90%； Library Hit %: libaray cache的命中率，通常应该大于98%； In-memory Sort %: 排序在内存的比例，假如这个比例过小，可以考虑增大sort\_area\_size，使得排序在内存中进行而不是在temp表空间中进行； Soft Parse %: 软解析的百分比，这个百分比也应该很大才好，因为我们要尽量减少hard parse。 soft parse 百分比=soft/(soft+hard)； Execute to Parse %: 这个数字也应该是越大越好，接近100%最好。有些报告中这个值是负的，看上去很稀奇。事实上这表示一个问题，sql假如被age out的话就可能出现这种情况，也就是sql老化，或执行alter system flush shared\_pool等。

Shared Pool Statistics Begin End

-----

Memory Usage %: 90.63 87.19

% SQL with executions#gt;1: 71.53 75.39

% Memory for SQL w/exec#gt;1: 59.45 65.17 % SQL with executions#gt;1: 这个表示SQL被执行次数多于一次的比率，也应该大为好，小则表示很多sql只被执行了一次，说明没有使用bind var；

2.2.3 等待事件分析

接下来，statspack报告中描述的是等待事件（Wait Events），这是Oracle中比较复杂难懂的概念。

Oracle 的等待事件是衡量Oracle 运行状况的重要依据及指标。

等待事件的概念是在Oracle7.0.1.2 中引入的，大致有100 个等待事件。在Oracle 8.0 中这个数目增加到了大约150 个，在Oracle8i 中大约有200 个事件,在Oracle9i 中大约有360 个等待事件。

主要有两种类别的等待事件，即空闲（idle）等待事件和非空闲（non-idle）等待事件。

空闲事件指Oracle 正等待某种工作,在诊断和优化数据库的时候,我们不用过多注重这部分事件。常见的空闲事件有：

- ? dispatcher timer
- ? lock element cleanup
- ? Null event
- ? parallel query dequeue wait
- ? parallel query idle wait - Slaves

? pipe get  
? PL/SQL lock timer  
? pmon timer- pmon  
? rdbms ipc message  
? slave wait  
? smon timer  
? SQL\*Net break/reset to client  
? SQL\*Net message from client  
? SQL\*Net message to client  
? SQL\*Net more data to client  
? virtual circuit status

? client message 非空闲等待事件专门针对Oracle 的活动,指数据库任务或应用运行过程中发生的等待，这些等待事件是我们在调整数据库的时候应该关注与研究的。  
一些常见的非空闲等待事件有:

? db file scattered read  
? db file sequential read  
? buffer busy waits  
? free buffer waits  
? enqueue  
? latch free  
? log file parallel write  
? log file sync 下面接合statspack中的一些等待事件进行讲述。 Top 5 Wait Events

| ~~~~~ Wait % Total          |        |           |       |      |
|-----------------------------|--------|-----------|-------|------|
| Event                       | Waits  | Time (cs) | Wt    | Time |
| -----                       |        |           |       |      |
| db file scattered read      | 26,877 | 12,850    | 52.94 |      |
| db file parallel write      | 472    | 3,674     | 15.13 |      |
| log file parallel write     | 975    | 1,560     | 6.43  |      |
| direct path write           | 1,571  | 1,543     | 6.36  |      |
| control file parallel write | 652    | 1,290     | 5.31  |      |

----- db file scattered read:DB文件分散读取。这个等待事件很常见，经常在top5中出现，这表示，一次从磁盘读数据进来的时候读了多于一个block的数据，而这些数据又被分散的放在不连续的内存块中，因为一次读进来的是多于一个block的。  
通常来说我们可以认为是全表扫描类型的读，因为根据索引读表数据的话一次只读一个block，假如这个数字过大，就表明该表找不到索引，或者只能找到有限的索引，可能是全表扫描过多，需要检查sql是否合理的利用了索引，或者是否需要建立合理的索引。  
当全表扫描被限制在内存时，它们很少会进入连续的缓冲区内，而是分散于整个缓冲存储器中。尽管在特定条件下执行全表扫描可能比索引扫描更有效，但假如出现这种等待时，最好检查一下这些全表扫描是否必要,是否可以通过建立合适的索引来减少对于大表全表扫描所产生的大规模数据读取。

对于经常使用的小表，应该尽量把他们pin 在内存中，避免不必要的老化清除及重复读取。 db file sequential read: DB文件连续读取。通常显示单个块的读取(通常指索引读取)，表示的是读进磁盘的block被放在连续的内存块中。  
事实上大部分基本代表着单个block的读入，可以说象征着 IO 或者说通过索引读入的比较多。因为一次IO若读进多个的block，放入连续的内存块的几率是很小的，分布在不同block的大量记录被读入就会碰到此事件。因为根据索引读数据的话，假设100条记录，根据索引，不算索引本身的读，而根据索引每个值去读一下表数据，理论上最多可能产生100 buffer gets，而假如是full table scan，则100条数据完全可能在一个block里面，则几乎一次就读过这个block了，就会产生

这么大的差异。

这种等待的数目很多时，可能显示表的连接顺序不佳，或者不加选择地进行索引。

对于高级事务处理（**high-transaction**）、调整良好（**welltuned**）的系统，这一数值很大是很正常的，但在某些情况下，它可能暗示着系统中存在问题。

你应当将这一等待统计量与**Statspack** 报告中的已知问题（如效率较低的**SQL**）联系起来。检查索引扫描，以保证每个扫描都是必要的，并检查多表连接的连接顺序。

**DB\_CACHE\_SIZE** 也是这些等待出现频率的决定因素。有问题的散列区域（**Hash-area**）连接应当出现在**PGA** 内存中，但它们也会消耗大量内存，从而在顺序读取时导致大量等待。它们也可能以直接路径读／写等待的形式出现。

**Free Buffer Wait:** 释放缓冲区。

这种等待表明系统正在等待内存中的缓冲，因为内存中已经没有可用的缓冲空间了。假如所有**SQL** 都得到了调优，这种等待可能表示你需要增大**DB\_BUFFER\_CACHE**。释放缓冲区等待也可能表示不加选择的**SQL** 导致数据溢出了带有索引块的缓冲存储器，没有为等待系统处理的特定语句留有缓冲区。

这种情况通常表示正在执行相当多数量的**DML**（插入／更新／删除），并且可能说明**DBWR** 写的速度不够快，缓冲存储器可能布满了相同缓冲器的多个版本，从而导致效率非常低。为了解决这个问题，可能需要考虑增加检查点、利用更多的**DBWR** 进程，或者增加物理磁盘的数量。

**Buffer Busy Wait:** 缓冲区忙。

该等待事件表示正在等待一个以**unshareable**方式使用的缓冲区，或者表示当前正在被读入**buffer cache**。也就是当进程想获取或者操作某个**block**的时候却发现被别的进程在使用而出现等待。一般来说**Buffer Busy Wait**不应大于1%。

检查缓冲等待统计部分（或**V\$WAITSTAT**），看一下等待是否位于段头。假如是，可以考虑增加自由列表（**freelist**，对于**Oracle8i DMT**）或者增加**freelist groups**。

其修改语法为：

```
SQL#gt; alter table sp_item storage (freelists 2);
```

**Table altered.** 对于**Oracle8i**而言，增加**freelist**参数，在很多时候可以明显缓解等待，假如使用**LMT**，也就是 **Local Manangement Tablespace**，区段的治理就相对简单还可以考虑修改数据块的**pctused\pctfree**值，比如增大**pctfree**可以扩大数据的分布，在某种程度上就可以减少热点块的竞争。假如这一等待位于**undo header**，可以通过增加回滚段（**rollback segment**）来解决缓冲区的问题。

假如等待位于**undo block**上，我们可能需要检查相关应用，适当减少大规模的一致性读取，或者降低一致性读取(**consistent read**)的表中的数据密度或者增大**DB\_CACHE\_SIZE**。

假如等待处于**data block**，可以考虑将频繁并发访问的表或数据移到另一数据块或者进行更大范围的分布（可以增加**pctfree** 值，扩大数据分布，减少竞争），以避开这个"热点"数据块，或者可以考虑增加表中的自由列表或使用本地化治理的表空间（**Locally Managed Tablespaces**）。

假如等待处于索引块，应该考虑重建索引、分割索引或使用反向键索引。反向键索引在很多情况下，可以极大地缓解竞争，其原理有点类似于**hash**分区的功效。反向键索引（**reverse key index**）常建在一些值是连续增长的列上，例如列中的值是由**sequence**产生的。为了防止与数据块相关的缓冲忙等待，也可以使用较小的块：在这种情况下，单个块中的记录就较少，所以这个块就不是那么"繁忙"；或者可以设置更大的**pctfree**,使数据扩大物理分布，减少记录间的热点竞争。

在执行**DML (insert/update/ delete)**时，**Oracle**向数据块中写入信息，对于多事务并发访问的数据表，关于**ITL**的竞争和等待可能出现，为了减少这个等待，可以增加**initrans**，使用多个**ITL**槽。

以下是一个生产系统**v\$waitstat** 试图所显示的等待信息：

```
SQL#gt; select * from v$waitstat where count#lt;#gt;0 or time #lt;#gt;0;
```

**CLASS COUNT TIME**

-----

**data block 453 6686**

**undo header 391 1126**

**undo block 172 3**

**latch free: latch释放**

**latch** 是一种低级排队机制，用于保护SGA 中共享内存结构。

**latch**就像是一种快速地被获取和释放的内存锁。**latch**用于防止共享内存结构被多个用户同时访问。假如**latch**不可用，就会记录**latch**释放失败(**latch free miss**)。

有两种与**latch**有关的类型：

- 马上。
- 可以等待。

假如一个进程试图在马上模式下获得**latch**，而该**latch**已经被另外一个进程所持有，假如该**latch**不能马上可用的话，那么该进程就不会为获得该**latch**而等待。它将继续执行另一个操作。

大多数**latch** 问题都与以下操作相关：

没有很好的是用绑定变量（**library cache latch**）、重作生成问题（**redo allocation latch**）、缓冲存储器竞争问题（**cache buffers LRU chain**），以及**buffer cache**中的存在"热点"块（**cache buffers chain**）。

通常我们说，假如想设计一个失败的系统，不考虑绑定变量，这一个条件就够了，对于异构性极强的系统，不使用绑定变量的后果是极其严重的。

另外也有一些**latch** 等待与bug 有关，应当关注Metalink 相关bug 的公布及补丁的发布。

当**latch miss ratios**大于0.5%时，就应当研究这一问题。

Oracle 的 **latch** 机制是竞争，其处理类似于网络里的CSMA/CD，所有用户进程争夺**latch**，对于愿意等待类型(**willing-to-wait**)的**latch**,假如一个进程在第一次尝试中没有获得**latch**,那么它会等待并且再尝试一次,假如经过**\_spin\_count** 次争夺不能获得**latch**, 然后该进程转入睡眠状态，持续一段指定长度的时间，然后再次醒来，按顺序重复以前的步骤.在8i/9i 中默认值是 **\_spin\_count=2000**。

假如SQL语句不能调整，在8.1.6版本以上，Oracle提供了一个新的初始化参数：**CURSOR\_SHARING**，可以通过设置**CURSOR\_SHARING = force** 在服务器端强制绑定变量。设置该参数可能会带来一定的副作用，对于Java的程序，有相关的bug，具体应用应该关注Metalink的bug公告。

**enqueue**

**enqueue** 是一种保护共享资源的锁定机制。该锁定机制保护共享资源，如记录中的数据，以避免两个人在同一时间更新同一数据。**enqueue** 包括一个排队机制，即FIFO（先进先出）排队机制。

**Enqueue** 等待常见的有ST、HW 、TX 、TM 等

**ST enqueue** 用于空间治理和字典治理的表空间(DMT)的分配。对于支持LMT 的版本，可以考虑使用本地治理表空间，对于Oracle8i，因为相关bug 不要把临时表空间设置为LMT. 或者考虑预分配一定数量的区。

**HW enqueue** 指段的高水位标记相关等待；手动分配适当区段可以避免这一等待。

**TX** 是最常见的**enqueue** 等待。**TX enqueue** 等待通常是以下三个问题之一产生的结果。

第一个问题是唯一索引中的重复索引，你需要执行提交（**commit**）/回滚（**rollback**）操作来释放**enqueue**。

第二个问题是对同一位图索引段的多次更新。因为单个位图段可能包含多个行地址（**rowid**），所以当多个用户试图更新同一段时，等待出现。直到提交或回滚， **enqueue** 释放。

第三个问题，也是最可能发生的问题是多个用户同时更新同一个块。假如没有自由的ITL 槽，就会发生块级锁定。通过增大**initrans** 和/或**maxtrans** 以答应使用多个ITL 槽，或者增大表上的**pctfree**值，就可以很轻松地避免这种情况。

**TM enqueue** 在DML 期间产生，以避免对受影响的对象使用DDL。假如有外键，一定要对它们进行索引，以避免这种常见的锁定问题。

**Log Buffer Space:** 日志缓冲空间

当你将日志缓冲（**log buffer**）产生重做日志的速度比LGWR 的写出速度快，或者是当日志转换（**log switch**）太慢时，就会发生这种等待。为解决这个问题，可以增大日志文件的大小，或者增加日志缓冲器的大小。

另外一个可能的原因是磁盘I/O 存在瓶颈，可以考虑使用写入速度更快的磁盘。

**log file switch (archiving needed)**



这个等待事件出现时通常是因为日志组循环写满以后，第一个日志归档尚未完成，出现该等待可能是 IO 存在问题。

解决办法：

可以考虑增大日志文件和增加日志组

移动归档文件到快速磁盘

调整log\_archive\_max\_processes .

log file switch (checkpoint incomplete): 日志切换（检查点未完成）

当你的日志组都写完以后，LGWR 试图写第一个log file，假如这时数据库没有完成写出记录在第一个log file 中的dirty 块时（例如第一个检查点未完成），该等待事件出现。

该等待事件说明你的日志组过少或者日志文件过小。

你可能需要增加你的日志组或日志文件大小。

Log File Switch: 日志文件转换

所有的提交请求都需要等待"日志文件转换（必要的归档）"或"日志文件转换（chkpt.不完全）"。确保归档磁盘未滿，并且速度不太慢。DBWR 可能会因为输入/输出（I/O）操作而变得很慢。你可能需要增加更多或更大的重做日志，而且假如DBWxR 是问题症结所在的话，可能需要增加数据库书写器。

log file sync: 日志文件同步

当一个用户提交或回滚数据时，LGWR 将session 会话的重做由redo buffer 写入到重做日志中。

log file sync 必须等待这一过程成功完成(Oracle 通过写redo log file 保证commit 成功的数据不丢失)，这个事件说明提交可能过于频繁，批量提交可以最大化LGWR 的效率，过分频繁的提交会引起LGWR频繁的激活，扩大了LGWR 的写代价。

为了减少这种等待事件，可以尝试每次提交更多的记录。

将重做日志置于较快的磁盘上，或者交替使用不同物理磁盘上的重做日志，以降低归档

对LGWR的影响。

对于软RAID，一般来说不要使用RAID 5，RAID5 对于频繁写入得系统会带来较大的性能损失，可以考虑使用文件系统直接输入/输出，或者使用裸设备（raw device），这样可以获得写入的性能提高。

log file single write

该事件仅与写日志文件头块相关，通常发生在增加新的组成员和增进序列号时。头块写单个进行，因为头块的部分信息是文件号，每个文件不同。更新日志文件头这个操作在后台完成，一般很少出现等待，无需太多关注。

log file parallel write

从log buffer 写redo 记录到redo log 文件，主要指常规写操作(相对于log file sync)。

假如你的Log group 存在多个组成员，当flush log buffer 时，写操作是并行的，这时候此等待事件可能出现。

尽管这个写操作并行处理，直到所有I/O 操作完成该写操作才会完成(假如你的磁盘支持异步IO或者使用IO SLAVE，那么即使只有一个redo log file member,也有可能出现此等待)。

这个参数和log file sync 时间相比较可以用来衡量log file 的写入成本。通常称为同步成本率。

control file parallel write: 控制文件并行写

当server 进程更新所有控制文件时，这个事件可能出现。

假如等待很短，可以不用考虑。假如等待时间较长，检查存放控制文件的物理磁盘I/O 是否存在瓶颈。

多个控制文件是完全相同的拷贝，用于镜像以提高安全性。对于业务系统，多个控制文件应该存放在不同的磁盘上，一般来说三个是足够的，假如只有两个物理硬盘，那么两个控制文件也是可以接受的。在同一个磁盘上保存多个控制文件是不具备实际意义的。

减少这个等待，可以考虑如下方法：

减少控制文件的个数(在确保安全的前提下)

假如系统支持，使用异步IO

转移控制文件到IO 负担轻的物理磁盘

control file sequential read/ control file single write

控制文件连续读/控制文件单个写

对单个控制文件I/O 存在问题时，这两个事件会出现。

假如等待比较明显，检查单个控制文件，看存放位置是否存在I/O 瓶颈。

使用查询获得控制文件访问状态：

```
select P1 from V$SESSION_WAIT
where EVENT like 'control file%' and STATE='WAITING';
```

解决办法：

移动有问题的控制文件到快速磁盘

假如系统支持，启用异步I/O

direct path write: 直接路径写

该等待发生在，等待确认所有未完成的异步I/O 都已写入磁盘。

你应该找到I/O 操作频繁的数据文件，调整其性能。

也有可能存在较多的磁盘排序，临时表空间操作频繁，可以考虑使用Local 治理表空间，分成多个小文件，写入不同磁盘或者裸设备。

SQL\*Net message from dblink

该等待通常指与分布式处理（从其他数据库中SELECT）有关的等待。

这个事件在通过DBLINKS 联机访问其他数据库时产生。假如查找的数据多数是静态的，可以考虑移动这些数据到本地表并根据需要刷新，通过快照或者物化视图来减少跨数据库的访问，会在性能上得到很大的提高。

slave wait: 从属进程等

Slave Wait 是Slave I/O 进程等待请求，是一个空闲参数，一般不说明问题。 2.2.4 High Load

SQL 分析

对于一个特定的应用程序或者系统来讲，要调整优化其性能，最好的方法是检查程序的代码和用户使用的SQL语句。

假如使用了 level 5 级别的 snapshot ，那么statspack生成的报告中就会显示系统中高负荷SQL语句（High Load SQL）的信息，而其具体信息可以在 stats\$sql\_summary 表中查到。缺省情况下 snapshot 的级别是 level 5。

按照 buffer gets, physical reads, executions, memory usage and version count 等参数的降序排列顺序，把SQL语句分为几个部分罗列在报告中。 2.2.5 报告的其他部分

statspack报告的其他部分包括了 Instance Activity Stats, Tablespace IO Stats, Buffer Pool Statistics, Buffer wait Statistics, Rollback Segment Stats, Latch Activity, Dictionary Cache Stats, Library Cache Activity, SGA breakdown difference 以及 init.ora 参数，等等。目前本文不对这些内容进行具体讨论，请参加其他具体文档。 2.3 trace session （.....）

2.4 基于成本的优化器技术内幕

Oracle基于成本的优化器（Oracle's cost-based SQL optimizer ，简称CBO），是Oracle里面非常复杂的一个部分,它决定了Oracle里面每个SQL的执行路径。CBO是一项评价SQL语句和产生最好执行计划的具有挑战性的工作，所以也使它成Oracle最复杂的软件组成部分。

众所周知，SQL的执行计划，几乎是Oracle性能调整最重要的方面了。所以想要学会如何调整Oracle数据库的性能，就要学会如何对SQL进行调整，就需要深入透彻理解CBO。

CBO的执行路径，取决于一些外部因素，内部的Oracle统计数据，以及数据是如何分布的。我们将要讨论下面的话题：

CBO的参数：我们从基本的优化器参数开始学习，然后学习每个优化器参数是如何影响Oracle的优化器的执行的。 CBO的统计：这里我们将讨论，使用Analyze或者DBMS\_STATS来收集正确的统计数据，对Oracle 优化器而言，是多么的重要。我们还将学习如何把优化器的统计数据，从一个系统拷贝到另外一个系统，这样可以确保开发环境和产品数据库环境下，SQL的执行路径不会变化。 下面我们开始讨论CBO优化模式以及影响CBO的Oracle参数 2.4.1 CBO的参数

CBO受一些重要参数的影响，修改这些参数后可以看到CBO性能上戏剧性的变化。首先从设置CBO的optimizer\_mode参数开始，然后讨论其他重要参数的设置。 在 Oracle 9i 中， optimizer\_mode 参数有四种取值，决定了四种优化模式： rule, choose, all\_rows, 和

first\_rows，其中 rule 和 choose 两种模式表示目前已经过时的基于规则的优化器模式（rule-based optimizer，简称RBO），所以我们在此着重讨论后两种CBO模式。 优化模式的设置可以在系统级进行，也可以对某个会话（session）进行设置，或者对某个SQL语句进行设置。对应的语句如下：

```
alter system set optimizer_mode=first_rows_10;
```

```
alter session set optimizer_goal = all_rows;
```

select /\*+ first\_rows(100) \*/ from student; 我们首先需要知道对一个SQL语句来说，什么是最好的执行计划（the best execution plan）？ 是使SQL语句返回结果的速度最快，还是使SQL语句占用系统资源最少？显然，这个答案取决于数据库的处理方式。 举一个简单的例子，比如有下列SQL语句：

```
select customer_name
from
customer
where
region = 'south'
order by
customer_name;
```

假如最好的执行



Tags: ORACLE    Tuning    ORACLE    性能    调整

- 上一篇：安装Oracle后，经常使用的修改表空间的SQL代码
- 下一篇：Oracle Pro\*C/C++ 游标和存储过程性能测试报告

相关文章列表

教你怎样在Oracle数据库中高速导出/导  
史上最简单的方法复制或迁移Oracle数据  
实例解析：工作中遇到的Oracle故障分析  
Oracle 10g 简直是吃系统资源的疯子！

解读Oracle11g在商业银行的三大应用亮  
解读ORACLE数据库的统一命名与编码规范  
Oracle10g中的current\_scn是如何计算的  
为什么Oracle中只能用sys和system登录