

## 第18章 调整内存

本章要点：

UTLBSTAT/UTLESTAT

调整共享池

调整数据库缓存

调整排序

调整多线程服务器

调整锁

再看操作系统集成

在Oracle中，调整内存通常是指调整共享全局区（SGA）。这包括监控和调整共享池（数据字典与库缓存）与数据库缓存。调整内存与调整应用和输入/输出紧密联系在一起，因为调整的一个主要目标——减少或消除争用——在调整过程的各方面都涉及到。例如，本书在调整输入/输出中讨论了它们都有必须调整的内存成分，例如回滚缓冲区和重做日志缓冲器。然而，因为主要强调的是输入/输出争用，所以在调整输入/输出里讨论调整回滚段和重做日志。你还必须调整其他的诸如锁与门的基于内存的结构，其中一部分将在本章讨论。分类排序是另外一个内存问题。当你对任何一样东西进行分类排序时，不管它是否是 CREATE INDEX、ORDER BY 或一个连结的结果，理想的情况是尽可能地在内存中执行，仅在必要时求助于磁盘。最终，作为一名数据库管理员，你必须积极地参与到操作系统的集成和调整中。

除非RDBMS由专用的数据库计算机的部分组成，否则包括Oracle在内的任何RDBMS必须从操作系统中申请内存。操作系统在低层处理大部分的内存管理，例如用于RDBMS的共享内存。依赖于RDBMS，它可以处理某些高级操作，例如锁定。作为一个示例，在内存管理方面考察一下Sybase与Oracle。Sybase选择使用一个大型的共享内存段并且通过低级操作系统调用来管理它本身。Oracle选择同样的方法。Sybase处理它自己的锁定和多线程过程中的进程间通信问题。另一方面，Oracle管理部分锁定和内部的进程间通信，还要依赖操作系统做一部分工作。这些处理内存的方法是RDBMS体系结构的直接结果。不论发生哪种情况，一个微型操作系统和其中的RDBMS必须和操作系统紧密结合，这样可以有效地处理它的资源请求。这不仅适用于内存，而且适用于所有的资源，包括输入/输出。但有几个例外。在UNIX操作系统中的原始磁盘空间就是一个明显的例外（参见附录A）。现在，把你的注意力转向实际的数据收集和诊断内存问题上来。

### 18.1 UTLBSTAT/UTLESTAT

正如你在第16章中所学习到的那样，UTLBSTAT和UTLESTAT脚本为你的全部诊断工作打下基础。经常用人工查询或诸如V\$SYSSTAT的V\$动态性能视图的自定义脚本来补充这些脚本。通过在不同的工作中使用这些脚本，你可以用它们来保证同其他Oracle数据库管理员和顾问针对某一系统的性能进行通信。

正如你看到的，这些脚本通常存在于 \$ORACLE\_HOME/rdbms<version>/admin 子目录中。

在实际使用它们以前，应在你的 `init.ora` 参数文件中设置 `TIMED_STATISTICS=TRUE` 或在你的会话层设置 `ALTER SESSION SET TIMED_STATISTICS=TRUE`。注册到 `svrmgr1`（服务器管理器行方式，版本 7.1 或更高）。接下来 `CONNECT/AS SYSDBA`（或者是 `CONNECT INTERNAL`）。然后，运行 `utlbstat.sql`。

这样就创建了你的开始集合表和视图（在 `SYS` 模式下）；这些对象已经在它们内部用 `BEGIN` 命名。然后选取开始统计数字并把它们存放在那里。接下来，假如你的应用还没有运行的话，那么运行它。作为一名数据库管理员，你的目标是在系统的峰值活动期间捕获系统统计数字。当峰值过后，或在一些合理的时间周期之后，运行 `utlestat.sql`。此时间间隔不必太长，但是必须充足。例如，假如你的峰值装载活动每天持续 2 小时（120 分钟），那么采样至少应是该时间的 20%，即大约 24 分钟，最好取该间隔的中间值（还有，你的数据库必须在该采样运行的整个期间内保持不变。这不会成为一个问题，然而，因为你正试图从一个活动的、峰值负载的生产数据库中采样，所以希望数据库无论如何要保持不变）。这样就创建了你的结束和差别集合表与视图。结束对象已经在它们内部用 `END` 命名。结束统计数字被选取并存储在那里。开始与结束统计数字之间的差别存储在差别表中。最终，`utlestat.sql` 从对象、格式化数据选取并在 `report.txt` 文件中存储信息。然后，困难的工作——解释开始了。

### 18.1.1 解释结果

不会有两个顾问在对 Oracle 进行调整的每一个细节上存在相同的意见，但是他们会同意大部分事情。经常听到有人说性能调整一半是科学，一半是艺术。我对这个评价表示赞同。换句话说，解释大部分是艺术，因为那部分工作需要人对系统的实力和弱点做出有价值的判断。

使用高级语法分析和程序的解释可以是并且已经是半自动或全自动进行的。然而，不可否认的事实是不管这个过程如何程序化，它仍然具有部分主观性。尽管它可以被编程，然而相同的两个调整程序或程序员会使用相同的规则或准则来判断“好的”或“差的”性能水平吗？大部分不能。此外，这仅考虑了通用调整。当你面对特殊的应用问题时，这些所谓的完全自动的性能调整程序很快就会失去它们的用处。

现在我将总结一下如何正确地使用 `UTLBSTAT/UTLESTAT` 与调整你的数据库系统：

- 1) 在实例或会话层设置 `TIMED_STATISTICS=TRUE`。
- 2) 登录到 `svrmgr1`（服务器管理器行方式）并 `CONNECT/AS SYSDBA`（或 `CONNECT INTERNAL`）。
- 3) 在监控期开始时运行 `$ORACLE_HOME/rdbms<version>/admin/utlbstat.sql`，其中监控期是正常的峰值活动的持续时间。
- 4) 在监控期结束时运行 `$ORACLE_HOME/rdbms<version>/admin/utlestat.sql`。
- 5) 通过使用合理的准则对 `report.txt` 中的结果进行解释（不久我会涉及到准则问题）。
- 6) 完成推荐的修改或调整（如果有的话）；这可以从修改一个 `init.ora` 参数到重新组织你的物理设计的任何事情。
- 7) 从步骤 3 开始重复此过程，直到对步骤 5 的结果满意为止。

### 18.1.2 回顾报告文件

`report.txt` 文件中包含了大量用来帮助你调整应用、内存和输入/输出的信息。在高层次上，

它关于共享池（数据字典与库缓存）、数据库缓冲区缓存、每个事务/登录数据、每个表空间/文件的输入/输出和等待事件的统计数字。你可以使用所有这些统计数字或它们中的一部分。

提示 确信你使用的单个统计数字与你的应用有关。在最后的分析中，你的用户会真正测试你的应用性能是否是“优良的”。

有些统计数字对同一项性能提供了不同的视图而且应一致。当这些视图不一致时，解释成为主观事物和应用特定的。例如，假如你有 3 个统计数字，以不同的方式报告有关库缓存的性能。如果这 3 个中的 2 个根据已接受的准则显示性能是“优良的”，而剩余的那一个显示性能是“不良的”。如果出现这种情况怎么办？这意味着你的应用性能是优良的吗？这就要依赖于你的应用类型以及它与它有关的每一个统计量的含义。例如，假如显示优良性能的两个统计量对于批处理系统更有意义，而你有一个 OLTP 系统，那么那些测量值会令人误解。

## 18.2 调整共享池

正如你在第 6 章中所学习到的那样，共享池基本上包括两个主要的结构：

数据字典缓存。

库缓存。

那些存储供稍后重新使用的已经过语法分析的 SQL 语句的区域是共享的 SQL 区域。专用的 SQL 区是一些与应用内游标持续时间有关的区域。在共享池的调整中，调整内存与调整应用明显重叠。本章增强了前面的一章（第 17 章）。

共享池是一个高速缓冲结构。和所有其他的高速缓存结构一样，它是内存驻留数据结构。

高速缓存（cache）是一种特殊类型的缓冲区。尽管缓冲区是一个“无智能的”机制，仅用来为数据在高速内存与慢速磁盘之间的路线上提供临时存储区，但高速缓存是一个“灵巧”的机制，保留关于那条信息的内存或部分信息的内存，因此高速缓存可以尽可能地避免到磁盘的不必要的行程。当一条输入/输出请求发出后，高速缓存查看该请求要求的数据是否已在内存中。如果数据在内存的话，它亲自回答该请求，返回请求的数据。这称为一个命中（hit）。如果数据不在内存的话，就会有一次到磁盘的访问。这被称为一个遗漏（miss）。

对于几乎所有的高速缓存装置来说，保证高效性能的准则是 90%+命中率，命中率可以被定义为  $100 \times (1.00 - (\text{遗漏的数量} / \text{请求的数量}))$ ，其中请求的数量 = 遗漏的数量 + 命中的数量。例如，假如你的高速缓存有 4 个遗漏和 46 个命中，那么你的命中率是  $100 \times (1.00 - (4/50)) = 100 \times (1.00 - 0.08) = 92\%$ ，这个数字非常好。高速缓存通常由一个最近最少使用的（LRU）算法来管理，该算法保证在任何给定的时刻，最近使用最多的（MRU）数据被保存在高速缓存中并且最近最少使用的数据被拿走。

当 Oracle 对一条 SQL 语句进行语法分析时，通过把一个数学公式应用到 SQL 语句的字母数字文本并使用该结果在高速缓存中存储（供以后查找和利用）它，Oracle 便可以在库缓存（library cache）中为该 SQL 语句分配一个 SQL 区。换句话说，它使用一个哈希函数（hash function）。为了一条语句可以被重复使用，该语句必须是相同的。例如，当在库缓存中对下述语句进行存储时，在 Oracle 看来这些语句并不是相同的：

```
SELECT * FROM EMPLOYEES;  
SELECT      * FROM EMPLOYEES;  
SELECT * FROM employees;
```

尽管对于你我来说，它们在功能上是相同的，但是它们的同一个哈希值不同。为了使它们成为相同的哈希值，在语句中应没有空白（空格、制表键、缩进或不可打印的控制字符）或其他不同。此外，下面的两条语句不能重复使用相同的哈希 SQL 区语法规则：

```
SELECT * FROM EMPLOYEES WHERE EMPLOYEE_ID=927354;  
SELECT * FROM EMPLOYEES WHERE EMPLOYEE_ID=746293;
```

这是事实，因为在 WHERE 子句中使用了直接量 927354 和 746293。因为在哈希函数中字母数字语句的输入是不同的，所以它们不可能放到相同的库缓存位置。如何才能放到相同的位置？除了较早前讨论过的 DSS 外，你可以使用绑定变量。绑定变量（bind variable）使 SQL 语句足够通用以便重新使用并且具有参数值而不是常量：

```
SELECT * FROM EMPLOYEES WHERE EMPLOYEE_ID=:EMPID;
```

这种类型的语句可以被重复使用并且一般存在于嵌入式 SQL 应用代码中，其中：EMPID 是绑定变量——在该例子中，主 SQL 变量是一个 C 整型。这个 C 变量的值现在可以采用 927354、746293 等等，可以在库缓存中重复使用。

### 18.2.1 改进库缓存性能的方针

将应用内部的不必要的语法分析调用减到最小。语法分析耗费大量的 CPU 时间。因为高速缓存和缓冲区被用到，作为一个副作用，它也要耗费大量的内存。游标打开和关闭应被仔细地放置在应用中，以方便为多条 SQL 语句重复使用专用 SQL 区。数据库管理员必须增加 init.ora 的参数 OPEN\_CURSORS 以便为分配足够的游标空间（专用 SQL 区）做准备。要确定你的应用的效率是否低下，运行 SQL TRACE/tkprof（第 17 章）并检查用于语法分析的计数列是否接近于执行（或取数据）的值。假如这样的话，应用为几乎每一次执行（或取数据）重新进行语法分析。

最大程度地重复使用那些必须被分析的语句。如上所述，SQL 语句要被重复使用必须相同。一种有助于重复使用的方法是采用所有应用开发者必须遵守的标准编码方式，例如“总是用大写字母编写 SQL 语句”。你可以让一个程序通过你的所有开发代码并在开发者的适应性较差的情况下实施它。此外，除了 DSS 应用外，在适当的时候使用绑定变量。这很好理解，与硬编码的直接量相比，绑定变量使你的应用通用化了。如果没有库缓存重用，这在可维护性方面付出了代价。

“钉住”（pin）内存中频繁使用的程序对象。在 Oracle 中，如果使用一个特殊的共享池包 DBMS\_SHARED\_POOL，那么游标、触发器、过程或包可以保持在内存中。为了创建这个包，运行 \$ORACLE\_HOME/rdbms<version>/admin/dbmspool.sql 脚本。你或许还需要运行 \$ORACLE\_HOME/rdbms<version>/admin/prvtpool.sql。查看你的操作系统平台版本以判断情况是否如此。要钉住内存中的一个程序对象，使用如下语句：

```
SQL> EXECUTE DBMS_SHARED_POOL.KEEP('<object_name>');
```

要解除这个固定：

```
SQL> EXECUTE DBMS_SHARED_POOL.UNKEEP('<object_name>');
```

要确定对象是否被成功地钉住：

```
SQL> SELECT SUBSTR(NAME,1,25), KEPT FROM V$DB_OBJECT_CACHE;
```

假如对象被钉住，那么 KEPT 列的值为 YES，否则为 NO。

把库缓存中的碎片减到最小。如果你不消除碎片的话，你的应用会遇到 ORA-04031 错误

(没有足够的连续空间)。一种方法是钉住内存中频繁使用的大对象。对于并不频繁使用的大对象,则为它们保留一些空间。你可以通过设置 init.ora 的参数 SHARED\_POOL\_RESERVED\_SIZE 和 SHARED\_POOL\_RESERVED\_MIN\_ALLOC 来完成这项工作。另外设置用于你的大对象的共享池“保留区”。基本上,要保证你所需的大对象能够找到空间。把 SHARED\_POOL\_RESERVED\_SIZE 设置为同时装载的最大对象的最大字节数。为了使用由 SHARED\_POOL\_RESERVED\_SIZE 指定的保留区,把 SHARED\_POOL\_RESERVED\_MIN\_ALLOC 设置为可以用于一个对象的最小字节数。要确定你希望在保留区中保存的一个特殊对象的大小,使用如下的语句:

```
SQL> SELECT SUBSTR(NAME,1,25) "NAME", SHARABLE_MEM  
2> FROM V$DB_OBJECT_CACHE  
2> WHERE NAME='<object_name>';
```

还有,要确定需要把 SHARED\_POOL\_RESERVED\_SIZE 设置为多大,使用如下语句:

```
SQL> SELECT SUM(SHARABLE_MEM)  
2> FROM V$DB_OBJECT_CACHE  
3> WHERE SHARABLE_MEM >= <SHARED_POOL_RESERVED_MIN_ALLOC>;
```

因此,要执行前面的查询,你必须有一些关于分类、最小值和“大对象”的粗略想法。所以,应采取如下步骤:

- 1) 按你的要求设置 SHARED\_POOL\_RESERVED\_MIN\_ALLOC。
- 2) 把 SHARED\_POOL\_RESERVED\_SIZE 设置为上一个查询的输出值,加上一个裕量(例如10%)。

再有,你可以把 CURSOR\_SPACE\_FOR\_TIME 设置为 TRUE 以免与游标有关的 SQL 区在被程序关闭以前从库缓存中删除。

**警告** 假如下面的任何一条符合你的状况时,那么不要改变 CURSOR\_SPACE\_FOR\_TIME 的 FALSE 缺省值:

V\$LIBRARY\_CACHE 中的 RELOADS 总是显示 0 值。

你正在使用 Oracle Forms 或 SQL \*Forms。

你使用任何动态的 SQL。

除了刚刚学过的许多基于应用的内存问题与调整方法外,你可以看一下 utlbstat.sql/utlestat.sql (report.txt) 提供些什么。一个 report.txt 的示例放在本书附带的 CD-ROM 上。特别是,你可以看一下 report.txt 中的第一部分,即库缓存统计数字。确保用于 SQL 区库的 GETHITRATIO 在 90s 内,最好是高 90s。假如你不运行 utlbstat.sql/utlestat.sql,你还可以使用如下语句:

```
SQL> SELECT GETHITRATIO  
2> FROM V$LIBRARYCACHE  
3> WHERE NAMESPACE='SQL AREA';
```

然而,此方法是自上一个实例启动以来并包含了上斜坡(ramp-up)时间的平均运行时间。上斜坡是任何系统开始启动时经历的初始化过程。在此期间收集到的统计数字会使人误解并且没有用处,因为它们基本上反应了必须填充各种 Oracle 缓冲区和高速缓存的物理读取。因此,有时这也叫做缓存上升(caching up)。正如你较早前学到的那样,希望在峰值装载时间里采样统计数字。当一个系统在峰值活动中运行了一段时间时,该系统被认为是处于平衡状态

( in equilibrium )。假如你在峰值活动到达后运行 utlbstat.sql，然后在峰值活动刚要停止前运行 utlestat.sql，你就有了一个有效的样本。因此，从V\$动态性能视图中选取统计数字会令人误解，特别是如果你在实例持续期的早期采样它们。实例持续时间越长，V\$视图的统计数字就越好，因为峰值与正常活动越来越多地超过了初始启动期。要保持准确的话，对 utlbstat.sql/utlestat.sql也应该这样。

---

假如GETHITRATIO小于0.90，运用改进库缓存性能的准则。这一般意味着应用代码可以被更有效地重新编写。

---

此外，从report.txt的第一部分（即库缓存统计数字）中计算用于SQL区库的RELOADS/PINS率。

---

假如RELOADS/PINS率大于0.01，通过设置init.ora的参数SHARED\_POOL\_SIZE来增加总共享池的容量（以字节为单位）。记住，这设置了整个共享池，不仅包括库缓存，而且包括数据字典缓存。

---

检查report.txt的统计数字部分，确保对于LATCH\_NAME='library cache'的HIT\_RATIO处于高90s中。其他测量高门争用的方法如下所示，高门争用是对不良的库缓存（和总共享池）性能的提示。

查看report.txt的System wide wait events部分，检查Event Name='latch free'的（wait）Count column。

还要运行如下查询：

```
SQL> SELECT COUNT(*) "LIBRARY_LATCH_WAITS"  
2> FROM V$SESSION_WAIT W, V$LATCH L  
3> WHERE W.WAIT_TIME = 0  
4> AND W.EVENT='latch free'  
5> AND W.P2 = L.LATCH#  
6> AND L.NAME LIKE 'library%';
```

---

如果下面任何一项是正确的话，运用改进库缓存性能的准则；如果必要，增加SHARED\_POOL\_SIZE的设置值：

LATCH\_NAME='library cache'的HIT\_RATIO小于0.98

Event Name='latch free'的（wait）Count大于10

LIBRARY\_LATCH\_WAITS大于2

---

正如你所预料的一样，共享池的数据字典缓存（data dictionary cache）部分保留用于Oracle数据字典的缓存结构。SHARED\_POOL\_SIZE参数是间接测量数据字典缓存大小的唯一方法。下面是保存在SYS和SYSTEM模式（SYSTEM表空间）中的对象：

X\$表。

V\$视图。

数据库管理员视图。

用户视图。

度量、诊断与调整库缓存以便它很好地运行，这样有助于数据字典缓存提高其性能，因为库缓存和数据字典缓存都共存于共享池中。它们不是分别配置的。

有两种方法来测量数据字典缓存性能。其中一种方法是查询V\$ROWCACHE视图：

```
SQL> SELECT SUM(GETMISSES)/SUM(GETS) "DC_MISS_RATIO"
```

```
2> FROM V$ROWCACHE;
```

另一种方法是使用 report.txt 的数据字典部分。计算所有的 GET\_MISS 的总和，用所有 GET\_REQS 的总和除以 GET\_MISS 的总和，得到一个类似的 DC\_MISS\_RATIO。

如果这两种方法中的任何一种产生的 DC\_MISS\_RATIO 大于 0.15，那么增加 SHARED\_POOL\_SIZE（然后重新测试）。

### 18.2.2 多线程服务器问题

你还要简要考虑多线程服务器（MTS）和服务内存与客户（用户）内存的分配问题。有一个 SGA 就有一个用户全局区（UGA），它包含了用户会话信息、排序区和专用 SQL 区。通常，缺省的 Oracle RDBMS 实例（称为专用的服务器）导致一个用户过程与服务器过程的一对一映射。使用多线程服务器，用户全局区被移动到共享池中。剩余的进程特定的内存被保留在进程全局区（PGA）中并且它保存不能共享的信息。在这种方法中，使用多线程服务器时需要的内存总数不会超过再分配的专用服务器。然而，你必须增加 SHARED\_POOL\_SIZE。要帮助对重新定位到 SGA 的用户全局区进行度量，运行如下查询：

```
SQL> SELECT SUM(VALUE)
2> FROM V$SESSTAT SE, V$STATNAME SN
3> WHERE SN.NAME = 'max session memory'
4> AND SE.STATISTIC# = SN.STATISTIC#;
```

该查询产生了自实例启动以来已使用的 UGA 会话内存的最大数量。你或许希望多次采样它并得到这些最大值中的最大值，然后把 SHARED\_POOL\_SIZE 增加到此最大值。

注意 在 Oracle 8 中，对于上一个查询使用 'session uga memory max'。

使用多线程服务器，你可以对服务器和用户内存的分布实施一些控制。影响用户内存的两个 init.ora 参数是：

```
SESSION_CACHED_CURSORS
CLOSE_CACHED_OPEN_CURSORS
```

假如希望的话，把 SESSION\_CACHED\_CURSORS 设置为缓存在用户内存区中的会话游标的最大期望值。只要重新进行语法分析的工作量较小，这就有助于在以增加用户内存为代价的前提下卸载服务器内存请求。当语句被频繁地重复使用时，可选择是否设置这个参数。

CLOSE\_CACHED\_OPEN\_CURSORS 的缺省设置值是 FALSE，即游标不在 COMMIT 时关闭。假如大部分 SQL 语句很少被重复使用的话，可把该值设置为 TRUE。

**警告** 确保对 CURSOR\_SPACE\_FOR\_TIME 和 CLOSE\_CACHE\_OPEN\_CURSORS 的设置值不会发生冲突。例如，把两者都设置为 TRUE 或都设置为 FALSE 看来是最好的办法。

下面，把你的注意力转向数据库缓冲区缓存。

## 18.3 调整数据库缓冲区缓存

或许用来提高 Oracle 系统性能的一个最重要的调整变化是，正确地设置你的数据库缓冲区缓存的大小。数据库缓冲区缓存（database buffer cache）是 SGA 中的高速缓存结构，它可以把最近最多使用（MRU）的 Oracle 数据块的拷贝保存在内存中。度量该区大小的两个参数是：

DB\_BLOCK\_SIZE

DB\_BLOCK\_BUFFER

DB\_BLOCK\_SIZE是指一个Oracle数据块的大小。在UNIX操作系统平台上该值可以在2KB（2048个字节）和64KB（65536个字节）之间变化。对于性能来说，该值通常是越高越好。假如你的数据库已经用一个相对较小的块大小（例如为2KB的缺省值）创建，假如可行的话，考虑重新建立一个数据库。如果想重建数据库，执行下述步骤：

- 1) 关闭实例。
- 2) 把你的数据库完全导出（如果可行）。
- 3) 在你的init.ora中增加DB\_BLOCK\_SIZE。
- 4) 启动该实例。
- 5) 把你的数据库作为SYS重新导入。

如果你的数据库太大而不能使用导出/导入的话，你要确保有你的表数据的ASCII文件（如果必要；选取它们或者使用第三方工具）。重新运行你的DLL以创建脚本。可能的话，使用SQL\*Loader以并行方式重新装载该表。

DB\_BLOCK\_BUFFER是保存在内存中的Oracle数据块的数量。每个缓冲区等于一个数据块。该值应足够大以便产生一个有效的高速缓存命中率，但又不会过高以致于引起操作系统分页。最后你希望做的是通过操作系统把你的SGA在内存中或内存外分页。分页是当Oracle达到DB\_BLOCK\_BUFFER大小时完成的主要工作，你不希望在Oracle下进行操作系统内存分页。在有请求时才对它们进行分页。因此，你的数据库缓冲区缓存以及共享池将装配在实（可用的核心）内存中并且它们的大小不会接近或大于该内存。合理的大小是总系统内存的1/2到3/4。例如，在一个1G的UNIX系统中，你或许要设置数据库缓冲区缓存以便你的SGA占用大约3/4的总量，即750MB。你必须仔细地考虑其他的Oracle与非Oracle应用内存要求、用户内存要求以及操作系统要求。数据库缓冲区缓存的容量是：

DB\_BLOCK\_BUFFERS x DB\_BLOCK\_SIZE

数据库缓冲区缓存有点用词不当。回忆一下高速缓存是一种特殊类型的缓冲区。因此，缓冲区缓存这个词实际上是多余的，有点混乱（或许Oracle本应该叫它数据库块缓存？）。总之，你需要真正理解的是数据库缓冲区缓存保存了Oracle的数据块。它与共享池不同，因为它保存数据，而不保存程序。

在把Oracle数据块传送到用户进程以前，Oracle RDBMS服务器总是把Oracle数据块读取到数据库缓冲区缓存中。用户进程或应用总是从数据库缓冲区缓存中读取（和写入）。下面是在缓冲区管理中的输入/输出请求步骤：

- 1) 用户选择数据（请求数据块）。
- 2) 服务器在数据库缓冲区缓存中查找它。
- 3) 如果通过哈希函数，服务器在LRU列表中发现了用户选取的数据，那么服务器便返回它。
- 4) 如果服务器没有发现它，那么服务器从磁盘上的数据文件块中读取并在适当时把它依附于（使用哈希函数）LRU列表的MRU或LRU端。
- 5) 如要用户不修改它，那么这个输入/输出请求步骤就完成了。
- 6) 如果用户修改它，DBWR把数据块（不干净的缓冲区）写回到它在磁盘上数据文件中

的位置。

一次只能有一个数据块访问索引。全表扫描可以用一条请求完成多个数据块的读取。通过如下的设置来设置块（批量大小）的数量：

```
DB_FILE_MULTIBLOCK_READ_COUNT=<要被读取的块的数量>
```

缓冲区可以是空闲的（干净的）、肮脏的、当前的或一致性读取的（回滚）。一个空闲缓冲区（free buffer）是指自从实例启动以来还没有被使用或曾经被使用过但目前可用的缓冲区。脏缓冲区（dirty buffer）是指已被使用，但还没有被刷新或者在检查点由 DBWR存储的缓冲区。当前缓冲区（current buffer）是指用于INSERT、UPDATE或DELETE服务中的缓冲区。从本质上来说，当前缓冲区更容易变脏。一致性读取（Read-consistent）缓冲区对SELECT语句和回滚进行服务。在全表扫描服务中读取的块被放置在 LRU缓冲区链的最近最少使用（LRU）端。然而，你可以在该链的MRU端缓存整个表。

如何调整数据库缓冲区缓存？因为内存输入/输出比磁盘输入/输出快几个数量级（它们是纳秒与毫秒的关系），所以希望尽可能通过内存满足输入/输出请求。这也就是说，与必须从磁盘中读取数据块相反，希望它们经常（超过总时间的 90%）存放在数据库缓存中。还希望把LRU门争用减到最小。LRU缓冲区链或列表通过门机制被锁定，就像那些用于 Oracle内核的缓冲区链一样，尤其是库缓存中的缓冲区链。如同任何一种门方法一样，你必须有足够的缓存，因为门（也称为自旋锁）和信号量一样不包含查询机制。

正如在调整库缓存中所提及的那样，几乎任何一个缓存结构的缓存命中率大于或等于 90%就被认为是好的。还有，缓存命中率是命中与请求之比。换句话说，它是 Oracle数据块满足输入/输出请求的次数除以总输入/输出请求数所得到的结果。当数据块在缓存中的时候叫做命中，当数据块不在缓存中的时候叫做遗漏（必须从磁盘中读取）。至少有两种方法来测量数据库缓冲区缓存的命中率。一种方法是运行如下的查询：

```
SQL> SELECT 1-(P.VALUE/(D.VALUE+C.VALUE)) "CACHE HIT RATIO"  
2> FROM V$SYSSTAT P, V$SYSSTAT C, V$SYSSTAT D  
3> WHERE P.NAME='physical reads'  
5> AND D.NAME='db block gets'  
4> AND C.NAME='consistent gets';
```

其中，'physical reads'是从磁盘中读出的数据块的数量，'db block gets'是从缓存的当前块拷贝中读出的数据块的数量，'consistent gets'是缓存中一致性读取（回滚）的块拷贝的数量。因此，数据库缓存命中率公式如下：

$$1 - (\text{physical reads} / \text{logical reads})$$

逻辑读取的数量等于当前拷贝读取的数量加上一致性拷贝读取的数量。物理读取的数量代表了遗漏的数量。逻辑读取的数量代表了请求的数量。你还可以从 report.txt的Statistics部分中得到物理读取、数据库块读取和一致性读取。使用相同的公式计算缓存命中率。

假如数据库缓存命中率小于 0.90，那么增加 DB\_BLOCK\_BUFFERS，然后重新运行 utlstat.sql/utlstat.sql。增加这个参数，就像大部分 init.ora参数一样，需要关闭实例并且再启动。然而，这不会造成重大问题，例如在数据库已经建好后改变 DB\_BLOCK\_SIZE。

在Oracle中有一种方法用来测试增加更多缓冲区后的效果。为什么你要这样做，而不是简单地增加DB\_BLOCK\_BUFFERS？情况经常是你需要把更多的缓冲区增加到你的数据库缓冲区缓存中，但是没有足够的实内存支持，因此，这项技术可以判断是否应购买更多内存。要

这样做，关闭实例并设置：

```
DB_BLOCK_LRU_EXTENDED_STATISTICS = <n>
```

其中<n>是你希望增加的缓冲区的数量。

然后再次启动该实例。让你的应用正常运行一段较为合理的时间，就像你运行 utlbstat.sql/utlestat.sql的时间一样。X\$KCBRBH表包含了你需要的信息。执行如下的查询：

```
SQL> SELECT SUM(COUNT)
2> FROM X$KCBRBH
3> WHERE INDEX < <n>;
```

该查询返回了通过增加这 <n> 个缓冲区而获得的附加缓存命中数。要确定你新假设的数据库缓存命中率是多少，把 <n> 添加到原始公式中：

$$1 - (\text{physical reads} - \text{<n>}) / (\text{logical reads})$$

正如你所看到的那样，它导致了一个较高的缓存命中率。作为一个示例，假设从 report.txt 中有 physical reads=40000, consistent gets=100000 并且 db block gets=30000。那么，在没有增加任何缓冲区以前你的缓存命中率是

$$1 - (40000 / (100000 + 30000)) = 1 - (40000 / 130000) = 1 - (0.31) = 0.69$$

该值低于 0.90。假定你想要增加 10000 个缓冲区。设置你的扩展统计数字并查询 X\$KCBRBH 表，该表表明将获得 30000 个命中。新假设的缓存命中率（伴随着性能的提高）将是

$$1 - ((40000 - 30000) / (130000)) = 1 - (10000 / 130000) = 1 - (0.08) = 0.91$$

该值明显提高。当然，这个例子使用了经过设计的数字，但是它足以强调一点，即当你用完内存并需要判断是否应购买更多内存时，尤其是要支持数据库缓冲区缓存的情况下，缓存命中率是非常有用的。最困难的事情是确定要添加多少缓冲区以使你得到你所需要的附加命中数目。建议你设法增加此百分数；首先增加 10%，然后根据结果再决定是增加还是减小。有一个类似的方法用于测试除去“不必要的”缓冲区后的效果。然而，任何数据库管理员都知道很少有不必要的缓冲区，所以本书不涉及它。

你可以把特殊的系统等待事件看作用来帮助测试数据库缓存性能的准则。像较早前在调整库缓存中所显示的一样，你可以从 report.txt 的整个系统等待事件部分或者 V\$SYSTEM\_EVENT 与 V\$SESSION\_WAIT 中收集 'buffer busy waits' 和 'latch free'。还有，从 V\$SYSSTAT 或 report.txt 中检查 'free buffer inspected'。特别是，假如下面任何一项大于 0 的话，说明有 contention 问题：

缓冲区忙等待。

空闲。

受检查的空闲缓冲区。

init.ora 的参数 DB\_BLOCK\_LRU\_LATCHES 被缺省设置为一台 SMP 计算机上 CPU 数量的一半。根据以前的经验，如果该值看起来不够的话，那么把它增加到 CPU 数量的 2 倍。

**警告** 如果你没有增加 DB\_BLOCK\_LRU\_LATCHES 的依据，就不要这样做，你可以一点点地增加它——例如，从 CPU 数量的 1/2 增加到 3/4，等等。如果一个预测的工作量较重，你可以把 DB\_BLOCK\_LRU\_LATCHES 增大一点儿。

如前所述，如果你在 MRU 端缓存整个表的话，你可以减小整个表对放置在 LRU 列表的 LRU 端的数据块进行搜索这种现象。你如何做这项工作呢？要么使用有非缺省值 CACHE 子句

的CREATE TABLE，要么在引用该表的第一个查询中嵌入一个CACHE提示。把init.ora的参数CACHE\_SIZE\_THRESHOLD设置为每张表允许保存的数据块的最大数量。一般，当你预计要经常重复使用相同的表（例如查询表）时，使用这些方法。当心不要保存太多的表，这样就达不到缓存的目标。情况变化很快，尤其是如果大部分非查询的大表不能得到至少是部分缓存的机会时更是如此。

假如你运行 \$ORACLE\_HOME/rdbms<version>/admin/catparr.sql脚本，它便创建V\$CACHE视图以及仅与 Oracle 并行服务器（OPS）有密切关系的其他视图。然而，V\$CACHE视图非常有用，因为它按对象提供了一个数据块到数据文件的映射。因此，你可以确定哪一个对象（例如表）在当前数据库缓冲区缓存中有数据块。它可以告诉你，你保存查询表的努力是否已起作用。该视图像许多V\$视图一样不是动态的，所以不管你是创建了新的对象还是改变了你当前对象的存储参数，你必须重新运行 catparr.sql脚本。

## 18.4 调整排序

排序（sort）是按某种顺序放置事物的过程，例如按字母顺序或按数字顺序，按升序或降序等等。和任何RDBMS一样，Oracle会由于各种原因而要求进行排序，这些原因中有时隐含地是因为一条SQL语句的性质，有时明显是接受用户的请求。总之，排序消耗大量的CPU时间、内存和磁盘。

首选的最优策略是尽可能地避免不必要的排序。然而，在多数情况下，当排序不能被避免时，则需要对排序进行调整以便它可以最佳地运行。排序可以完全在内存中发生，这是所希望的情形。然而，更有可能溢出到磁盘排序，特别是有大表时，不管数据库有多么好的物理设计，排序也极其消耗时间。

第二条最优策略是尽可能地在内存中排序，仅在不得已时在磁盘上排序。当然，这暗指分配足够的临时磁盘空间（事实上是TEMP表空间），并从Oracle数据文件、回滚段和重做日志的剩余部分中物理地分离该空间。

### 18.4.1 什么是触发排序

显而易见，CREATE INDEX语句需要在索引键上执行一个排序操作来建立B\*树结构。ALTER INDEX...REBUILD同样需要相同的排序。然而，你可以选择在操作系统层对数据进行排序，然后用NOSORT选项创建该索引。这通常不会给你任何东西，除非你碰巧有该数据的一个已排序拷贝，因为你仅仅用RDBMS排序换取操作系统排序，这根本称不上一个交易。其他的选择包括使用一个快速的第三方排序工具（例如SyncSort），或利用Oracle的并行查询选项（PQO）来使用SQL\*Loader并以并行非排序方式装载数据。

ORDER BY与GROUP BY通常需要排序。然而，在大多数情况下，在索引列上的ORDER BY使用已经排序的索引。要在不执行的情况下验证这一点，按照在第17章中讨论过的那样使用EXPLAIN PLAN。DISTINCT限定符必须使用一个排序方法（除非它在具有唯一索引的列上使用）来消除重复的列值。同样，UNION必须消除重复的行。然而，由定义可知，UNION ALL允许重复行的存在，所以因为它不消除重复，不需要排序。假如在两张UNION表上实施主键，就不会有任何重复，所以UNION ALL是对UNION操作的推荐替代品。INTERSECT与MINUS也需要消除重复，但消耗比UNION操作小。类似，IN或NOT IN需要排序，特别是如

果它们支持嵌套子查询的话,就更需要排序。连结操作需要对连结键上不存在索引的表进行排序。更平常的情况是在主键上连结表(已经有唯一索引),因此不必对任何表排序。下面的列表总结了能够触发排序的SQL命令或操作符:

```
CREATE INDEX, ALTER INDEX...REBUILD  
ORDER BY, GROUP BY  
DISTINCT  
UNION, INTERSECT, MINUS  
IN, NOT IN
```

#### 18.4.2 排序参数

影响排序操作的两个主要的 init.ora 参数是:

**SORT\_AREA\_RETAINED\_SIZE**: 一个内存中的排序可以使用的最大内存数。

**SORT\_AREA\_SIZE**: 外部磁盘排序操作(包括临时段的分配)可以使用的最大内存数。

假如一个排序操作需要比 **SORT\_AREA\_RETAINED\_SIZE** 还多的内存用于内存中的排序,那么它就试图在 **SORT\_AREA\_SIZE** 中执行一个外部磁盘排序,在进程中分配一个临时段。假如排序操作需要更多的内存,那么它把该排序任务分为多个排序操作( sort run )并为此分配多个临时段。服务器进程每次对一个段排序,返回已排序段的合并结果。这些内存分配不存储在SGA共享池中,但当使用MTS时例外。相反,它们是UGA的一部分。如果你正在使用MTS,它们是SGA共享池的一部分,因为UGA被重新定位在那里。

使用EXPLAIN PLAN(参见第17章),你会发现许多SQL语句在它们的执行计划中需要多个排序。当前正在执行的排序被称为活动排序( active sort )。连结排序( join sort )是一种支持连结操作的排序。任何一个活动排序需要 **SORT\_AREA\_SIZE**。任何一个连结排序需要 **SORT\_AREA\_RETAINED\_SIZE**。这些设置仅适用于专用服务器。对于PQO来说,每一个并行查询服务器需要 **SORT\_AREA\_SIZE**。然而,两组并行服务器可以同时工作。所以,对于PQO来说,设置如下值:

$\text{SORT\_AREA\_SIZE} \times 2 \times (\text{并行度})$

$\text{SORT\_AREA\_RETAINED\_SIZE} \times (\text{并行度}) \times (\text{排序数} > 2)$

对于PQO来说,最佳值是1MB。更高的值也不会产生较好的性能。一般说来,除了MTS以外,设置 **SORT\_AREA\_SIZE = SORT\_AREA\_RETAINED\_SIZE**,对于MTS需要一些特殊的考虑。

**提示** 对于MTS来说,把 **SORT\_AREA\_RETAINED\_SIZE** 设置为小于 **SORT\_AREA\_SIZE** 的数。作为一条准则,你可以如下设置:

```
SORT_AREA_RETAINED_SIZE =  
(SORT_AREA_SIZE / the number of expected concurrent sorts),  
but not less than 1/10 (SORT_AREA_SIZE).
```

当一个排序不能完全在内存中进行时,必须创建临时(排序)段。也就是说,像已经讨论过的那样,当排序操作对内存的需要超过了 **SORT\_AREA\_RETAINED\_SIZE** 的设置值时,Oracle就需要分配一个临时段并设法在 **SORT\_AREA\_SIZE** 中工作。一个真正的临时表空间(7.3版本或更高)段不能包含任何永久对象,只能由单独的排序段组成。临时表空间用

CREATE或ALTER TABLESPACE <tablespace\_name> TEMPORARY...语法创建。再有，这些临时表空间由一个段组成，该段最初由需要它的第一个排序创建。随着排序并行性和操作容量的增加，该段在区间内增长。

你如何设置 INITIAL 与 NEXT 区间参数。一个有用的指导是设置 INITIAL=NEXT=( <由数据文件或磁盘规定的最大容量 >/<预期的并行排序的数量 > )。预期的并行排序的数量可以粗略地推算为并行查询数量的两倍。当你不希望一个已测量的大区间小于数据文件的容量时，就会出现上述情况，对于表空间的一般用法（永久）来说，这通常是一个好的建议，稍后你会在第 19 章中看到。还有，因为你不要任何需要多个区间的排序，所以设置 INITIAL 和 NEXT 等于 SORT\_AREA\_SIZE 的倍数与用于该区间头的系统开销的数据块之和。同时，由于并行存取的随机性，你可以把几个排序存储在相同的区间内。因为你不想出现任何意外的事情，例如越来越大的 NEXT 区间，所以把 PCTINCREASE 设置为 0。此外，还因为并发性再次充当了一个要素。除了实际的度量尺寸技巧外，采用相等尺寸的区间是一个合理的方法。它可以同随机尺寸需求在一起很好地工作（没有一个排序需求远离平均值）。

在 SGA 中，一个被称为排序区间池（SEP）的内存结构包含组成属于临时表空间的单排序段的区域。当一个进程申请排序空间时，该池提供将要重复使用的空闲区间（空闲区间是指那些已经由一个较早前运行的进程分配和使用，目前空闲但没有被解除分配的区间），这与重复使用数据库缓存中的缓冲区的功能很像。此外，V\$SORT\_SEGMENT 包含有关用户数量、区间和使用临时排序段的数据块方面的信息。你可以使用此信息来确定效率（命中）以及正确地度量你的区间。

提示 记住要重新定义用户的缺省表空间。使用 ALTER USER 语法或企业管理器完成这项工作。TEMP 传统上是主要临时表空间并且经常用来存储永久（处理）对象。然而，可以创建你需要数量的临时表空间（例如 3 或 4 个）并根据不同的处理要求把它们分配给不同的用户组。这有助于消除用户之间和 Oracle 本身的争用。例如，保存用于 Oracle 的 TEMP，然后创建并指定 TEMP2、TEMP3 等等，使它们属于不同的用户或进程组。

Oracle 提供了绕过数据库缓冲区缓存进行排序的能力。这被叫做排序直接写入（sort direct write）。当然，你仍然需要 SORT\_AREA\_SIZE 个字节，但是每一个排序操作可以有它自己的存储区并把它们直接写入到磁盘中。缓冲区的数量用 init.ora 参数 SORT\_WRITE\_BUFFERS（2~8）来设置，缓冲区的大小由 SORT\_WRITE\_BUFFERS\_SIZE（32~64KB）设置。一个普通（串行）排序操作需要大小为（SORT\_WRITE\_BUFFERS × SORT\_WRITE\_BUFFERS\_SIZE）+ SORT\_AREA\_SIZE 的排序直接写缓冲区。

对于 PQO 来说，每一个（并行）排序需要：

$$((\text{SORT\_WRITE\_BUFFERS} \times \text{SORT\_WRITE\_BUFFERS\_SIZE}) + \text{SORT\_AREA\_SIZE}) \times 2 \\ \times (\text{并行度})$$

init.ora 参数 SORT\_DIRECT\_WRITES 确定是否使用数据库缓冲区缓存的排序操作。如果设置为 AUTO（即系统的缺省设置），如果 SORT\_AREA\_SIZE 不小于 10 × 排序直接写缓冲区，则使用排序直接写缓冲区。如果设置为 FALSE，排序写入在被写回磁盘以前在数据库缓冲区缓存中得到缓冲。这是普通的排序缓冲区写。假如设置为 TRUE，排序写就总是排序直接写。VLDB、DSS 和数据仓库通常将该参数设置为 TRUE（或至少设置为缺省值 AUTO）。

### 18.4.3 其他排序微调参数

如下是影响排序性能的其他微调参数：

`SORT_READ_FAC`

`SORT_SPACEMAP_SIZE`

`SORT_READ_FAC`是一个比率，它代表了读取一个 Oracle 数据块的时间量除以块传送率。

该参数必须小于或等于 `DB_FILE_MULTIBLOCK_READ_COUNT`。设置它的公式是：

$(\text{平均寻找时间} + \text{平均延迟时间} + \text{块传送时间}) / (\text{块传送率})$ 。

此参数考虑到操作系统和硬件（磁盘）设备在排序读取上的特性。然而，如果你使用具有混合存取时间（存取时间 = 寻找时间 + 延迟时间）的混合磁盘怎么办？采用最小公分母方法并把它设置为最慢磁盘的特性吗？

**警告** 除非你有一个非常相似的硬件（磁盘）配置以及足够的硬件经验，否则我建议你不要设置该参数。假如进行了不正确的设置，危害远大于帮助。

`SORT_SPACEMAP_SIZE`表示排序空间映射（sort space map）的字节大小，排序空间映射是多个排序运行地址中的每一个排序（每一个环境）一个映射。设置该值的公式为（总排序字节 / `SORT_AREA_SIZE`）+ 64。总排序字节（total sort byte）要求你知道被排序的列乘以行数的字节大小。你可以使用整个表大小，但这是一个过高的估计（不一定是坏事情）。

按如下语句查询 `V$SYSSTAT` 视图：

```
SQL> SELECT M.VALUE / D.VALUE "Memory Sort Ratio"
2> FROM V$SYSSTAT M, V$SYSSTAT D
3> WHERE M.NAME = 'sorts (memory)'
4> AND D.NAME = 'sorts (disk)';
```

从 `report.txt` 的 `Statistics` 部分中，用 'sort (memory)' 的总数除以 'sort (disk)' 的总数来计算 Memory Sort Ratio。

假如计算 Memory Sort Ratio 的两种方法都产生低于 0.95 的比率，你需要按照不足率来增加 `SORT_AREA_SIZE`（和 `SORT_AREA_RETAINED_SIZE`），其中不足率用 0.95 减去 Memory Sort Ratio 来计算。例如，假如你的 Memory Sort Ratio 是 0.78，那么不足率是  $0.95 - 0.78 = 0.17$ 。这表明你需要把你的排序参数至少增加 17%。

## 18.5 调整多线程服务器

本节不对多线程服务器（MTS）的体系结构进行详细介绍，而是提供一系列简明的解释与准则帮助你调整你的 MTS 配置，特别是内存性能的优化。回忆一下 MTS 把会话用户内存、用户全局区（UGA）重新定位到 SGA 共享池，该共享池被许多会话共享。通常，当你想要增加一个数据库系统的吞吐量（例如，每秒钟的事务量），尤其是在系统重负载（并行）时，你就要使用多线程服务器。换句话说，多线程服务器提供一个具有至少 200 个以上的并行用户的 OLTP 系统使用。

**警告** 假如你计划把多线程服务器用于一个负载并不很重的系统的话，使用多线程服务器所花费的系统开销会超过它所带来的好处。也就是说，对于较轻负载的系统来说，多线程服务器实际上会损害系统性能。所以建议把多线程服务器主要用于负载较重的 OLTP 系统。

提示 将MTS\_SERVERS设置为1/100（并行事务的数量），其中并行事务的数量等于并行用户的数量乘上每个用户所拥有的事务的数量。将MTS\_MAX\_SERVERS设置为1/10（MTS\_SERVERS）。逐渐增加它们的设置值并进行测试。类似地，设置 MTS\_DISPATCHERS=MTS\_SERVERS以及MTS\_MAX\_DISPATCHERS=MTS\_MAX\_SERVERS。

要对共享服务器进行监控与调整，使用如下语句：

```
SQL> SELECT BUSY / (BUSY+IDLE) "Shared Servers Busy"
2> FROM V$SHARED_SERVER;
```

```
SQL> SELECT SERVERS_HIGHWATER
2> FROM V$MTS;
```

对于调度程序：

```
SQL> SELECT BUSY / (BUSY+IDLE) "Dispatchers Busy"
2> FROM V$DISPATCHER;
```

```
SQL> SELECT SUM(WAIT)/SUM(TOTALQ) "Dispatcher Waits"
2> FROM V$QUEUE
3> WHERE TYPE = 'DISPATCHER';
```

```
SQL> SELECT COUNT(*) "Number of Dispatchers"
2> FROM V$DISPATCHER;
```

---

假如 Shared Servers Busy 大于 0.50，或者 SERVERS\_HIGHWATER 接近于或等于 MTS\_MAX\_SERVERS 的话，那么增加 MTS\_MAX\_SERVERS 的值。假如 Dispatchers Busy 大于 0.50，Dispatcher Waits 大于 0，或者 Number of Dispatchers 接近于或等于 MTS\_MAX\_DISPATCHERS 的话，那么增加 MTS\_MAX\_DISPATCHERS 的值。通常，如果你增加了一个的话，你或许也要增加另一个。

---

## 18.6 调整锁

和上一节一样，本节不会深入地讨论软件工程与 Oracle 锁机制的功能细节问题（参见第 25 章有关 Oracle 锁的背景知识）。你当然会看一些简要的背景知识与有用的调整建议。Oracle 中的锁，就和任何 RDBMS 一样，是基于内存结构的。和门一样，它由两个基本的逻辑结构组成：一个门(gate)和一个队列。假如仅有门的话，那么就有一个门。假如有一个门和一个队列以及一个排队进程的话，就有了一个锁。

调整的主要目标是消除不必要的等待和死锁，以及在关键处理期间不会用完锁。Oracle 锁的粒度是行级。也就是说，对于 INSERT、UPDATE 和 DELETE 语句，缺省情况下，锁被保持在行级。除非在代码中明确地指定，否则缺省情况下 SELECT 语句（查询）不持有锁。DDL 也持有锁，但是其性能调整能力是有限的；所以把注意力转向调整 DML 锁的性能这个主要问题上。

注意 直到事务提交或回滚，事务才释放锁。

对锁定的不良应用编码带来的直接后果是锁会引起不必要的等待。以不必要的高粒度对锁进行编码会引起不必要的锁等待——例如，当你仅需要一行时却锁定整张表。假如这个代码包含了不必要的长事务，会引起过度的锁等待。这方面的例子是在整个事务期内持久地保留客户/服务器连接。这种类型的编码经常可以被分为两个或多个较短持续时间的事务。最后，假如代码没有以正常方式被提交（或回滚），锁仍然被不适当地保持着，也会引起不必要的等

待。当编写代码时，应保证下列各项：

- 仅在必要时持有锁。
- 在尽可能低的层次上持有锁。
- 保持事务尽可能短小。
- 尽可能频繁地提交（或回滚）。
- 可能的话，取消客户/服务器事务。

这是调整锁与调整应用相互重叠的地方。当编写代码时，保证仅在必要时持有锁，在尽可能低的层次上持有锁，保持事务尽可能地短小，尽可能频繁地提交（或回滚）并且在可能的情况下取消客户/服务器事务。

提示 在你的所有外键上创建索引以消除不必要的父子锁定。

要监控锁定，作为SYS运行`$ORACLE_HOME/rdbms<version>/admin/catblock.sql`以创建锁定视图。然后你可以从DBA\_OBJECTS、DBA\_WAITERS、DBA\_BLOCKERS、V\$SESSION和V\$LOCK表中收集会话等待信息。你还可以运行 `$ORACLE_HOME/rdbms<version>/admin/utllockt.sql`脚本以得到这些信息中的大部分。检查 System wide wait events中的Count for Event Name='enqueue'。假如其中的任何一个信息源提示锁定比必要的高，那么就应用以前的准则。假如必要，使用ALTER SESSION KILL语法删除会话。假如发生了死锁，则一个跟踪文件被转储。检查该跟踪文件以确定哪一个会话以及什么类型的锁引起了死锁。然后消除不必要的锁定，必要时对你的处理重新排序。

假如需要增加锁的数量以减少锁等待或者因为你已经到达了缺省值的上限，你可能需要增加ENQUEUE\_RESOURCES。ENQUEUE\_RESOURCES是DML\_LOCKS、DDL\_LOCKS、其他参数和平台特性的函数。你可以ALTER TABLE DISABLE LOCKS以加速某个专用运行，但是我一般不推荐这样做。还有，你可以设置DML\_LOCKS=0来加速一个完整的实例，但是这或许产生不可预料的负面效应。很显然，在不使用重量锁的情况下，Oracle仍然能够以某种方式管理并行性，但是这种机制常发生完整性问题；因此，我不推荐它。

提示 设置 $DML\_LOCKS = (\text{并行用户的最大数}) \times (\text{表的数量})$ 。你还可以明确地设置ENQUEUE\_RESOURCES，但是这样会清除DML\_LOCKS。例如，假如你有U个用户和T个表，那么应设置 $DML\_LOCKS = (U \times T)$ ，还可以加上一些裕量，例如10%。

## 18.7 再看操作系统集成

不必讨论太具体的操作系统平台，只需简要地看一些与Oracle中的调整内存有关的常见操作系统内存问题。平台的细节可以在附录A和附录B中发现。主要的操作系统集成内存问题如下：

- 共享内存。
- 信号量。
- 进程间通信。
- 虚拟内存。
- 内存文件系统。

共享内存（shared memory）是由所有RDBMS厂商，尤其是在UNIX系统上的RDBMS厂

商虚拟使用的一种机制。共享内存支持多线程以及进程之间的内存共享。Oracle把后一种方法用于SGA，在该实例的所有会话之间共享内存。多线程服务器还依赖此资源以便在进程间模拟多线程。

**警告** Oracle SGA应很好地适应由操作系统给定的共享内存。否则，会有不必要的内存分页与交换，这可能会削弱一个系统。

信号量（semaphore）是真正的锁定机制。Oracle把它们用作诸如DML锁这样的排队资源的基础。再有，它们由一个（全局内存）门和队列以及一组队列操作组成。Oracle的锁定操作映射到操作系统的低级信号量操作。

进程间通信（interprocess communication）是指允许进程之间互相通信的本地操作系统通信协议。它们能够用网络套接字、流、命名管道或其他机制来实现。因为Oracle不是完全多线程的，所以它大量依赖于操作系统的进程间操作。Oracle的会话间通信被映射到这些低级的操作。当使用SQL\*Net时，IPC协议参考并使用缺省的操作系统进程间通信方法。

虚拟内存（virtual memory）是一种特殊类型的高速缓存。它是对磁盘实内存的扩展。正如同所有高速缓存一样，虚拟内存的命中率可以被计算出来。虚拟内存必须很充足，以满足操作系统本身和包括Oracle的所有的应用对总内存空间的需要。当实内存被分页或交换时，它被送给虚拟内存（磁盘）。支持虚拟内存的磁盘叫做备份存储（backing store）。还有，不希望Oracle SGA在那里映射或交换，也不希望任何主要的Oracle后台进程被交换出来。

内存文件系统（memory file system）是完整保存在实内存或虚拟内存中的文件系统。有时它们也叫做RAM盘（在DOS中）或临时文件系统（在UNIX中）。总之，它们的工作性能优于普通的磁盘文件系统一定的数量级。因为它们至少可以部分保存在内存中并利用虚拟内存高速缓存技术，所以大部分操作（至少一部分操作）来自内存或送到内存。Oracle有时把它们作为外来的性能调整解决方法使用。例如，你可以在一个操作系统的内存文件系统中创建一个Oracle临时表空间（或永久表空间）。这些表空间也可以用于存储查询表等用途。