

第19章 调整输入/输出

本章要点：

调整表空间与数据文件

调整块和区域

调整回滚段

调整重做日志

Oracle 8的新输入/输出特性

通常，调整输入/输出被认为是对物理设计的扩展与改进。当你刚开始做物理设计时，需要完成大量的定量估计工作。如果你不调整输入/输出性能就无法接受的话，那么你就需要在应用程序运行到峰值时刻对它进行实际测量。在 Oracle中调整输入/输出主要包括调整构成数据库的段（表和索引）的基本物理结构。它们包括表空间和数据文件。表空间由区间组成，进而由块组成。数据文件是支持 Oracle物理结构的操作系统（OS）实体。它们和其他的 Oracle结构在第5章中已经讨论过了，因此本章不会过深地涉及它们的定义和函数，只是顺便再强调一些有关的概念。

输入/输出意指读和写。在数据库术语中，特别是对 DML而言，SELECT操作是读，而 INSERT、UPDATE和DELETE操作是写。DDL（CREATE、ALTER或者是DROP）总是写操作。因此，从 Oracle结构中读和向 Oracle结构中写都被认为是输入/输出问题。例如，发出一条SELECT语句将产生从一个或多个索引/表中读的操作。它也产生一些最小的重做日志信息。发出INSERT、UPDATE或者DELETE就产生对一个或多个索引/表、回滚数据和重做日志信息的读和写操作。这就得出一个非常微妙的一般结论：读就仅仅是读，而写却是读加写。如何把已经读到数据缓冲区的信息写到块中呢？只能使用 SQL* Loader的直接路径功能。

因此，在 Oracle中考虑调整输入/输出时，不仅要考虑调整表空间、区域、块和数据文件，同时还要考虑调整回滚段和重做日志（因为用户 DML会产生各种各样的输入/输出）。因此，下面这部分涵盖了各种输入/输出：表空间和数据文件、区域和块、回滚段和重做日志。同前面的调整内存和应用程序的章节类似，你会遇到调整输入/输出与调整内存以及调整输入/输出与调整应用程序相交叉的情况。

记住，任何输入/输出操作都需要在采取进一步的行动前把 Oracle数据块读到数据库高速缓存中。这就是一个调整内存和调整输入/输出相交叉的例子。考虑一个不同的观点：假设你有一个OLTP应用软件，它仅从许多表中的几个表中大量地读和写，而那些表要被放到单独的表空间，甚至是单独的磁盘中。这就是一个调整应用程序和调整输入/输出相交叉的例子。用高层次的眼光看调整表空间和数据文件是开始学习的好起点。

19.1 调整表空间与数据文件

正如你在第5章中所了解的，表空间是 Oracle用于物理存储的结构。表空间存储段的集合：表和索引。一个表空间在操作系统级映射到一个或多个数据文件上去。你在第3章和第16章中

学习了应用程序类型的概念以及它如何影响到你的物理设计的问题。复习一下，为了有一个适当的物理布局，表空间（和它们的数据文件）应尽可能地分散在不同的磁盘上。把表空间分散在不同的盘上能够消除或者至少能够减少磁盘争用。

磁盘争用在多用户或程序试图同时访问同一张磁盘时发生。例如，你有两个必须经常连结在一起的表，例如 DEPARTMENTS 和 EMPLOYEES 表，它们的表空间通常要被分散在两张不同的磁盘上，因为要存取相同磁盘上的表或者索引，所以会导致对同一资源的争用。理想情况就是这四个段放在四张不同磁盘上的四个表空间里。另一种方法是和聚簇那样，把这些段放在相同的磁盘上。你可以研究 Oracle 版的聚簇技术，在第 10 章中被称为外来的解决方案。

19.1.1 分区表空间

和你在第 3 章和第 10 章学习的一样，希望使你的 Oracle 物理布局为如下形式：

系统（SYSTEM）放在一张单独的盘上。

TEMP 至少放在一张单独的盘上。

数据 1（DATA1）到数据 N（DATAN）放在 N 张单独的盘上。

索引 1（INDEX1）到索引 N（INDEXN）放在 N 张单独的盘上。

ROLLBACK 至少放在一张单独的盘上。

redo log 1 到 redo log N 放在 N 张单独的盘上。

记住，在最初的创建中，用户的缺省表空间和缺省的临时表空间指向的是 SYSTEM！如果它们没有被正确创建的话可以通过执行下面的语句加以改变：

```
SQL> ALTER USER <user>
      2> DEFAULT TABLESPACE <tablespace>
      3> TEMPORARY TABLESPACE TEMP;
```

为了要确定 DATA（或者 INDEX）表空间能否共存，你需要按表的活动性级别对它们进行分类，另外还要测定应用程序的类型。

例如，假定你有一个 DSS 应用程序，该应用程序在通常的操作中所有的表都是只读的（它们被大批装载时除外）。那么这是否就意味着你可以把所有的表放在一张盘上呢？不一定！为便于说明，考虑一下从 40 个表中读出 10 个表的情况。这 10 个表中的 7 个是可以并行存取的，其中的 4 个几乎总是连结在一起使用。理想情况是你至少要用 9 张盘使这些数据独立。

我是如何得出这个数字的？需要 7 张单独的盘，因为 7 个表是并行存取的，包括那 4 个经常连接在一起的表。你至少再需要一张盘用于剩下的 3（10-7）个不能并行存取的表，同时你还至少再需要一张盘用于剩下的 30（40-10）个表（如果一张盘能放下所有那 30 个表的话）。总之，跟应用程序分类一样，这是活动性的分类：把表按存取的频率分类。如果一个表非常活跃，那么就说这个表是热的（hot）。同样一个表中的活动的列也被称为是热的（hot）。一个温暖的（warm）表或列是指相对于 hot 成为来讲不太活动的表或列。那么，寒冷（cold）的表就是指那些很少进行存取的。另外，还有低 / 中 / 高活动性命名法。总之，可以按下述方法给表分类：

H：高活动性，或者热的。

M：中等活动性，或者温暖的。

L：低活动性，或者寒冷的。

尽管只是一个简单的分类，但它对物理设计和性能调整有很大的帮助。例如，高 DML 活动性表会形成高碎片，如果把它们保存在本身单独的表空间或磁盘上，那么就可以避免较低活动性表的不必要碎片。当分类工作结束后，遵循下面简单的方针给表空间分区以及给表 / 索引定位：

把每个热表/索引放在它们本身单独的表空间/磁盘上。

把每个温暖表/索引放在它们本身单独的表空间/磁盘上。

把多组寒冷表/索引放在它们本身单独的表空间/磁盘上。

把已连结的表/索引放在它们本身单独的表空间/磁盘上。

保持数据和索引表空间独立。

把最热的表空间放在最快的盘上，最寒冷的放在最慢的盘上。

由于磁盘空间有限，如果需要的话，可以把温暖的表 / 索引和寒冷的表 / 索引放在相同的表空间/磁盘上，但这就需要考虑并行存取的问题（例如，可能不想把一个温暖的交互式存取的表同一个寒冷的、处理时间长达几个小时的批量存取的表放在同一个表空间上）。

如果不并行存取，把“相似”的表/索引放在相同的表空间/磁盘上。

前面方针的扩展：由于磁盘空间有限，如果需要的话，把热的、温暖的和寒冷的表位置交叉存放。换句话说，就是要了解你的存取模式、使用类型及所有表的并行级别。有了这些知识，如果磁盘空间紧张的话，你就可以把两个不在一起并行存取的热表放在相同的表空间 / 磁盘上，把不同表空间上的热索引和寒冷表放在一个同一张盘上，或者把所有放在六个不同的表空间中的三种类型的表和索引放在同一张盘上。建议：除非绝对需要否则不要这样做。这是对低磁盘空间上的数据库系统所做的最后的努力。

聚簇提供了一种可供选择的存储方法用于存储频繁连结的表。

19.1.2 聚簇

簇（cluster）是一种特殊类型的表空间，在这里能够物理嵌套父 - 子或者主 - 从层次关系。例如，如果一个雇员只能在一个部门工作，这就是一个从 DEPARTMENTS 表到 EMPLOYEES 表的一对多逻辑关系。EMPLOYEES 表有一个外键 DEPTID，对应着 DEPARTMENTS 表的主键 DEPTID。你可以像这样建立一个簇：

```
SQL> CREATE CLUSTER DEPTS_EMPS (DEPTID NUMBER (9))
2> SIZE 256
3> TABLESPACE DATAn
4> STORAGE (...);

SQL> CREATE TABLE DEPARTMENTS
2> (DEPTID NUMBER(9) PRIMARY KEY, ...)
3> CLUSTER DEPTS_EMPS (DEPTID);

SQL> CREATE TABLE EMPLOYEES
2> (EMPID NUMBER (9) PRIMARY KEY, ...
3> DEPTID NUMBER (9) REFERENCES DEPARTMENTS)
4> CLUSTER DEPTS_EMPS (DEPTID);
```

可选择的 SIZE 变量指定簇键（这个例子中是 DEPTID）及所有与它相关的行预计要用的字

节数。表 19-1 是 DEPTS_EMPS 簇如何被物理地组织的一个文本表示法。

表 19-1 DEPTS_EMPS 簇

DEPARTMENTS表（聚簇前）：			
DEPTID	DEPTNAME	etc.	
1	PERSONNEL		
2	ACCOUNTING		
EMPLOYEES表（聚簇前）：			
EMPID	EMPNAME	DEPTID	etc.
1	William Day	1	
2	James Hutch	1	
3	Ely Jones	1	
4	Peter Page	2	
5	Tom Edwards	2	
聚簇后的DEPARTMENTS和EMPLOYEES表：			
DEPTID	DEPTNAME	etc.	
1	EMPID	EMPNAME	etc.
	PERSONNEL		
	1	William Day	
	2	James Hutch	
	3	Ely Jones	
2	ACCOUNTING		
	4	Peter Page	
	5	Tom Edwards	

“ etc. ” 是 DEPARTMENTS 表和 EMPLOYEES 表中的其他非键列。你可以看到，EMPLOYEES 表被物理嵌套在 DEPARTMENTS 表中，这样无论什么时候这两个表都会通过 DEPTID 的连结一同存取，数据已经用那种方式组织好了，并且比普通情况下没有聚簇的两个表更易使用。在很多情况中，特别是在那些包括实际的范围查找（限制的或不受限制的）的情况下使用常规索引（B 树结构）。但是，如果你的应用程序查询几乎总是点查询（精确的匹配、等式的比较）的话，你也许应当使用哈希索引。

哈希索引就是把列值作为输入，然后使用一个专用的内部哈希函数计算该行的物理地址（实际是 ROWID）作为输出。如果列是 NUMBER 数据类型，它均匀分布而且不是组合的，那么你就可以使用它而不是内部哈希函数来创建哈希索引。例如，要用哈希索引重新创建 DEPTS-EMPS 簇，句法如下所示：

```
SQL> CREATE CLUSTER DEPTS_EMPS (DEPTID NUMBER (9))
2> SIZE 256
3> HASH IS DEPTID HASHKEYS 29
3> TABLESPACE DATAn
4> STORAGE (...);
```

HASH IS 选项告诉 Oracle 用 DEPTID 列代替内部哈希函数，HASHKEYS 参数说明要创建多少哈希存储段来保存哈希簇键索引（DEPTID）的输出值。这个参数要与不同的簇键值的数量相等，该值调整为下一个最大的质数。在这个例子中，假设你有 25 个部门（DEPTID），应选择下一个最大的质数 29。要了解更多的关于如何管理哈希簇和索引以及测定它们大小的内

容，请参考Oracle服务器管理者指南。

19.1.3 监控

你已经看到了准则和聚簇能帮助减少对热表空间的争用，但是怎么才能区分出哪个表空间是热的呢？你在第3章中学到，如果从早期建模说明书中得出好的定量估价，也就是好的事务分析数字，那么你就会有一个开头并且能用那些数字指导你最初的物理设计。

比如，如果20个表中的3个平均每秒产生150个事务（tps），而剩下的表产生的事务数每秒还不到30个，那么你就可以说那3个表相对于其他表是热表，因此应该和剩下的表以及它们的索引隔离出来。但在布置阶段之后，你需要监控那些实际的、低级别输入/输出数字，例如每秒物理读写的次数。

用普通的两个Oracle备用工具帮助做监控：V\$动态性能视图和utlbstat.sql/utlestat.sql适当地运行得到的report.txt。这些工具在第18章中解释了。你也可以用OEM/PP检查文件的输入/输出。你需要检查V\$DATAFILE和V\$FILESTAT视图：

```
SQL> SELECT NAME, PHYSRDS, PHYSWRTS  
2> FROM V$DATAFILE DF, V$FILESTAT FS  
3> WHERE DF.FILE# = FS.FILE#;
```

还需要从report.txt的输入/输出部分检查PHYS_READS和PHYS_WRITES。物理读和物理写的总数就是那个文件或表空间输入/输出的总数。考虑一下每张磁盘中所有表空间的文件总数。如果用V\$视图的方法，选择信息两次，组成你的统计集合：在你的应用达到最大容量后进行一次，在应用的负载刚要减少前再进行一次。这就模仿了 utlbstat.sql/utlestat.sql的方法。使用任何一种方法，按磁盘统计开始和结束输入/输出总数。从结束时的输入/输出中减去开始时的输入/输出。从统计-集合运行的结束时间减去开始时间。把时间数字转换成秒。实现的输入/输出除以在开始和结束之间占用的时间。

这有个例子：3号盘包含文件6和文件7，开始数字是在上午11:00有1200个物理读和400个物理写。结束数字是在上午11:20有31000个物理读、17000个物理写。这等同于：

```
((31000-1200) + (17000-400)) / (11:20-11:00)=  
(29800+16600) / (20m)=  
(46400) / (1200s)=  
38.67 I/O数/s
```

这表明这张盘上每秒有不到40个输入/输出，这比较合理，因为对大多数现代磁盘来说，这被普遍认为是一个饱和点。然而，如果所有其他的磁盘每秒的输入/输出都大大地低于40，那么仍然希望卸载这张盘以平衡输入/输出。不仅希望减少孤立盘的数量，而且还希望尽可能地平衡输入/输出。实际上，这个原则应延伸到所有的盘。

准则：如果任何给定的盘每秒的输入/输出接近或超过40，你可能需要把一些负载卸载到其他盘上去。最好的情况是移动表，比较坏的情况是移动表空间和它们相应的数据文件。之后，你需要考虑交叉存取、聚簇、数据条，或者如果需要的话买更多的盘。你的目标是平均所有盘的每秒输入/输出以便它们都是40或者更少。如果给定你的应用程序的最大负载需要，那么尽可能地使所有磁盘的每秒输入/输出数相等以便所有磁盘负载相同。

19.2 调整块与区间

Oracle块由区间进行组织，区间包含表空间。它们是表空间存储的物理基础。因此，对它

们的存取和增长管理的效率越高，性能也就越好。

19.2.1 使用预分配值

从第16章中，你学到了动态分配会导致太多的额外开销，并损害输入/输出性能。静态的预分配几乎在所有情况中都是首选的。你可以静态地预分配一个段（表或索引）或者一个表空间。通常，你选择一种方法或者另一种方法。如果你预分配表空间，你就应该有一个关于表大小以及它们如何映射到区间上的好设想。如果你把表空间设为缺省值，之后预分配表，那么你在把不同区间的表混合在同一个表空间时就要有一定的灵活性。但是，记住你不要混合太多不同区间大小的表（通常情况下不超过三个），因为这实际上产生了碎片，而这是所不希望的。

不必回顾第5章和第21章中给出的所有的存储术语和参数，只需简要考虑一下同一个表空间存储两个表的两种不同方法。假设你有两个表，每个表最大需要 100MB 空间。创建一个 256MB 的表空间以为一些额外的增长留出空间，该表空间只有一个数据文件。第一种方法——预分配表空间，使用如下的语句：

```
SQL> CREATE TABLESPACE TS1
2> DATAFILE '/data1/file1.dat' SIZE 256M
3> DEFAULT STORAGE (INITIAL 100M NEXT 100M
4> MINEXTENTS1);

SQL> CREATE TABLE T1 (a number(9), ..., z number(9))
2> TABLESPACE TS1;

SQL> CREATE TABLE T2 (a number(9), ..., z number(9))
2> TABLESPACE TS1;
```

第二种方法——预分配表，使用如下语句：

```
SQL> CREATE TABLESPACE TS1
2> DATAFILE '/data1/file1.dat' SIZE 256M;

SQL> CREATE TABLE T1 (a number(9), ..., z number(9))
2> TABLESPACE TS1
3> STORAGE (INITIAL 100M NEXT 10M
4> MINEXTENTS 1);

SQL> CREATE TABLE T2 (a number(9), ..., z number(9))
2> TABLESPACE TS1
3> STORAGE (INITIAL 100M NEXT 10M
4> MINEXTENTS 1);
```

预分配表不仅在增长方面上，而且在与这个表空间相关的性能方面都会提供好的粒状控制。如果有一个表在你只分配 10MB 的时候只是偶尔扩展，那么为什么要分配 100MB 呢？通常，预分配的存储单元越出色，性能就越好，这其中的许多原因在第 10 章中已经讨论过了。

19.2.2 使用 Oracle 条

你在第3章关于物理设计与 RAID 的讨论以前就已经遇到了条。这一节告诉你如何做手工的条，手工条也称为 Oracle 条。

Oracle 条本质上就是区间预分配的一种形式，每个区间处理（几乎）所有与之相对应的数据文件，这些数据文件都被方便地存放在一张单独的磁盘上。当然，缺点是你应当知道长期的最大增长是多少，或者你至少应该知道中期的最大大小是多少。然而，这有助于并行执行许多高活动性表空间的输入/输出操作。假设你有一个活动性很高的表要加上条。它的最大大

小不到600MB，你有3张磁盘可用于给它加条。语法类似下面这样：

```
SQL> CREATE TABLESPACE TS1
2> DATAFILE '/data1/file1.dat' SIZE 200M,
3> DATAFILE '/data2/file2.dat' SIZE 200M,
4> DATAFILE '/data3/file3.dat' SIZE 200M;

SQL> CREATE TABLE T1 (a varchar2(25), ..., z varchar2(25))
2> TABLESPACE TS1
3> STORAGE (INITIAL 198M NEXT 198M
4> MINEXTENTS 3 PCTINCREASE 0);
```

为了替换缺省设置值（50），把PCTINCREASE设置为0是必要的。现在你已经通过人工预分配必要的（最大）区间大小的方法，在3张不同的盘上存储了一张表，该必要的区间大小略微小于（通常为1%）相应数据文件的长度。这个表已经在盘上被有效地加上条了。条单元是一个大小为198MB的Oracle区间。同RAID条相比，这是一个非常大的条单元，因为RAID条单元通常以KB的倍数或者更小的单位来度量。尽管如此，它仍然有效，并且不需要附加硬件或软件。另一方面，如果你已经错误地度量了表的大小，或者特别是如果表的增长-收缩行为反复无常的话，那么对这种方法的维护很快就会变得很难。不管别的，如果仍然需要条的话，那么这时你就希望把RAID作为Oracle条的替代品。

像我们在第3章和第4章中讨论过的一样，对于支持表或回滚表空间的数据文件来说，最好选择RAID 3或RAID 5。RAID 0对索引是个好选择。RAID 1对重做日志是个好选择。就象刚提到的，RAID可以用来代替Oracle条，但它也可以和Oracle条一起使用。例如，在前面的通过数据文件给表加条的例子中，每一个数据文件都有可能在单个的RAID卷（磁盘集）中加上条，而不是简单地存储在一个单独的（非RAID）磁盘中。这种方法能够提供真正的性能收益，但是需要数据库管理员综合持久地管理，因此要聪明地选择你的策略。记住，尽管RAID 1, 3和5能够容忍单个磁盘的损失，但RAID不能替代一个好的备份系统。

19.2.3 避免碎片

调整的一个主要目标是避免不必要的碎片。一种方法是通过估计或测量以及正确的区间预分配进行正确地估价。同时还要避免不同的活动性级别、大小以及已知的区间需求的表在相同表空间中混合。当一个表空间中的区间被分配和解除分配时会引起区间碎片。通过自由空间碎片或者简单的表碎片两种途径形成区间碎片。自由区间是指那些从来没被分配过或者在段被删除后释放的区间。当一些单独的自由区间块遍布整个表空间时，这就被称为瑞士干酪（Swiss cheese）碎片或者冒泡（bubbling）。当在表空间的运行中存在一些连续的自由区间块时，这就是通常所说的蜂窝（honeycomb）碎片。在任何一种情况下，都有自由空间碎片。当一个表动态地伸长超过它原来创建的区域时，这就被称作表碎片。

尽管碎片通常主要被认为是空间问题，但做在一个数据库系统中的所有事情一样，它会对性能有影响。无论直接或非直接地，区间碎片会引起动态扩展。从性能来看，这种扩展是你所不希望发生的。如果在一个表空间的生命期中，段创建和撤消引起了瑞士干酪的效果，就会由于强行从多个区间中输入/输出而损伤性能。一般来说，这通常也不是特别糟。然而，在最坏的情况中，它会严重降低系统的性能。例如，假设你的PQO是基于准确的数据排列而设置，用来加速对一个大的热表的全表（连续的）搜索。过段时间，那个表就分裂成区间碎片。因为PQO结构原来是基于负载-平衡区间而设置的，所以它变得越来越慢。另外，原来对

连续磁盘单元的一组顺序读已经变成了对不同的（随机的）磁盘单元的群体读，这需要更高的活动性！

用下面的方法帮助确定你的表碎片的程度：

```
SQL> SELECT SEGMENT_NAME, EXTENTS
2> FROM DBA_EXTENTS
3> WHERE EXTENTS > 4
4> ORDER BY EXTENTS;
```

如果返回了任何段，那么就认为它们就太零碎了（除非你已经计划把这些区间作为 Oracle 条的一部分或者 VLDB 表的一部分）。要校正这个情况，做下面两件事中的一项：

创建第二个带新存储参数的表，从第一张表中选择。使引用的约束条件无效。删除第一张表。把第二张表重新命名为第一张表的名字。重新使引用的约束条件有效。重新建立所有索引和授予必要的特权。

带压缩选项导出表。使引用的约束条件无效。删除表。把导出的导入回来。重新建立所有索引和授予必要的特权。

用下面的方法帮助确定你的自由空间碎片的程度：

```
SQL> SELECT TABLESPACE_NAME, COUNT(TABLESPACE_NAME)
2> FROM DBA_FREE_SPACE
3> ORDER BY TABLESPACE_NAME
4> GROUP BY TABLESPACE_NAME
5> HAVING COUNT(TABLESPACE_NAME) > 10;
```

如果返回了任何表空间，就认为这些空间太零碎了。发出 ALTER TABLESPACE <表空间名> COALESCE 语句（对于 Oracle 7.3 和更高的版本）帮助校正它。然后重新运行查询。如果有些表空间仍然被返回并且碎片比较高（大于 5 倍的段数）的话，如果行得通，你也许希望对每个表按前面的准则做。删除所有的表。删除表空间。删除数据文件。重新创建表空间。重新创建所有的表。重新运行查询来进行一个用来与以后的增长做比较的基准测试。

提示 为了使 SMON 能够与一个表空间的自由空间结合，无论是自动的还是通过 ALTER 命令，表空间必须使它的 PCTINCREASE 大于 0。然而，除非这个表忽略 PCTINCREASE 的设置，否则，这会导致表空间内表的不希望的几何增长。建议：对表空间把 PCTINCREASE 设置成 1，对表设置成 0。

块碎片能够通过转移行或者通过链接行发生。转移行是指那些已经在一个块中被修改了，但是长度增加了以至于超过了原来块中剩余的自由空间的行。它们被重新分配了另一个块（不在自由表中），指针在原来的块中维护。链接行是那些被插入或者修改了，而长度不适合任何块的自由空间的行。它们用指针在块间被拆分。链接行可以通过适当的度量和设定 DB_BLOCK_SIZE 来避免。转移行可以通过启用足够的修改空间（ $100 - (PCTFREE + PCTSUED)$ ）来避免。为了确定链接行或转移行的程度，运行下面的语句：

```
SQL> ANALYZE TABLE T1 COMPUTE STATISTICS;
SQL> SELECT TABLE_NAME, CHAIN_CNT
2> FROM DBA_TABLES
3> WHERE CHAIN_CNT > 0
4> ORDER BY CHAIN_CNT;
```

上述语句给出一个有链接行或转移行的表的排序清单。为了帮助补救转移行的情况，你

可以删除并重建表。如果表非常大，你可以运行 `utlchain.sql`来建立 `CHAINED_ROWS`表。之后运行下面的语句：

```
SQL> ANALYZE TABLE T1 LIST CHAINED ROWS;
```

转移行的 `ROWID`存储在 `CHAINED_ROWS`中。你可以用它从你的表中选择转移行并把它它们放到一个临时表中。之后从原来的表中删除它们，再从临时表中把它们插回到原来的表中。

关于碎片性能问题的有关话题包括高水位标志、自由表和表 /索引的重新组织。高水位标志（`HWM`）是到现在为止那个段中已使用过的块的数目。删除操作不会降低 `HWM`。在全表扫描期间，`Oracle`必须读低于高水位标志的未被使用的块。尽管下面的方法没有为当前的段复位高水位标志，你仍然可以用它为其他段回收废弃的空间：

```
SQL> ALTER TABLE T1 DEALLOCATE UNUSED;
```

另外，`TRUNCATE TABLE <表名>`可以为表和所有的索引复位 `HWM`，这与 `DELETE FROM <表名>`形成对比。

自由列表（`freelist`）连接在自由块清单上，每个段中包含一个或更多的清单。在 `V$WAITSTAT`中或者使用 `report.txt`的 `System wide wait events`部分检查自由表的等待事件。如果任何自由表中的事件比0或者比你的基准数字大很多，就要考虑往热表中添加自由列表或者自由列表组。它们必须被重新建立。

如果在低于 `HWM`的块中平均有很多自由空间，那么就要考虑重新建立你的表。在分析表并计算或者估计统计数字后，检查下面的语句：

```
SQL> SELECT TABLE_NAME, AVG_SPACE  
2> FROM DBA_TABLES  
3> WHERE AVG_SPACE > (.10 * BLOCKS)  
3> ORDER BY AVG_SPACE;
```

换句话说，如果你的表在低于 `HWM`的块中有超过10%的自由空间，那么用导入/导出或者临时表的选择/重命名方法删除并重建该表。

频繁修改的表的索引应该能承受检查。它们经常需要重新创建。一条通用准则是如果索引的大小接近实际表大小的 $1/3$ ，那么就删除并重建索引。另一条通用准则是如果索引的级别数超过3，那么该索引可能就太高了。分析你的索引并检查 `INDEX_STATS`表来搜集信息。如果需要的话，删除并重建想要的索引或者使用 `ALTER INDEX <索引名>REBUILD`语句。

在块级别中 `INITRANS`是个影响输入/输出性能的微调参数。 `INITRANS`设置每个块中初始事务槽的数目。它应代表并行用户或存取块的程序的数目。这在表级设置。如果你的应用程序有很高的并行需求，把 `INITRANS`设置得相对高一些以避免过多地动态创建事务槽。这个字节系统开销很低，而性能很好。

19.3 调整回滚段

回滚段是或多或少的随机输入/输出单元。它们被并行地（由 `DBWR`）写入，在数据库缓冲区缓存中得到缓冲，并保存在专门的表空间中。回滚段会在它们之间，以及它们与其他数据库输入/输出单元（例如数据表空间）之间产生争用。它们提供取消不提交事务的影响的能力。因此，它们通常被称为取消日志，或者叫做通用事务日志的取消部分。取消数据意指在内存的数据库缓冲区缓存中，或是在磁盘上的回滚段中一致性读的 `Oracle`块。它们用于事务回滚、实例恢复以及共享的读能力。用循环的方式把它们分配给事务。

性能调整的主要目的是为了减少争用。换句话说，你希望等待回滚段的事务数最少，特别是在许多 OLTP 系统所具有的高并行环境中。回滚段的争用可以通过创建足够数量的适当大小的段来很好地避免。回滚段把事务表放在它们的头部中。争用相同回滚段的并行事务在争用事务表本身时被显示出来。你如何能发现这种争用呢？用下面的方法：

```
SQL> SELECT CLASS, COUNT
2> FROM V$WAITSTAT
3> WHERE CLASS LIKE '%undo%'
3> AND COUNT > 0;
```

你也可以使用这种方法：

```
SQL> SELECT USN, WAITS
2> FROM V$ROLLSTAT
3> WHERE WAITS > 0;
```

你也可以从 report.txt 的回滚部分检查 UNDO_SEGMENT 中是否 TRANS_TBL_WAITS>0。你也可以为 '%undo%' 事件检查 report.txt 的 System wide wait events 部分，或者从视图 V\$SYSTEM_EVENT 中选择这些事件。还有，如果你遇到了 ORA-01555 错误：“快照太旧”，表明你已经把回滚用完了。你或者需要更多的段、更大的段或者两者都要。

如果 ORA-01555 错误频繁地出现或者事务表等待事件比 0 大得多，这就表明争用。特别是，对非常大的数据库或者非常高的并行查询数据库来说（例如某些 DSS 系统），如果来自 V\$ROLLSTAT 或者 TRANS_TBL_WAITS/ TRANS_TBL_GETS 二者中任何一个的 WAITS/GETS 的比值大于 1%（0.01），就有可能存在回滚争用的问题。分配回滚段的数目等于并行用户和程序的数目除以 4，最大值为 50。

其他的回滚段调整建议如下。对于回滚段，设置 NEXT=INITIAL。对于回滚段，PCTINCREASE 总是 0。把 MINEXTENTS 设置为大于等于 20，这样 INITIAL*MINEXTENTS 大约比表空间（数据文件）大小低一个百分点，以防止动态扩展在回滚段生命期开始时受并行增加的影响。设置 OPTIMAL=INITIAL*MINEXTENTS 以防止回滚段不必要的减少。你可以通过 V\$TRANSACTION 视图计算事务产生的取消数目。在最高负载时观察 SUM(USED_UBLK) 和 MAX(USED_UBLK)，确定该时间间隔中的取消总数和最大值。几次采样后设置 INITIAL 为大于等于 MAX(USED_UBLK)。为了进行微调，适当地设置 init.ora 参数 PROCESSES、TRANSACTIONS 和 TRANSACTIONS_PER_ROLLBACK_SEGMENT。有关这些参数更详细的内容参考 Oracle 的服务器管理器指南。但是，这些参数对分配适当数目和大小的回滚段只有很小的影响。

小事务（通常在 OLTP 系统中出现）是指那些产生少量取消信息的事务。对于这种情况，按以前的准则，用 50 个回滚段在应用程序中尽可能频繁地进行提交。大事务（通常在批系统中出现）需要单个的大回滚段。因此，你不需要许多回滚段，只要一个大的即可。为了确保一个特别长时间运行的（大的）事务能够使用特别大的回滚段，采用下面的语句：

```
SQL> SET TRANSACTION USE ROLLBACK SEGMENT <rollback_segment>;
```

如果你有许多只能有限改变的小事务，比如每次只改变一个块中的几行或者插入一行，那么就把 init.ora 参数 DISCRETE_TRANSACTIONS_ENABLED 设置为 TRUE。在用这个参数前，在 Oracle 的服务器管理器指南中了解使用事务的标准和先决条件。

19.4 调整重做日志

重做日志是连续的输入/输出部分。同时，它们一次只（被 LGWR）写到一处，被缓冲在 SGA（日志缓冲区）的一个单独部分，并作为操作系统文件存储。这样重做日志之间就没有争用了，只在重做日志和其他数据库输入/输出部分（例如数据表空间）之间有争用。与回滚段可用于额外用途相对，重做日志只用于实例恢复（前滚）。当下面的事情之一发生时 LGWR 过程写重做日志（实际上，执行一次日志刷新）：提交、DBWR数据刷新（检查点）、检查点发生超时（到达 LOG_CHECKPOINT_INTERVAL），或者日志缓冲区超过总数的 1/3。它们以循环方式分配给 LGWR，并且日志交换总是触发检查点。

性能调整的主要目的是为了使下面这些最小：

给定应用程序的检查点。

重做等待事件。

重做门争用问题。

下面是配置检查点机制的建议：

1) 把不小于 2 个组中的不小于 2 个成员放在单独的输入/输出通道（磁盘和控制器）中，每个成员有相同的大小。

2) 在 init.ora 中把 CHECKPOINT_PROCESS 设成 TRUE，以便在从 LGWR 过程到 CKPT 的过程中同步卸载文件头。

3) 把 init.ora 参数 LOG_CHECKPOINT_TIMEOUT 设成 0。

4) 把 LOG_CHECKPOINT_INTERVAL 设为在操作系统块中的一个重做日志大小与一个操作系统块的大小之和。这儿有个例子：所有重做日志成员的大小都被设为 1MB，1MB=1024KB=2048 个操作系统块（每个操作系统块 512 字节）。因此，把 LOG_CHECKPOINT_INTERVAL 设为 2048+1（即 2049），同时已经把 LOG_CHECKPOINT_TIMEOUT 设成 0，就使检查点只发生在日志进行交换时，这通常可以提高性能。

警告 非常繁重的 OLTP 环境需要许多大的重做日志来支持这种设置。换句话说，你可能需要减少检查点的活动性，从几乎每分钟一次减少到每个日志交换事件一次，也许是每 15~30 分钟一次。

警告 如果使用 ARCHIVELOG 模式，确保你的最长运行事务（即产生最多重做信息的那个事务）不会消耗掉你所有可用的重做日志成员。否则，无论何时，只要重做交换超过了归档，你的数据库将死机！

应如何监控和调整重做等待事件以及重做门争用问题？检查 report.txt 关于 'redo log space requests' 的 Statistic 部分和 report.txt 中关于 'log file space /switch' 的 System wide wait events 部分。这些事件也可以分别用 V\$SYSSTAT 和 V\$SYSTEM_EVENT 视图检查。

如果这些事件中的任何一个比 0 大得多，这就表明 LOG_BUFFER 长度被设得太小了。把它增加百分之五或者更多，让你的应用程序运行一段时间，再查询这些事件一次。重复这个过程直到这些事件达到或非常接近 0。

使用 report.txt 的 Statistic 部分中的 'redo size' Total，或者 V\$SYSSTAT 的值，确定如何适当地设置重做的大小。这个统计量是采样间隔中产生的重做信息的总数。

提示 减少在直接路径方式下使用SQL*Loader产生的不必要的重做信息的数量。如果你正在归档，还要指定UNRECOVERABLE。如果你有PQO，就要为你的并行表和索引的创建语句指定UNRECOVERABLE。

像所有的门（或者自旋锁）结构一样，重做拷贝门是基于内存的机制，这种机制提供重做LOG_BUFFER的无队列锁定。实际上只有一个重做分配门，一个进程在它写时可以独自占用它（描述为一个WAIT或者WILLING_TO_WAIT申请或状态）而其他的进程等待；进程可以释放它（描述为一个NOWAIT或IMMEDIATE申请或状态）并且在写时保持一个重做拷贝门，这使另一个过程能够拥有这个重做分配门。如何度量对这个资源的争用呢？用V\$LATCH视图并计算NAME类似'redo%'的MISSES/GETS的比率。同样，计算report.txt的两个门部分中MISSES/GETS和NOWAIT_MISSES/ NOWAIT_GETS的比率。另外，查看report.txt中的HIT_RATIO和NOWAIT_HIT_RATIO的数字，它们是刚刚计算过的MISSES/GETS比率的倒数。

注意 如果任何一个MISSES/GETS比率比1%（0.01）大得多或者两个命中率中的任何一个比99%（0.99）小很多，那么就可能有太多的重做门争用。要想减少门争用，增加init.ora参数LOG SIMULTANEOUS COPIES的值，这个参数缺省情况下设置为CPU数目与CPU数目的两倍之间的一个值。要减少锁定次数以及促进最短工作优先（SJF）分配，可以减少init.ora参数LOG SMALL ENTRY MAX SIZE的值。它表示一个门限值。如果一个过程的重做信息比它小——意思是如果它产生了相对小的重做日志项——它就可以独占地保持一个重做分配门（并迅速写信息，而且比大过程更快地释放门，从而增加并行操作）。用SHOW PARAMETER在SVRMGRL中检查它的值。每次把它减少一个百分比（5%或10%）。重复这个过程直到重做争用数可以接受为止。

19.5 Oracle 8的新输入/输出特性

除了在第17章中所讨论过的诸如唯一索引表和等分区对象等Oracle 8新索引特性以外，还有一些其他的Oracle 8的新特性为输入/输出性能的提高提供了机会，包括分区扩展表名与直接载入INSERT。

19.5.1 分区扩展表名

与等分区对象（表和索引、相同值集合的相等分区）一样，表分区为分区视图提供了一个代用品。使用PQO的分区视图（Oracle 7）的创建方法是：水平地把一个单独的长表放到许多小表中的方法人工建立一个视图把它们联合起来，在每个小表中的值集合上建立约束条件，把init.ora的PARTITION_VIEW_ENABLED参数设置为TRUE。分区表（Oracle 8）通过按选中的列分区建立。然而，与其有一个由联合的表组成的视图，还不如有一个由联合的分区组成的表。另外，你可以在一定的限定下直接引用分区：

```
SQL> SELECT * FROM T1 PARTITION (P1);
```

另外，Oracle 8提供了把以前创建的Oracle 7分区视图转换为Oracle 8分区表的能力。

19.5.2 直接载入INSERT

有了直接载入INSERT，INSERT就可以利用相同的SQL*Loader直接路径模式功能的优势，

这种方法绕过了数据库缓冲区缓存，不产生重做信息，直接写入到数据文件中。此外，在 Oracle8 中，你可以把一个表、索引或者是表空间放置在不日志模式中。这就额外加强了你的直接载入 INSERT。直接载入 INSERT 由于性能的缘故要交替使用空间，因为它们在段的 HWM 上面插入行。这样就浪费了空间，但如果一个段经常接近于满的话，这不是主要问题，不要体验增长收缩模式，（特别是）要有一个表空间给它自己。对热表来说，这种能力可以用于与已计划的定期重组联合。