Oracle 与提高性能有关的特性

本章讨论 Oracle 服务器内容，管理员可通过这些方面来提高数据库性能。尽管这些主题属于 Oracle 服务器的一部分，但测试人员可将这些方面当作提高数据库性能的手段。

我们知道，数据库索引是创建在数据之上的为提高数据访问性能的对象。在设计数据库索引时，必须谨慎并且需要了解用户访问数据的模式。在某些情况下，设计不善的索引将导致系统性能的下降，本章索引是我们讨论的重点。

这里列出了讨论需要的参数以及其使用方法。

- MAX_DSPATCHERS：这个参数指定了系统允许同时进行的调度进程的最大数量。
- MAX_SHARED_SERVERS：这个参数指定了系统允许同时进行的共享服务器进程的最大数量。如果系统中出现的人为死锁过于频繁，那么管理员应该增大这个参数的值。
- PARALLEL_ADAPTIVE_MULTI_USER：当这个参数的值为 TRUE 时，系统将启动一个能提高使用并行执行的多用户系统性能的自适应算法。这个算法将根据查询开始时的系统负载自动降低查询请求的并行度。
- PARALLEL_AUTOMATIC_ENABLED：如果将这个参数的值设置为 TRUE，那么 Oracle 将确定控制并行执行的参数的默认值。
- PARLLEL_BROADCAST_ENABLED：这个参数允许管理员提高散列连接和合并连接操作的性能，在这样的连接操作中，系统将一个大尺寸的结果集与一个小尺寸的结果集连接在一起（在合并操作中，数据的尺寸是根据字节数，而不是记录数确定的）。
- PARLLEL_EXECUTION_MESSAGE_SIZE：这个参数指定了系统并行执行时的消息的尺寸（在 Oracle 的旧版本中，这个概念是指并行查询、PDML、并行恢复和并行复制数据等）。
- PARLLEL_MAX_SERVERS：这个参数指定了实例能同时运行的并行执行进程和并行恢复进程的最大数量。随着用户需求的增长，在创建实例时为这个参数设置的值将不再能满足用户需求，所以应当增大这个参数的值。
- PARLLEL_MIN_PERCENT：系统将联合使用 PARLLEL_MAX_SERVERS、PARLLEL_MIN_SERVERS 和该参数。这个参数允许指定并行执行进程（即参数 PARLLEL_MAX_SERVERS 之值）的最小百分比。
- PARLLEL_MIN_SERVERS：这个参数指定了实例并行执行进程的最小数量。其值就是实例启动时 Oracle 创建的并行执行进程数。
- PARLLEL_THREADS_PER_CPU：这个参数指定了实例默认的并行度和并行自适应以及负载平衡算法。它指明了并行执行过程中一个 CPU 能处理的进程或线程数。
- PARTITION_VIEW_ENABLED：这个参数指定了优化器是否使用分区视图。Oracle 推荐用户使用分区表（这是在 Oracle8 之后引入的）而不是分区视图。分区视图只是为了提供 Oracle 的后向兼容性。
- REVOVERY_PARALLELISM：这个参数指定了恢复数据库系统时使用的进程数。

索引

在数据库系统中，索引是一种可选结构，其目的是提高数据访问速度。不论在物理上

还是逻辑上，Oracle 数据库的索引都是独立于与之相关的表或簇中的数据的。可利用索引来提高用户访问数据的速度或直接从索引中独立检索数据。如果对索引的配置和使用进行了优化，那么索引能大大降低数据文件的 I/O 操作和提高系统性能。

对用户或应用程序而言，索引是透明的，且不需对应用程序做任何修改。但是，如果知道数据库中存在某个索引，那么可能在编写 SQL 语句时充分利用这个索引。索引的唯一优势在于可以提高用户访问数据的速度。

在为一个表创建索引之后，Oracle 将自动维护这个索引。当用户在表中插入、更新或删除记录时，系统将自动更新与该表相关的索引。一个表可以有任意数量的索引，但一个表的索引越多，用户在该表中插入、更新或删除记录时所导致的系统开销也越大。其原因是无论何时更新表，系统都必须更新与之相关的索引。

■ 与索引有关的概念

索引是建立在表的一个或多个字段之上的。索引的作用大小取决于该字段或字段集的选择性。所谓选择性是指索引能降低数据集中的程度。如果表中与某个索引相关的字段值各不相同，那么该索引就有很好的选择性，一个选择性很差的索引的例子是基于字段值仅为 true/false 的字段创建的索引，因为表中很多记录该字段的字段值都相同。在本章后面部分，读者将明白选择性将帮助程序员创建不同类型的索引。

一个索引可能只能帮助管理员降低检索的记录数，而不能唯一地确定一条记录。例如：如果为一个表的 LastName 字段创建了一个索引，现在用户需要搜索 John Smith，那么这个索引将返回 LastName 字段值为 Smith 的所有记录，因而用户还不得不在返回的记录中搜索含 John 的记录。索引的选择性越好，就越有助于降低返回记录的数量，从而提高数据访问速度。

■ 索引类型

在 Oracle 数据库中，可使用两种主要的索引类型，这两种主要索引类型的索引还包含了几个子类型索引。这两种主要类型的索引是 B-tree(B 树)索引和 bitmap（位图）索引。B 树索引的形状象一棵树，搜索条件将遍历整棵树以找到用户请求的数据。

而位图索引将为与该索引相关的数据文件的每个不同值创建一个位图并利用这个位图检索数据。在本章后面读者将会看到在某些情况下位图索引是非常有用的。

B 树索引有几种子类型，而位图索引却没有任何子类型。一个 B 树索引可限于表的一个或多个字段。索引可以是唯一的也可以不是唯一的。

可根据应用程序所访问记录的字段值来创建索引；而应用程序开发者也应当开发能充分利用索引的应用程序。

■ B 树索引

B 树索引是最普通的索引类型。B 树索引使用属性搜索来检索该索引指定的记录。在 B 树索引中，数据被存放在一棵倒置的树上，B 树索引也因此而得名。实际上，Oracle 的索引称为平衡树索引（Balanced Tree Index），平衡树索引的最高层（搜索开始的地方）称为根节点，树的末端（存放记录 ID 的地方）称为叶节点，位于根节点和叶节点之间的节点被称为枝节点。

当创建一个索引时，系统将自动为该索引创建一个索引段。索引段包含能加快数据访问速度的信息，其原理是使用尽可能少的 I/O 操作确定被索引记录的位置。Oracle 正是通过著名的 B 树索引结构来检索数据的。

B 树索引的目的是平衡数据访问时间。如图 111 所示，B 树索引是降序比较字段值的索引。当系统向下遍历 B 树索引时，系统将用户需要查询的值与 B 树的上层节点——即所谓的枝节点——进行比较。根据比较结果，系统将用户的查询值与更多的枝节点进行比较，直到达到最底层的索引节点为止。最底层的索引节点，即根节点，包含了所有被索引的数据值

和相关记录的 ID。

对于唯一性索引，根节点的数据值中只存在一条记录的 ID。而对于非唯一性索引根节点的数据值中可能存在多条记录的 ID。对于非唯一性索引，系统首先按照索引关键字对叶节点数据值进行排序，然后再按照数据记录的 ID 进行排序。

在 B 树索引中，所有叶节点都处于同一层次。不论叶节点的数据为何值，访问数据所花的时间都大致相同。不管是使用精确匹配还是基于某个范围内的查询，B 树索引都为用户提供了快速访问数据的能力。另外，不论表的规模有多大，B 树索引都能够提供很好的性能，而且表规模的增长不会降低索引性能。

B 树索引包含了几类子索引。下面将分别介绍这些子索引。

一、唯一性索引和非唯一性索引

唯一性索引是值具有附加约束条件的，并且不可能出现重复字段值的索引。尽管管理员可能会指定这里的附加约束条件，但通常情况下将附加约束条件与表相关联比与索引本身相关联要好一些。通过自动在表的唯一键上定义索引，Oracle 强制实现了唯一性索引的附加约束条件。

非唯一性索引不会附加使索引值必须唯一的约束条件。当用户希望快速访问非唯一性记录时，这种类型的索引是非常有用的。然而，对检索特定记录而言，唯一性索引的性能要优于非唯一性索引，其原因是唯一性索引的值被分解为唯一值。对于非唯一性索引，当系统遍历到索引的最底层时，系统将返回所有满足检索条件的记录。

二、简单索引和复杂索引

索引的另一种类型是复合索引，即基于表中多个字段的索引。这些字段的值可以以一定的顺序排列，系统不要求这些字段是相邻字段。

当 SELECT 语句中有 WHERE 子句，并且 WHERE 字句参考了表中的多个字段时，复合索引是非常有用的。因为系统是按照索引中定义的字段顺序访问索引的，所以聪明的做法是按照字段的访问频率来定义字段在索引中的顺序。检索数据时参考最多的字段应排在第一位，其余字段的排序以此类推。

复合索引也非常有利于将多个非唯一性索引字段组合成一个唯一性索引字段集，可利用复合索引创建一个唯一性复合索引。

三、基于函数的索引

基于函数的索引是指系统设计者或 DBA 创建的用于访问数据的特定函数。这个函数可以是数学函数也可以是 PL/SQL 函数。只有当基于该函数的索引被用于 SELECT 语句的 WHERE 字句时，该函数才能发挥作用。

例如 A 公司打算给去年在该公司购买了更多产品的所有顾客发感谢信。那么确定应该选择哪些顾客 ID 的查询语句应该具有如下形式：

```
SELECT cust_id
  FROM customers
  WHERE this_years_sales-last_years_sales>0
```

可创建一个以 c_id 字段为变量的函数条件查询，该函数具有如下形式：

```
CREATE INDEX idx ON customers (this_years_sales-last_years_sales)
```

这个索引既降低了系统负载又提高了查询性能。

■ Oracle 位图索引的工作方式

位图索引的实现原理与 B 树索引完全不同。位图索引将为被索引字段中的每个唯一值创建一个位图。这个位图包含了关于哪些记录的字段值等于该字段值的信息。位图索引的优势之一在于管理员可联合使用多个位图索引：这样，这些索引被组合在一起。

对于具有唯一值或良好选择性的字段或字段集，B 树索引是非常适用的；而位图索引正

好相反。位图索引适用于值为 yes/no 或 true/false 这样的字段。在找到所包含的很多记录都满足特定检索条件的节点之前，系统将遍历整个 B 树索引，然后再选择这些记录。而在位图索引中，系统不需要遍历整个索引就能立即找到满足检索条件的记录。

如果建立索引的字段因为其字段值没有足够的区分度而不具备良好的选择性，那么为该字段建立一个位图索引是非常合适的。如图 4.3 所示，如果能联合使用多个字段，并且这些字段的联合位图索引能大大降低系统处理的数据量，那么为这些字段创建一个位图索引是非常适合的。

■ 选择创建索引的字段

只有正确使用了索引，才能发挥其功能。而索引的使用主要取决于建立索引的字段值。请记住：在一个表上建立的索引越多时，那么在更新、插入或删除记录时所引起得系统开销也越大。因此，有选择的建立索引是非常重要的。

下部分内容将介绍有效创建和使用索引的技巧和方法。

一、索引和降低系统处理的数据量

索引的主要作用之一就是降低系统处理的数据量。对 CPU 使用和等待完成 I/O 操作的时间上，I/O 操作引起的系统开销都是非常昂贵的。降低 I/O 操作可提高系统性能和处理能力。如果不使用索引，那么为了找到特定的数据，系统将不得不扫描表中的所有数据。

例如如下查询语句：

```
SELECT *
  FROM emp
  WHERE employee_id=10290
```

如果不使用索引，系统必须扫描整个 emp 表并检查表中每条记录的 employee_id 字段的值。如果 emp 表很大，那么这个操作可能意味着数量巨大的 I/O 读写和很长的处理时间 。

如果为 emp 表的 employee_id 字段创建了索引，那么系统将遍历该索引并找到用户所查询记录的 ID。找到记录 ID 之后，只需一条额外的 I/O 操作就能检索到用户所需数据。

用于说明这个问题的最好例子是只需查找一条记录的情况。在表的每条记录中，类似 employee_id 这样的字段的值可能在整个表中都是唯一的。这意味着查询结果值返回一条记录，这种查询的效率是非常高的。

某些情况下，索引必须返回大量数据。如下面的例子：

```
SELECT *
FROM customers
WHERE region='North'
```

这个查询语句很可能返回大量数据，因为索引操作返回了大量记录的 ID，并且系统必须独立访问这些记录的 ID，所以这种情况下不使用索引可能比使用索引的效率更高，直接进行表扫描可能效率更高。不同情况下，采用哪种查寻方法更好，很大程度上取决于表的数据量和组织形式。

对于不同的数据，在某些情况下位图索引可能非常有用，而在另外一些情况下，使用位图索引可能没有任何好处。

假设 customer 表中有一个用于指明顾客信誉好坏的字段。我们在进一步假设 99%的顾客都有良好的信誉。如果需要查询具有良好信誉的顾客，那么直接对表进行扫描可能效率更高。但是，如果需要查询的是信誉不好的顾客，那么通过索引来访问数据可能具有更高的效率，其原因是只有 1%的顾客复合查询条件。因为这个字段的选择性很低，所以在这里推荐使用位图索引。

二、索引和更新

如果对表创建了索引，那么更新、插入和删除表中的记录都将导致额外的系统开销。在系统提交这些操作之前，系统将会更新所有与该表相关的索引。这可能需要花很长时间并额外增加一定的系统开销。

三、在字段选择性很低的情况下适用索引

在某些情况下，表中的某些字段的选择性可能很低。开发人员没必要为所有表创建索引，实事上，在某些情况下索引引起的问题比解决的问题更多。在很多情况下，需要反复试验才能确定一个索引是否有助于提高系统性能。

但某些情况下，位图索引能在字段选择性不高的情况下工作的很好。一个位图索引可以和其他位图索引联合使用以降低系统检索的数据集。对于某些值为 true/false、yes/no 或其它小范围数据的字段，建立位图索引是非常合适的。请读者记住：位图索引所占用的空间是随着与该索引相关的字段的不同值的数量的增加而增加的。

■ 如何创建索引

如果决定创建一个索引，那么确定为哪些字段创建索引是非常重要的。对于不同的表，可能会选择一个或多个字段创建索引。

可使用如下方法来确定在哪些字段上创建索引：

1. 选择那些最常出现在WHERE子句中的字段。经常被访问的字段最可能受益于索引。
2. 经常用于连接表的字段是创建索引的必然候选字段。
3. 必须注意索引导致的查询语句性能的提高与更新数据时性能的降低之间的平衡。
4. 经常被修改的字段不适合创建索引，其原因是更新索引将导致系统开销。

在某些情况下，使用复合索引的效率可能比使用简单索引的效率更高。下面的一些例子说明了应当在何种情况下使用复合索引。

1. 某两个字段单独来看都不具有唯一性特性，但结合在一起却有唯一性特点,那么这种情况下复合索引将工作的很好。例如：A字段和B字段都几乎没有唯一性值，但绝大多数情况下，字段A和B的某个特定组合却具有唯一性特点。那么在检索数据时，可在WHERE字句重视AND操作符来将这两个字段连接在一起。
2. 如果SELECT语句中的所有值都位于复合索引中，那么Oracle将不会检索表，而直接从索引中返回数据。
3. 如果多个查询语句的WHERE子句中作为查询条件的字段都不相同，但返回的记录相同，那么应当考虑利用这些字段创建一个复合索引。

如果对复合索引进行了仔细设计，那么复合索引非常有助于提高系统性能。就像单字段索引一样，如果应用程序开发者在设计应用程序时就充分考虑到了系统中的复合索引，那么这些复合索引将发挥巨大作用。

在创建索引之后，开发人员应当定期利用SQL TRACE工具或EXPLAIN PLAN来察看用户查询是否充分利用了索引。很有必要花费一定精力来试验使用索引和未使用索引在效率上的差别，以判断索引所耗费资源是否物有所值。

应该删除哪些不经常使用的索引。可使用ALTER INDEX MONITORING USAGE 语句来跟踪索引的使用情况。还可以从系统表ALL_INDEXES、USER_INDEXES和DBA_INDEXES中查询用户访问索引的频率。

如果为一个不适合创建索引的字段或表创建了索引，那么这可能会导致系统能力的下降。而如果创建的索引合理，那么这将降低系统的I/O操作并加快访问速度，从而大大提高

系统性能。

仔细计划索引并利用SQL TRACE工具或EXPLAIN PLAN定期测试索引使读者能有效利用索引，从而提高系统性能。

Oracle 的并行执行特性

Oracle9i的并行执行特性就是Oracle旧版本中读者熟知的并行查询选项。因为在Oracle9i中，这个特性已内嵌在Oracle RDBMS中，而不再是一个选项，所以Oracle9i已淘汰了并行查询选项。虽然自Oracle7起这个特性已经有了很大发展，但其基本思想还是保持不变。

并行执行使Oracle的某些功能由多个服务器进程协同处理成为可能。这些功能包括查询、创建索引、加载数据和恢复数据库。所有这些功能都遵循一个共同规则：充分利用CPU资源。

为什么要实现并行执行？RDBMS的绝大多数操作都可分为以下三类：

1. 被CPU限制的操作：这类操作的速度和单CPU运行速度一样。通过并行化操作，多个CPU可并行处理系统负载，因此可以更快完成该操作。
2. 被I/O限制的操作：这类操作花了绝大部分时间等待系统完成I/O操作。当系统中同时出现多个I/O请求时，绝大多数RAID控制器将很好工作。另外，当一个线程需要等待完成I/O操作时，可充分利用CPU来处理另一线程的CPU部分。
3. 被竞争限制的操作：并行处理不能改善由资源竞争所限制的操作。

对表扫描而言，系统花在等待数据从磁盘返回的时间常常比处理数据所花的时间还长。通过并行机制，可用多个服务器进程处理查询操作，从而弥补了系统在这方面的问题。当一个进程在等待I/O操作时，CPU可执行另一个进程。如果数据库系统运行在SMP(Symmetric Multiprocessor，对称多处理器)计算机、计算机群集或MMP(Massive Parallel Processing，大规模并行处理)计算机上、那么开发人员可充分利用并行处理机制的优势。

■ 并行查询处理

并行查询处理允许多个服务器进程以并行方式处理某些Oracle语句。Oracle服务器能以并行方式处理的语句有：

1. SELECT语句。
2. 在INSERT、UPDATE和DELETE语句中的子句查询。
3. CREATE TABLE tablename as SELECT语句。
4. CREATE INDEX语句和ALTER INDEX REBUILD语句。

在大型操作，比如表扫描和排序，并行查询是非常有效的。

一、 并行查询操作

在表扫描这样的传统查询中，服务器进程以顺序方式读取数据。这个查询所耗时间的大部分都花在了等待系统完成I/O操作上。

并行查询将该查询分为几个不同的部分，每部分由不同的服务器进程进行处理。这些进

程称为查询服务器。一个被称为“查询协调者”的进程负责调度这些查询服务器。

查询协调进程负责调度查询服务器和协调从各个服务器返回给用户的查询结果。这种安排的结果导致系统中产生了多个相对较小的表扫描（这个过程对用户是透明的）操作。从用户观点来看，这个操作仅仅是一个更快的表扫描而已。

系统将一个SQL语句和一个并行度提交给查询协调进程，而协调进程则负责将该查询分配给查询服务器并将每个查询服务器返回的结果组合成一个整体。并行度是指分配给该查询的查询服务器数量。

Oracle服务器能对如下操作实现并行执行：

1. 连续
2. 排序
3. 表扫描

每种操作都需要制定查询的并行化方式。并行查询能达到的性能取决于用户访问数据的尺寸和该查询的并行度。

查询协调进程将确定查询并行化的实现方式。决定并行化实现方式的步骤如下：

1. 优化器决定该语句的执行计划。
2. 协调进程决定哪些操作能以并行方式执行。
3. 协调进程决定该查询使用的服务器进程数量。
4. 协调进程决定执行该查询操作的服务器进程。
5. 协调进程重组各个服务器进程的返回结果并将该结果返回给用户。

可通过下述过程确定查询语句的并行度。

1. 查询线索：SQL语句中用户定义的线索是确定并行度的先决条件。
2. 表定义：表定义是确定并行度的次先决条件。
3. 初始化参数：Oracle初始化参数是确定并行度最后考虑的条件。

不论上述条件取何值，查询服务器的数量都不能超过内存池中可用查询服务器的数量。而内存池中查询服务器的数量是由Oracle初始化参数PARALLEL_MAX_SERVERS确定的。

确定并行度的线索是在SQL语句中的注释字符串中定义的。定义查询并行度线索的语法如下：

PARALLEL(alias_or_tablename, [integer/DEFAULT] , [integer/DEFAULT])

并行度线索指定了被扫描的表名或别名，在表名或别名之后是该语句所使用的查询服务器数量（或为DEFAULT）。该语句中最后一个选项决定了在并行查询中表在不同实例间的分配方式。下面是一个使用查询线索的例子：

```
SELECT /*+FULL (emp) PARALLEL (emp ,4)*/
Employee_name
FROM emp
```

通过在该语句中加上FULL和PARALLEL线索，Oracle优化器将创建一个使用全表扫描的执行计划。另外，如果系统有足够的查询服务器，那么在执行全表扫描时将使用四个查询服务器。这个语句将忽略表和Oracle初始化参数所定义的默认并行度。

查询线索NOPARALLEL将关闭并行扫描表功能，并忽略表和Oracle初始化参数所定义的默认并行度。NOPARALLEL线索的语法如下：

NOPARALLEL (alias_or_tablename)

二、调整并行查询

在多处理器或并行处理计算机上，并行查询操作是非常有效的；在单处理器计算机上，如果大部分时间都化在了等待系统完成I/O操作上，那么使用并行查询也是非常有效的。具有足够I/O带宽的系统，尤其是使用磁盘阵列的系统，都能够从并行查询操作中受益。

如果通常情况下系统CPU的使用率达到了100%，并且系统中只有少量磁盘驱动器，那么使用并行查询是没什么意义的。同样，如果系统的内存非常紧张，那么并行查询也没多大意义。

对于并行查询，管理员可调整两个方面：一是系统I/O，二是并行服务器。通过正确设置数据文件，读者可提高并行查询的效率。

三、I/O配置

并行查询的功能是分解查询操作，以使查询操作能更有效的利用系统资源。提高查询操作效率的方法之一是当查询操作的一部分在等待系统完成I/O操作时，允许CPU继续处理该查询操作。如果整个表都位于同一磁盘驱动器上，那么并行查询操作将不能提高查询效率。

通过将表分成多个部分并将这些部分分别存放在不同的磁盘驱动器上，开发人员可用系统的分片存储功能、Oracle的分片存储功能或磁盘阵列（一种较好的方法）等方式实现表的分片存储。

大尺寸的连续盘区也有助于提高并行查询操作的性能。在表扫描操作中，查询协调进程将连续盘区分解成大、中、小三种不同的盘区组，直到完成整个表扫描操作为止。这样做的目的是实现每个查询服务器间的负载平衡。如果表中包含了一些大尺寸盘区，那么查询协调进程能更容易的找到大盘区组以分配给查询服务器。

系统的临时表空间可能需要存储一个分片存储卷上的几个大尺寸盘区。这种安排有主助于提高系统排序操作的性能。

四、并行度

在并行查询选项中，合理的I/O分布和并行度是最需要调整的两个方面。对并行度的调整一方面需要反复的试验，另一方面还需要一定的分析。对开发人员而言，在实验并行度时做好详细记录是非常重要的。应当首先根据如下一些因素考虑并行度：

1. 计算机的CPU能力：CPU的数量和能力将影响系统能运行的查询进程数量。
2. 系统处理大量进程的能力：一些操作系统能处理很多并发进程，而另一些操作系统则没有这方面的能力。
3. 系统负载：如果系统现在的运行已经达到了极限，那么对并行度的调整不会有太大效果。如果系统运行已达其能力极限的90%，那么太多的查询进程将使系统不堪重负。
4. 系统处理的查询数量：如果系统的大部分操作是更新操作，但仍有少量的重要查询存在，

那么开发人员可能希望系统运行多个查询进程。

5. 系统的I/O能力：如果磁盘上的数据是分片或是使用磁盘阵列存储的，那么系统能够处理多个并行查询。
6. 操作类型：系统是否需要处理很多的全表扫描或排序？并行查询服务器非常有助于这类操作。

这些因素对系统采用的查询并行度有一定影响。请记住：上述这些因素是帮助开发人员考虑并行度问题时能作出良好判断的一些方法。下面是关于并行度的另一些建议。

1. 诸如排序之类的需要大量CPU资源的操作应当采用较低的并行度。其原因是这类受限于CPU的操作已经充分利用了CPU，而不需等待系统的I/O操作。
2. 诸如全表扫描之类的需要大量磁盘I/O的操作应当采用较高的并行度。需要等待磁盘I/O的操作越多，系统就越能受益于并行操作。
3. 如果系统中有大量的并发进程，那么应当采用较低的并行度。因为太多的进程将使系统不堪重负。

在确定使用并行操作之后，读者可通过查询动态性能视图V$PQ_SYSSTAT来监控系统。可使用下面的查询语句来监控系统：

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

STATISTIC	VALUE
Servers Busy	0
Servers Idle	12
Servers Highwater	16
Servers Sessions	380
Servers Started	4
Servers Shutdwn	4
Servers Cleaned Up	0
Queries Initiated	21
DML Initiated	9
DFO Tree	77
Session active	12
Local Msgs Sent	2459361
Distr Msgs Sent	0
Local Msgs Sent	2459318
Distr Msgs Sent	0

15 rows selected.

在查看上述查询的输出结果时，下面一些统计信息是非常有用的。

Servers Busy：在任何时候处于忙碌状态的服务器数量，多检查几次这个数据以便清楚了解其平均值。如果这个值和初始化参数PARALLEL_MIN_SERVERS的值相等，那么开发人员可能设置了过多的查询服务器。

Servers Idle：在任何时候处于空闲状态的服务器数量。如果系统总是有很多服务器处于空闲状态，那么应当考虑降低PARALLEL_MIN_SERVERS参数的值。

Servers Started：实例已经启动的查询服务器的数量。如果处于忙碌状态的服务器数量很少，而启动的服务器却很多，那么可能只是零星的使用了查询服务器。

Servers Shutdwn：因为处于空闲状态而被关闭的查询服务器的数量。绝大多数情况下，这个值和Servers Started的值相同。

确定系统并行度之后，可以开始测试并行度，并分析从V$PQ_SYSSTAT视图和操作系统监控工具中获得的信息。开发人员还需要注意CPU的使用情况和系统过多的I/O等待。如果CPU使用率过高，那么应当降低系统并行度。反之，如果CPU使用率过低，并出现了大量的I/O操作等待时间，那么应当尽量提高系统并行度。

请记住：系统的并行度取决于SQL语句的线索、表定义和Oracle初始化参数等。初始化参数PARALLEL_MAX_SERVERS决定了查询服务器的总数量；而系统启动时查询服务器的数量取决于初始化参数PARALLEL_MIN_SERVERS的值。

系统正在使用的查询服务器数量等于系统正在执行的并行查询数量乘以并行度。如果试图使用比PARALLEL_MAX_SERVERS参数定义的更多的查询服务器，那么将不能执行并行化查询操作。

■ 并行创建索引

并行查询的另一个特征是以并行方式创建索引的能力。通过这个特性，系统用于创建索引的时间将大大减少。

类似于并行查询操作，并行创建索引操作也有一个协调进程调度两组查询服务器。其中一组查询服务器的功能是扫描需要创建索引的表以获得ROWID和与索引相关的字段值；另一组查询服务器则对第一组查询服务器所得到的值进行排序，并将排序后的结果传递给协调进程。而协调进程再根据这些排序之后的条目创建B树索引。

创建索引所采用的并行度规则和并行查询操作的优先级规则相同，即：第一优先级是CREATE INDEX语句中PARALLEL子句，第二优先级是表定义，最后一个优先级是Oracle初始化参数。

使用并行方式创建索引将比常规方式快好几倍。使用与采用并行方式创建索引的条件和并行查询的条件一样。一个被配制成能充分利用并行查询优势的系统也将在并行创建索引时获得良好的性能。

■ 并行加载数据

通过使用多个并发会话同时往同一个表中写数据可实现数据加载的并行化。对于不同的系统配置，通过并行方式加载数据，可以提高数据加载性能。因为加载数据不仅需要大量的CPU资源而且还需要大量的I/O操作，所以在高贷款的SMP或MPP系统中，并行化数据加载将提高数据加载的性能。并行化数据加载将通过多个直接加载器进程实现，每个加载器进程都使用PARALLEL=TRUE和DIRECT=TRUE两个选项。当开发人员将PARALLEL参数设置为TRUE时，加载器进程将不会对正在加载数据的表实行独占式加锁，反之，如果这个参数

的值不为TRUE，那么加载器进程将对正在加载数据的表实行独占式加锁。在并行加载数据过程中，加载器为每个并发进程创建了临时段，并行加载结束时将这些临时段合并起来。

尽管当每个临时文件都位于不同的磁盘时，并行数据加载具有良好的性能，但某些情况下，数据加载性能的提高不足以抵消为此所需的手工分片存储数据所产生的复杂性。所以建议开发人员在操作系统层次上采用分片存储表或磁盘阵列方式。通过将每个输入文件放在独立的卷上的方式，可充分利用系统顺序读取数据的优势。

并行加载数据是非常有用的，特别是在数据加载时间很紧张的环境中尤其如此。通过将每个输入文件放置在不同的卷上，测试人员可提高并行加载数据的性能。并行查询操作中所有的通用调整原则都是用于并行加载数据操作。

■ 并行恢复

并行恢复是并行查询选项的一个非常重要的特性。在评估Oracle和测试软硬件时，需要有意地使系统崩溃，以检验系统的可恢复性。通过使用并行恢复选项，恢复数据库和实例的时间将大大缩短。

当被恢复的系统有很多磁盘并支持异步I/O操作时，系统恢复所花的时间将会大大缩短。对于只有很少磁盘的小系统或不支持异步I/O的系统，采用并行方式恢复系统是不明智的。

在传统的系统恢复操作中，一个进程既从重做日志文件中读数据，又将用户对数据库所作的修改写进数据库中。因为这种恢复系统的方式必须等待磁盘完成I/O操作，所以这种恢复系统的操作需要花很长时间。

在并行恢复方式中，一个进程负责从重做日志中读取和调度重做入口并将这些入口传递给恢复进程，而恢复进程则负责将用户对数据库所作的修改写进数据库中。

因为调度进程是从重做日志中顺序读取数据，所以其I/O性能远高于向数据库文件中随机写入数据的进程。因为向数据文件写数据需要搜索大量数据文件，所以为系统中的每个数据磁盘分配一个或两个恢复进程是一个好主意。

系统的恢复进程越多，系统的I/O性能也越高，同时能使用所有的磁盘。因为数据库恢复是在实例启动时进行的，这时系统中还不能进行其他操作，所以这样的恢复方法缩短了系统的停机事件。并发恢复进程的数量是由初始化参数RECOVERY_PARALLELISM确定的。这个参数的值不能超过初始化参数PARALLEL_MAX_SERVERS的值。

通过指定足够数量的恢复服务器，可以立即降低实例恢复时间。如果系统不支持异步I/O或磁盘数量很少，那么不要使用并行方式恢复系统。反之，如果I/O子系统带宽很高，并且数据也被合理分片（通过软件或硬件实现），那么采用并行方式恢复系统能提供系统恢复的性能。

总之，在平衡操作负载方面并行查询选项是非常有用的，当其他进程在等待系统完成I/O操作时，并行查询可使CPU转而处理其他进程，从而充分利用系统的CPU资源。对于多处理器计算机，使用并行查询是非常有用的，但这并不是说并行查询不能用于单处理器计算机。

簇

簇（Cluster），有时被称为索引簇，是Oracle数据库中用于存储表的一种方法。在一个簇中，系统将多个相关的表存储在一起以缩短用户访问相关记录的时间。只有当这些相关表经常被同时访问时，才适合使用簇。对用户和应用程序而言，簇的存在是透明的，簇只影

响数据的存储方式。

在某些情况下使用簇是非常有利的，而另外一些情况下使用簇却可能非常不利。应当仔细考虑簇是否有助于提高系统性能。一般而言，如果集中存放的数据主要用于连接表中，那么使用簇是很好的。

如果两个表存放了相关数据，并且这两个表经常被同时访问，那么通过使用簇可将相关数据预装入SGA中，从而提高用户访问数据的性能。因为开发人员经常同时使用这两个表，所以在用户访问其中一个表时将另一个表的数据也放入SGA中可大大缩短用和访问数据的时间。

因为在表连接中，系统是在一次操作中检索所有的数据，所以在表连接中使用簇是非常有利的。下面一个例子说明了簇能发挥作用的地方。假设开发人员管理着一个销售数据库。为了降低数据库中的重复数据，创建了两个表。第一个表示销售人员表，这个表有如下字段：

ID

Name

RegionID

此外，还有一个记录地区信息的表，这个表有如下字段：

RegionID

RegionName

RegionManager

创建地区表的唯一原因是用来满足用户查询的需要，这样地区的名字就不会重复出现在销售人员表中。如果地区表中的RegionManager字段发生了改变，开发人员也不需要对销售人员表做大的改动。

另外，在如下所示的查询语句中，用户对销售人员表的所有访问都是通过地区表上的一个连接实现的。

```
SELECT Name, RegionName, RegionManager
FROM salesperson, Region
WHERE salesperson.RegionID=Region. RegionID
```

这个例子很好的说明了何种情况下适合使用簇。应当根据RegionID字段将这两个表形成簇。这个簇中同时存在于两个表中的字段称为簇键，必须为这个字段创建索引。

图4.7显示了这个作为簇的表的大致形状。请注意表中的簇关键字是RegionID。

如果用户经常同时使用这些信息，那么这个簇将有助于提高系统性能。除了能在用户同时访问这两个表时提高性能外，簇还能使用户非常容易的单独访问这两个表中的任何一个。如果一般情况下开发人员不会同时使用这些信息，那么簇将不能提高系统性能。并且这种情况下，簇实际上会导致系统性能的轻微下降，其原因是额外的表信息将占据更多的SGA空间。

簇的另一个不足之处在于当用户执行INSERT语句时将降低系统性能。引起性能下降的原因是簇在使用存储空间上采用的方法更加复杂，并且系统需要将多个表存储在同一个数据块中。簇表比单个表占用了更多的存储空间，这将导致系统扫描更多的数据。

总之，对于用户经常以连接方式同时访问的多个表使用簇是非常有用的。簇能降低将额外数据读入SGA的I/O操作并将这些数据预先放入缓冲区中，从而明显提高系统的性能。

如果这些表涉及到大量的INSERT语句或用户不经常同时访问这些表，那么簇将没有太大用处，开发人员也不应当为这些表创建簇。

如果系统经常对这些表中的某一个表做全表扫描，那么不应当为这些表创建簇。引为如果创建了簇，那么额外数据将占用SGA的部分空间并导致额外的I/O操作，这两方面的原因都会降低系统性能。

散列簇

散列簇和簇非常相似，但散列簇采用散列函数而不是索引访问簇键。散列簇根据散列函数的计算结果保存数据。散列函数是一个数学函数，这个函数根据散列键的值来确定簇中的数据块。

为了找到索引簇中的数据块，系统必须执行一个或多个与簇索引有关的 I/O 操作。在散列簇中，簇键告诉 Oracle 数据块的存放位置，这种方法降低了检索记录时的 I/O 操作数量。

索引簇根据记录的簇键值将相关数据存储在一起；与索引簇相反，散列簇是根据记录的散列值将相关的记录存储在一起。散列值的数量取决于 CREATE CLUSTER 命令中 HASHKEYS 参数的值。簇键的尺寸和数量非常重要，开发人员必须仔细计算这两个值。不要为系统经常只对其中一个表进行扫描的那些表创建散列簇。因为簇需要占用的额外空间，由此导致的额外 I/O 操作系统降低系统性能。

对于应用程序经常修改其簇键的表或本身就被经常修改的表不宜创建散列簇。其原因是散列键是基于数学函数的，如果系统需要不断重复计算散列健，那么将导致大量的系统开销。

何时创建散列簇呢？

尽管可以以类似于索引簇的方式创建散列簇，但你不必为所有表创建散列簇。实际上，开发人员更经常做的是为一个单独的表创建散列簇，以利用散列簇的特性。通过使用散列索引，系统只需一次 I/O，操作系统就能检索到用户所需数据，如果使用 B 索引，系统需要多次 I/O 操作才能检索到需要的数据。

因为散列簇采用了数据值来计算用户需要访问的记录的位置，所以散列簇特别适用于簇键具有惟一值且绝大部分查询为等式查询的表。在等式查询中，通常情况下系统只需一次读操作就能检索到需要的数据，簇键不必是表中一个单独字段。如果用户的一般查询使用了与多个字段相关的等式，那么可使用这些字段创建一个复合簇。

当一个或多个表的尺寸比较固定时，采用散列簇能使系统些性能实现最优。如果表一直保持在系统为之分配的初始空间内，那么一般而言散列簇不会导致系统性能的下降。然而如果表的增长超出了系统为之分配的初始存储空间，那么散列存储将导致系统性能的下降，其原因是表需要更多的块存储数据。

由于系统处理散列簇时将读取只包含少量数据的块，所以散列簇有可能会降低系统扫描表的性能。因为在初始状态下，系统是根据簇键值来将数据分配到簇中的，所有某些数据块中可能只包含极少量数据。

当簇键值发生变化时，散列簇也将降低系统性能。因为存储数据的块的位置是根据簇键值来确定的，所以当簇键值发生变化时，为了维护散列簇，系统需要将数据从一个块迁移到另一个块。

适合创建散列簇的表应具有以下特征：

1. 簇键值具有唯一性。

2. 用户对表的大部分查询都是关于键值的等式查询。
3. 表的尺寸是静态的，几乎不会出现任何增长。
4. 簇的键值不会发生任何变化。

一个适合于创建散列簇的典型例子是一个用于存储零件信息的表。通过建立一个关于零件号的散列键值，用户对该表的访问将非常有效而且非常快。任何时候，如果系统中有一个静态的表，其中的一个或多个字段的值具有惟一性特征，那么可考虑为这样的表创建散列簇。

和索引簇一样，使用散列簇既有优点也有不足。当用户使用关于簇键的等式查询检索数据时，散列簇具有很高的效率。如果用户不是根据簇键值检索数据，那么这个查询就不能被散列化。这就像在索引簇中见到的那样，当在散列簇表中使用 INSERT 语句时，将降低系统性能。

在考虑创建索引簇和散列簇时，应当根据用户访问该表的方式来判读簇能否提高系统性能。和 RDBMS 其他很多方面一样，如果根据错误的决定调整系统，那么最终结果将降低系统性能。

如果在创建散列簇时严格满足了散列簇的一些条件，那么可以充分利用散列簇的优势。如果创建的散列簇满足了上述所有条件，那么散列簇将极大地提高系统性能。

同时读取多块数据

当系统执行表扫描时，Oracle 具备同时读取多个数据块的能力，这种能力提高了系统的 I/O 速度。通过同时读取多块数据，Oracle 能够从磁盘上读取更大的数据块，从而避免了对磁盘上数据进行搜索的操作。通过降低磁盘搜索和读取更大的数据块，可以降低系统的 I/O 开销和 CPU 开销。

Oracle 的这个特性被称为多块读取。只有对于连续数据块，多块读取特性才有助于提高系统性能。通常情况下，一个盘区内的数据块都是连续的。如果数据存放在多个小盘区上，那么多块读取的性能将会降低。

Oracle 初始化参数 DB-FILE-MULTIBLOCK-READ-COUNT 指定了一次多块读取能读取的数据量。因为这个参数几乎不会导致系统性能的下降，所以一般情况下都将这个参数的值设置得很高。I/O 的尺寸取决于初始化参数 DB-FILE-MULTIBLOCK-READ-COUNT 和 DB-BLOCK-SIZE 的值。

为了充分发挥多块读取数据的优势，应当尽量配置自己的系统以使数据库的块尽可能都是连续的。为了做到这一点，应当使用优化尺寸的盘区来创建数据库。盘区的尺寸应当远大于一次多块读取操作所读取的数据的尺寸。

然而，创建这样的盘区可能并不是一件简单的工作。如果盘区尺寸过大，那么将导致 Oracle 很难找到足够的连续空间以创建盘区。另一方面，如果盘区尺寸过小，那么不仅会影响多块读取的性能，还会引起更多的存储空间的动态扩展。了解系统的初始数据和数据增长模式将有助于开发人员合理设置数据库系统盘区的尺寸。

分区

分区（Partition）是 Oracle8 的一个新特性。这一特性的目的是允许大尺寸表或索引分解成小的更容易管理的部分，即分区。与原来的表或索引相比，因为分区变小了，所以系统访问分区的速度更快、效率也更高，但分区的存在对用户而言是透明的。开发人员可将分区

想象成表或索引被分解之后的多个小尺寸的表或索引。系统可单独访问这些小尺寸的表或索引，也可以以组或整体方式访问之。

类似于索引，分区也是基于特定的键创建的。可以使用键字段或键字段集来将数据分解出成适当的分区。每个分区都有它自己的 Oracle 段。这些段的总和就组成了数据库的表或索引。

可使用多种分区方案来创建分区。这些方案确定了如何将数据分解成分区。可使用如下一些方案：

1. Range Partitioning：这种方案根据数据的范围，比如月、年等等对表中的数据进行分区。
2. List Partitioning：这种方案和 Range Partitioning 分区方案很类似，但这种方案是按照数据的值而不是数据的范围来进行分区划分的。
3. Hash Partitioning：这种分区方案使用散列函数来实现对数据的自动分区。
4. Sub- Partitioning：这种方法就是开发人员熟悉的复合分区方法。这种方法允许开发人员同时使用多种分区方案。

在下面几部分内容中，开发人员将会了解到各种分区方案的特点，并能够确定哪种分区方案适合于自己的系统。

■ 分区的概念

分区的目的是用于大尺寸表，在这些表中，系统无法有效的检索数据。索引的低效性主要是因为表中的数据量过大，且是一个整体。某些情况下，报表需要大量数据，且在相关查询语句的 WHERE 子句中包含了降低数据处理量的一些谓词，但在这种情况下因为系统需要检索的数据量非常大，所以这方面的操作仍然是低效的；这是 Oracle 的分区功能可以提高系统效率。

请考虑这样一种情形：查询之处理在最后一个月内出现的数据。例如：

```
SELECT region,sales_person,SUM(sales)
FROM sales_data
GROUP BY region,sales_person
WHERE data >= TO_DATA (‘2001-01-01’,’YYYY-MM-DD’)
AND data < TO_DATA (‘2001-02-01’,’YYYY-MM-DD’);
```

在这个例子中，选择了 2001 年 1 月中的所有销售数据。因为一月份发生的销售数量非常大，所以系统需要选择如此多的值，从而为事物记录的日期字段建立索引肯定是非常低效的。如图 4.8 所示，系统将对 salesdata 表做扫描操作。

通过将该表中的数据按发生的月份进行分区之后，系统仍然会对该表进行扫描，但是 Oracle 优化器将会发现对表进行了分区，并且查询语句的 WHERE 子句中包含了分区关键字。如图 4.9 所示，因此，虽然系统仍然会对该表进行扫描，但却只扫描必要的分区：

因此，系统从内存中读取的数据、CPU 处理的数据以及系统的 I/O 操作都降低了，这不仅提高了系统性能而且节约了系统资源。Oracle 优化器将自动完成所有这一切。

■ 按数据范围进行分区

这个分区方案允许将数据分区成不同的范围（这即是按数据范围进行分区名字的由来）。当使用这种方案创建分区表时，表和分区的定义如下：

```
CREATE TABLE rangetable(
Id1 number,
Id2 number,
```

```
Name1 varchar2 (40) ,
Name2 varchar2 (40) ,
Name3 varchar2 (40) )
PARTITION BY RANGE ( id1 )
(    PARTITION part1 VALUES THAN (2501)
TABLESPACE rangets1,
PARTITION part2 VALUES THAN (5001)
TABLESPACE rangets2,
PARTITION part3 VALUES THAN (7501)
TABLESPACE rangets3,
PARTITION part4 VALUES THAN (MAXVALUE)
TABLESPACE rangets4 );
```

从上面的例子可以看，按数据范围进行分区的一种普遍方式是按照数据发生的日期进行分区：

```
CREATE TABLE sales_data
(region VARCHAR2 (10),
sales_person INT,
sales_amount NUMBER (10)
sales_data DATE)
PARTITION BY RANGE (sales_data)
```

在这个例子中，每一月份都有自己的分区。用户插入 sales_data 表中的数据将被系统自动分配到相应的分区中。

在分区中包含 MAXVALUE 范围常常是一个好主意，如果创建的分区中不包含这个范围，而某条记录的值却不在指定的所有任何分区之内，那么插入这条记录的 INSERT 语句或更新这条记录的 UPDATE 语句将失败。

- List 分区方案

List 分区方案和按数据的范围进行分区的方案非常类似，只是该方案是用了值的列表作为分区依据而不是使用数据的范围。这个列表中包含了一系列不同的值，系统使用这些值来将记录分配到相应的分区中。下面是一个使用 List 分区方案的例子。

```
CREATE TABLE sales_data
(region VARCHAR2 (10),
sales_person INT,
sales_amount NUMBER (10)
sales_data DATE)
PARTITION BY LIST (region)
(
PARTITION office1 VALUSE IN (1,2,3,4) ,
PARTITION office2 VALUSE IN (5,6) ,
PARTITION office3 VALUSE IN (7,8,9,10)
);
```

在 List 分区方案中，不能指定 MAXVAL，因此开发人员必须确保值列表中包含了所有可能的值。

- 散列分区方案

散列分区方案使用一个散列函数来确定应当将数据存放在哪个分区中。这种分区方案是用于开发人员不清楚如何合理分区数据，但又感觉到对数据进行分区确实有助于提高系统性能的表。例如本章前面所使用的 sales_data 就是这样一个例子。

如果用户通过 sales_person 字段访问了该表的大量数据，但开发人员不清楚销售人员的分布情况，那么应当为这个表创建一个散列表。对那些在 WHERE 字句中使用 sales_person 字段作为查询条件的查询，散列分区是非常有利的。这种情况之所以非常适合使用散列分区方案是因为开发人员可能不知道数据的分布情况。表中的很多记录可能都包含了同一销售人员的 ID，因而弱化了任何可能存在的数据的惟一特性。

可用如下语句创建一个散列分区：

```
CERATE TABLE sales_data
(region VARCHAR2 (10),
sales_person INT,
sales_amount NUMBER(10)
sales_data DATE)
PARTITION BY RANGE(sales_data)
SUBPARTITION BY HASH(sales_person)
SUBPARTITIONS=4
STORE IN    (sp1,sp2,sp3,sp4 );
```

在上面的例子中，散列函数将 sales_data 表分解成四个分区。

■ 复合分区方案

复合分区方案是指在一次分区操作中使用了两种不同类型的分区方案。例如：可能首先使用了基于数据日期范围的分区方案，然后在这种方案确定的分区中再使用散列分区方案。这种分区方案特别适用于那些尺寸巨大的表，在这样的表中，仅有一种分区方案不足以降低用户访问的数据量。下面就是一个复合分区放案的例子：

```
CREATE TABLE sales_data
(region VARCHAR2 (10),
sales_person INT,
sales_amount NUMBER (10)
sales_data DATE)
PARTITION BY RANGE (sales_data)
SUBPARTITION BY HASH (sales_person)
SUBPARTITIONS=4
(
```

这个例子使用了基于数据范围的分区方案和散列分区方案的混合分区方案来创建一个分区表。根据不同的数据和访问方式，复合分区方案能极大提高系统性能。

■ 分区的好处

分区有以下几方面的好处：

1. 对能被分区的大尺寸表进行扫描时，分区可降低 I/O 操作和 CPU 的使用率。
2. 可在分区的层次上而不是表的层次上加载数据。
3. 能以删除分区的方式删除数据，而不必使用 SELECT 语句来删除大量数据。
4. 对用户和应用程序而言，分区是完全透明的。
5. 可在分区层次上而不是在表层次上维护数据。

在很多环境中，这些好处能极大提高系统性能；然而如果要充分利用这些优势，开发人员的数据和应用程序必须要适合于分区。

■ 分区和索引

Oracle 为使用分区的索引提供了不同选项。根据个人的喜好，开发人员可对索引进行分区，也可不对索引分区。可采用与分区表相同的方法对索引进行分区。分区索引中包括了很多不同的选项，下面列出了这些选项的一部分：

1. 开发人员可在一个分区表上创建一个非分区索引。这个索引称为全局非分区索引。
2. 可对分区或非分区表创建一个分区索引。索引的分区键不需要与表的分区键相同。
3. 可使用不同的键在分区表上创建一个分区索引。这个索引成为全局分区索引。
4. 可对同一个表的不同分区分别创建不同的索引。这样的索引称为局部分区索引。

正如分区表的目的一样，分区索引的目的也是用来降低系统检索的数据量。一个典型的例子是在 sales_person 表上创建一个索引，然后再对索引按日期范围进行分区。对于大尺寸表，分区方法降低了查询时系统遍历的索引尺寸。当然，为了使这种方法发挥作用，开发人员必须在查询语句中同时使用表分区键和索引分区键。

稳定性计划

稳定性计划是 Oracle9i 中的一个新特性，这个特性允许 Oracle 优化器产生的执行被存储起来以备以后使用。存储并允许修改执行是一个很好的想法。即使系统的外部因素发生了变化，例如系统的统计数据甚至 Oracle 优化器的版本发生了变化，这个特性也是 Oracle 能以一种稳定的方式执行命令。

当系统改变引起新的执行计划从而导致系统查询性能降低时，稳定性计划是非常有用的。尽管优化器是 Oracle 的一种非常好的工具，并且优化器所做的绝大多数决定都是最优的，但稳定性计划也可能不能提供所有情况下的最优性能。

多线程服务器

用户可通过专用服务器进程连接到 Oracle 实例，也可以通过多线程服务器进程连接到 Oracle 实例。不论采用哪种连接方式，绝大多数情况下，对数据库终端用户而言，这两者的外观和行为都是一样的。因为每一个专用服务器进程都将占用大量内存资源和系统资源，所以有必要对多用户连接采用多线程服务器进程。

某些时候需要采用专用服务器进程连接系统，比如以下一些情况：

1. 启动和关闭数据库。
2. 执行介质恢复。
3. 执行批处理作业。

在这些情况下，使用专用服务器将使系统工作得更好

■ 专用服务器进程

当一个用户的 SQL 请求被传递给 RDBMS 时，执行该命令的不是用户进程而是服务器进程。这种机制避免了 Oracle 被一个用户进程直接操纵，从而增加了 Oracle RDBMS9i 的稳定性和健壮性。

对于专用服务器进程，用户进程和服务器进程之间是一一对应的。在系统启动了多线

程服务器进程的情况下，为了启动一个专用服务器进程，用户必须在连接字符串中使用 SERVER=DEDICATED 参数。这个参数将为用户进程创建一个专用服务器进程。

■ 多线程服务器进程

多线程服务器进程允许多个用户使用一定数量的共享服务器进程。

所有对共享服务器的请求都必须经过调度进程，这个过程对 SAG 大缓冲池中的用户对共享服务器进程的请求进行排队。当系统处理完用户请求后，系统通过 SGA 中的大缓冲池将请求结果返回给调度进程。

因为共享服务器进程使用共享缓冲池对用户请求进行排队并返回数据，从而大大减少 CPU 和内存的使用。然而，正如读者所猜想的那样，多线程服务器也会增加系统开销。这就是为什么执行需要很长时间的批处理作业时使用专用服务器的原因。

■ 调整多线程服务器进程

对配置和调整多线程服务器，需要调整参数文件中的以下参数。还应当小心监控系统的大缓冲池，以确保系统的运行没有超出分配给它的内存空间。

应当尽量监控只有少量用户的共享会话所使用的内存。监控结果将会告知共享会话使用了多少内存。利用这些数据，就能估算所有会话总共需要多少内存。可通过下面的 SQL 语句获得这些信息：

```
SELECT SUM(value) || 'bytes'""Memory"
FROM v$sesstat,v$statname
WHERE name ='session memory'
AND v$sesstat.statistic#=v$statname.statistic#;
```

上面语句的输出结果告诉开发人员使用了多少内存。将使用的总内存量除以连接数，就可确定每个会话使用的内存数量。然后就可从每个会话使用的内存量来推算为了支持系统的所有会话，需要为系统的大缓冲池分配多大内存空间。

如果认为大缓冲池的内存太少，那么可通过调整初始化参数 LARGE_POOL-SIZE 来增加大缓冲池的内存空间。请记住：除多线程服务器之外，系统还使用大缓冲池来执行并行查询操作。

初始化参数 MTS_DISPATCHERS 决定了每个网络协议使用的调度进程呢购的数量。通过提高这个参数的值，用户可能会看到用户会话的性能得到了提高，其原因是用户会话不再需要等待可用调度进程。在下面一个例子中将会看到如何为 TCP/IP 协议配置五个调度进程。MTS_DISPATCHERS="TCP,5"

需要分别为不同的网络协议设置调度进程。对于用户数量较少的协议，应当设置较少的调度进程。并发会话数量与调度进程之比越大，用户在使用调度进程时，等待调度进程的可能性就越大。除前面讨论的参数之外，还有如下一些参数与多线程服务器有关。

1. MTS_MAX-DISPATCHER：实例能创建的调度进程的最大数量。这里的调度进程包括所有网络协议的调度进程。
2. MTS_SERVERS：实例刚启动时的共享服务器进程数。如果将这个参数设置为 0，那么 Oracle 将不会是用共享服务器进程。为了满足系统需要，共享服务器进程的数量将会动态增长。
3. MTX_MAX_SERVERS：这个参数指定了实例中允许运行的共享服务器进程的最大数量。

可使用 ALTER SYSTEM 参数动态地修改调度进程的数量和共享服务器的最小数量。

● 总结

本章讨论了很多概念和思想，开发人员可利用这些内容进一步提高系统性能。本章讨论的内容更加具体，根据系统的不同配置，读者可能会使用本章讨论到的技术也可能不会使

用到这些技术。实际上，如果对本章讨论的一些技术把握不好，那么这些技术可能会损害系统性能。

如果应用程序能利用簇和索引的优势，那么簇和索引将有助于提高系统性能。如果应用程序使用了所以而又需要频繁更新数据。那么这样的索引将会降低系统性能，其原因是在数据更新的同时，系统也需要更新索引。同样的，如果用户的数据访问模式使用了簇，那么簇将会有助于提高系统性能；反之，簇将会成为系统的负担。

Oracle 并行查询选项和并行服务器选项能极大提高系统性能。通过将查询分配成多个服务器进程，并行查询选项提供了一种提高系统 CPU 使用率的机制。并行服务器为开发人员提供了一个健壮的可升级的服务器群集功能，这个特性不仅能提高系统性能，还为系统提供了立即排除错误的能力。应用程序和需求将决定这些特性是否有助于提高系统性能。在调整系统时，非常重要的一点是需要注意调整系统所冒的风险。在上一章中，我们讨论了不同的系统调整主体，例如共享内存池的尺寸和数据块缓冲区等等。提高共享内存池和数据库缓冲区的尺寸将会提高缓冲区的命中率，但这样的调整也存在着损害系统性能的风险。然而，如果读者对本章讨论的某些系统特性配置不正确，那么它们对系统性能造成的负面影响会更大。任何时候调整系统，都应当预先估计该调整所面临的风险，特别是在没经过详细测试就对生产系统进行调整时更是如此。

如果对系统需要调整的方面和该调整所影响的方面进行了仔细考虑，并对应用程序的数据访问模式有深入了解，那么可以将调整面临的风险降到最小。应当仔细考虑对系统所做的修改将会对系统造成的影响，并尽量了解系统修改对应用程序的影响。只有这样，才能在优化系统性能的同时，将系统调整的风险降到最低。