

第四部分 性能调整

第16章 性能调整基础

本章要点：

再看物理设计

为什么进行调整

调整规则

调整目标

使用投资的回报

再看应用类型

理解基准测试

使用Oracle诊断工具

16.1 再看物理设计

第3章“物理数据库设计、硬件和相关问题”中说明物理（和逻辑）设计是调整数据库的第一步。事实的确如此，在许多有关调整方面书籍中的一个首要步骤便是调整设计。通过采用正确的逻辑和物理设计，你便可以完成这一步骤。如果所有的步骤都已完成的话，你的设计将使你的数据库能够在一个可接受的水平上运行。然而，有两件事情总是伴随着数据库系统：增长和变化的应用需求。

假如你的数据库主要由查询表组成且保持相对不变，或者由于其他原因而使得数据库很少增长或增长得非常缓慢，那么这个因素不是问题。然而，如果你的数据库和大部分其他数据库一样以10%或更高的百分率过度增长的话，那么，不管最初的逻辑与物理设计多么好，你都需要周期性地调整该数据库以便容纳或补偿此增长。此外，所有的数据库系统已经变成不断重定义应用这一现象的牺牲品。这里有一个你或许曾经遇到过的例子：

经理：“应用程序不工作了。它的运行速度太慢了。数据库怎么了？”

程序员：“代码没有改变。”

数据库管理员：“数据库没有改变。”

那么问题出在哪里？在本实例中，使用（usage）可能已经改变了。还有一种流行的说法叫做应用需求已经改变了。例如，假定原先的应用程序用来实现对七个表分别完成只读存取；然而，如果该应用程序目前用来将所有这七个表合起来以收集比原先要求的更多的信息的话，那么或许需要进行一些调整。事实上，或许需要完成某些物理再设计，这些物理再设计也称为物理重构。

你可以在物理上重构一个数据库以提高其性能并仍然可以使应用随意地运行。这对 Oracle 以及其他厂商来说是可能的，因为 Oracle 遵守 Codd 规则8：物理数据独立。因此通过与用户联

系，答案或许可以找到：

用户：“应用程序不工作了吗？我们正在使用一些不同的表。”

除了增长与变化的应用需求外，还有一件需要利用数据库管理员的性能调整技巧的事情：不良设计的数据库。

由于有增长和变化的应用需求，所以你已经假定数据库被很好地设计。不幸的是，正如许多数据库管理员所知道的那样，事实经常并非如此。很多时候，某个数据库管理员要接收另一个数据库管理员留下的数据库系统，或者面对一个由于财产迁移而得到的数据库系统，这被称为修理别人的数据库系统性能。在诸如此类的情况下，在调整性能以前需要做的事情是逻辑和物理分析与重新设计。数据库有可能不需要大的再设计，但也有可能需要大修。

在任何一种情况下，当这个设计已解决时，数据库管理员能够推进到性能调整。然而，在重新设计后，最好首先把该应用程序投入使用并做一些性能监控工作。毕竟，正确的设计（或重新设计）一开始都能带来好的性能。很可能不再需要新的额外的调整（实际上，我发现事实就是如此）。

16.2 为什么进行调整

为什么要调整？你已经回答了这个问题。根据用户定义的准则，当前的数据库系统由于如下的理由而不能令人满意地运行：

不良的设计。

增长。

变化的应用需求（可能包含对可接受性能的重新定义）。

什么时候数据库的调整工作不能完全有效？当对于数据库来说是外部的但对于整个客户 / 服务器应用程序的性能至关重要的组件未能满意地运行时，如果没有相应的其他应用基础结构部分的调整，数据库调整或许就不会产生效果。除了大部分独立的批量数据库应用以外，大部分现代数据库系统是基于客户 / 服务器结构的。

对于后端数据库来说，属于外部的主要组件是后端操作系统（或操作系统），网络和客户操作系统。主要例子如下：

非常微弱的客户（PC）。

网络饱和。

非常弱的、已饱和的或不良调整的操作系统。

从客户（PC）所拥有的较老的 CPU（486或更低级）、相对小的内存（16MB或更小）或较低的存储器容量（1GB或更低）的意义上来说，它们或许是弱的。尤其是在应用对客户来说是重量的并由此需要一个胖的客户的情况下。胖客户（fat client）是指自己完成大部分应用处理的客户。瘦客户（thin client）显然是胖客户的对立面，它的大部分处理发生在服务器端。除非应用程序仅需要哑终端存取或瘦客户，否则弱客户将不足。然而，瘦客户将足以满足因特网计算范型，在该范型中网络浏览器是客户端的唯一要求。

通过改变定义，网络被认为已达到饱和。饱和意味着一个系统组件已经达到了最大的可能吞吐量。一个10Mbps的以太网在达到自身带宽的30%时就被认为是饱和的。网络专家用不同的方法处理饱和问题，包括重新分割本地网络，增加更多的网络段或者改用宽带网络介质类型，诸如100Mbps的光纤分布式数据接口（FDDI）。总之，你已经很好地踏上了数据库调整

之路。

就像客户一样，操作系统可以很弱。它仅相对于它必须支持的应用程序来说是弱的。一般而言，具有4个32位的处理器、256M内存和10G磁盘空间的现代数据库服务器被认为是弱的。就像网络一样，操作系统可以在各种情形下饱和。在重负载的情况下（许多并发用户或大量的数据移动），操作系统的CPU、输入/输出和内存能够达到饱和。

另一种方法是当组件饱和时，系统在那个组件上阻塞。因此，假如一台 UNIX机器的CPU是饱和的，例如达到85%或更高的利用率，那么这个UNIX系统就被认为是在CPU上阻塞，或简单地称为CPU约束。

最后，操作系统或许仅需要一些回报。例如，DSS仅需要把非常大的文件存储在磁盘上，系统管理员（SA）可以调整UNIX文件系统以便它把非常大的文件最佳存放在连续字节块中。

就像你将在第17章、第18章、第19章和第20章中看到的那样，调整数据库系统和调整操作系统缠绕在一起。对于系统管理员和数据库管理员来说这是一件好事情，因为在调整数据库系统或操作系统时不需要请求允许，你只需做它便是。然而，在实际中，数据库管理员经常与系统管理员以及资源分配和调整问题的管理程序发生冲突。我的建议是通过Oracle的诊断工具收集适当的证据并使用这些图表证明你的资源需求是正当的。Oracle的诊断工具稍后在16.8节和第17~20章中讨论。

16.3 调整规则

现在列举你几乎能在任何一本性能调整参考书中发现的主要性能调整规则。

主要的性能调整规则如下所示：

- 1) 划分与克服。
- 2) 预分配、预取数据和预编译。
- 3) 优先次序。
- 4) 大块、块和批量。
- 5) 对应用进行适当地分段。

这些看起来熟悉的规则便是主要的性能调整规则。它们是在第3章中讨论过的物理设计规则，其中2号和3号颠倒了一下顺序。现在看一下这些规则如何应用到性能调整，而不仅仅是物理数据库设计中。

16.3.1 调整规则1

本基本规则是划分与克服。实际上，你想使用并行技术作为对瓶颈的补救。

看一下以下情况：假如性能监控显示数据库系统有一个较为集中的繁重物理读取和写入某一盘的过程，为了实现并行存取，需要将Oracle的物理和逻辑结构分开。通常，一个应用有几个用户或者进程正试图存取那张盘上相同的表或表空间。必须被连接在一起的表经常存在于同一张盘上。这导致性能变差。

操作系统必须代表Oracle为该表的一些数据块存取该磁盘，为第二张表再次执行本循环，然后为两张表的所有块重复整个过程，直到Oracle已经把两张表连接在一起。硬件要做的事情是磁盘的读/写头必须读取第一张表的申请扇区（物理数据块），重新定位读/写头以读取第二张表的申请扇区直到找到为止，然后重复。磁盘的存取时间（用毫秒表示）包含搜索时间和

等待时间（转动延迟）。搜索时间指已提及的重新定位，读/写头必须沿着磁盘的半径向里或向外移动以寻找包含申请扇区的磁道。搜索时间控制等待时间或者用于磁盘旋转以存取扇区的时间。

重要的事情是连接同一张盘上的两个或多个表需要大量的搜索时间用于磁盘读取。作为一个数据库管理员，你的目标是减少搜索的次数。你想使争用减到最小并消除此瓶颈。要做这项工作，你将连接到不同磁盘上的表分开。这种方法同样适用于那些共存于同一张磁盘上的表，这些表尽管没有连接在一起，但它们能够被同时存取。结果是相同的：瓶颈。解决方法是相同的：分开它们。

通过这种方法，RAID对这个方案就像标准磁盘一样容易敏感。起初，RAID，尤其是1、3和5级（曾在第3章中讨论过）看起来不可能遇到同样类型的问题。明显地，去除几张盘中的一个单表（一个RAID卷）无疑会提高性能。然而，假如该表与一些必须和它连接或至少同时存取的表一起存储在同一个RAID卷中的话，你会遇到相同的问题：瓶颈。解决方法也是相同的：分开它们。

你仍然遇到瓶颈问题的原因起初不是很明显，但要仔细思考这个问题。在RAID卷的所有盘中去除所有的表块（存储在它们各自表空间的数据文件中）。尽管没有一张表在任何一张磁盘中作为一个整体存在，然而所有的表块共存于全部磁盘中。今后，你还会遇到同样的问题。如同标准磁盘设置一样，不是把已连接的或同时存取的表分散到不同的磁盘上，而是把这些表分散到不同的RAID卷中。

16.3.2 调整规则2

本基本规则是预分配、预取数据和预编译。只要可能的话，提前做此工作。

看一下以下情况：假定你有一个特别易变的数据库系统，它明显地且频繁地进行增长和收缩。事实上，作为一种应用类型，可以把它叫做VCDB。当在非增长长期访问数据库时，性能较为合理。然而，在它的增长阶段，性能非常差，尤其是在增长初期。

这里可能发生的事情是动态增长降低了联机系统的速度。在Oracle术语中，这被称为动态扩展（dynamic extension）。因为正在更新或插入行，所以Oracle必须扩展，或者在创建或改变时按照由存储子句所指定的分配表的下一个区域，或者采用表空间缺省值。在这两种情况下，从创建时刻直到最近的增长期，表都有足够的空间，然后必须扩展以适应它。当然，简单地将一张表在Oracle中进行扩展不是世界末日，事实上，它是一个事件的正常过程。

然而，当一张表经常扩展时，特别是为大量数据进行扩展时就会损害联机存取，这就成为一个大问题了。并发用户不应该遭受损失。对于这种需要频繁或大量数据扩展的数据库系统而言，最好使用预分配。你需要创建具有足够存储量的表来解决预期的最大峰值容量。在表（或表空间）的创建语句的存储子句中，设置INITIAL和NEXT扩展容量，然后设置MINEXTENTS，这样 $INITIAL + ((MINEXTENTS - 1) \times NEXT)$ 等于该表的最大预期峰值容量。

16.3.3 调整规则3

本基本规则是优先次序。首先解决最重要的问题以便在投资上获得最大的回报。

看一下以下情况：你有一个运行缓慢的批处理系统。当你执行产品运行时，该系统缓慢运行。在检查了组成应用的SQL代码后，发现有几个要重新编写的效率非常低的程序。要完

成这项工作或许会花费2个月时间。然后你分析性能监控统计数字，发现回滚显示了高的争用，这显然是因为你正在同时运行几个并发程序以帮助加速运行。通过增加更多的段，修改回滚，但是你的运行时间几乎没有改善。要做什么？

不管你如何调整后端，前端（或应用代码）对客户/服务器系统的总时间的使用进行支配，这不会引起网络和其他问题。你必须首先调整应用，然后调整数据库。当产品比以前运行得快时，投资便很值得。

假如此调整由于一些原因而不可行，那么其他诸如逻辑或物理重构的数据库努力可以作为临时措施来实施。但老实说，根本不应该这样做；你正在为了适应不良的应用设计而改变好的数据库设计。使用那种方法，你不会收获很大。

16.3.4 调整规则4

本基本规则是大块、块和批量。要处理的一组适当的事情聚集在一起并且彼此不会为资源产生竞争。

看一下以下情况：你的DSS经由因特网给外部分析员提供实时、只读信息。你把一个Web页做为数据库系统的前端。并发有时是中小规模的（最多有100个用户），但是不会引起问题。用户所抱怨的是从大表中选择时的吞吐量。

这有一个典型的例子。当从一张表中选择大部分或全部行时，优化程序必须选择任何索引上的全表扫描作为它的存取路径。假如这是用户所要求的话就不能避免，这种类型的存取是应用需求的一部分。你能做什么来加速它？你一开始可以在你的UNIX系统中将DB_BLOCK_SIZE设置为8K字节，然后可以把该值增加到16K或32K。块越大，读的时候效率越高。换句话说，对块的每一次读取你都可以得到更多的数据行，因此，通过增加大数据查找的块容量使读取的效率更高。

必要时你还可以考虑增加DB_FILE_MULTIBLOCK_READ_COUNT和DB_BLOCK_BUFFERS。同时读取更多的块和增加块缓冲区也会对DSS有所帮助，稍后你将在第18章和第19章中学习相关的更多知识。

16.3.5 调整规则5

本基本规则是对应用进行适当地分段。把各块应用分配到适当的地方（客户、中间层或服务器）。

看一下以下情况：你有一个执行命令输入和控制的客户/服务器数据库系统。然而，命令输入屏在PC机上运行缓慢。反应时间很长，应用程序似乎使客户负担过重，因为一个输入要花费几秒钟的时间（就反应时间而论是慢的）。

你首先需要检查应用是如何被分区的。大部分数据验证工作在前端代码中完成，这可能是出现故障的地方，因为它对大部分的代码行进行解释。大部分前端代码算法如下所示：

- 1) 把日志记录到数据库。
- 2) 从一些表中读取数据。
- 3) 把数据键入到屏幕上。
- 4) 等待前端对一些表中键入的数据进行验证。
- 5) 假如完成的话将日志取出，否则重复步骤2~4。

伴随于此的问题是事务在整个时间处于激活状态，它交互式地验证前端表中的数据。这个验证过程（步骤4）应该在服务器上非交互式发生，它属于那儿因为它直接处理数据。处理通常应该尽可能地定位在靠近它们操作的数据的地方，除非有一些强制理由（例如服务器功率不足）。对先前的应用进行分段的标准方式将它划分为3个主要的步骤：在事务中读取（输入数据），没有事务的情况下进行本地修改（交互式地）以及在事务中回写变化（输出数据）。应用分段（application segmenting）也称为事务截断（transaction chopping）。如果正确操作的话，它将缩短反应时间。

16.4 调整目标

有不同的方式确定性能调整工作的目标。数据库管理员会全部考虑它们。考虑到你的应用类型，这曾在第3章中讨论过，稍后在本章中将再次讨论（参见16.6节）。数据库系统可以用各种定量的方法分析，这也在第3章中也讨论过，稍后在本章的16.7节中继续讨论。这些中最重要的如下：

吞吐量——每个单元时间完成的工作，以每秒钟的事务（tps）量表示；越高越好。

反应时间——应用做出反应的时间，以毫秒或秒表示；越低越好。

背景时间——应用占用的运行时间；越低越好。

在任何一个系统中，吞吐量和反应时间作为调整目标通常是互相对立的。如果反应时间长（坏），吞吐量或许高（好）。如果吞吐量低（坏），反应时间或许低（好）。

常识有助于挑选出这两个冲突的度量标准。多数的并发用户在一定时间内使用一个系统，每个用户很有可能比平时要经历更长的延迟，但是通过系统的事务数量将会更大。相反，假如你减少在某一个时间窗口中访问系统的并发用户数量，以在该时间内完成的全部事务量减少为代价，每个用户将会享受到更快的反应时间。

一般说来，依赖于应用的要求，OLTP系统需要更低的反应时间或更高的吞吐量。DSS希望低的反应时间。然而，DSS也可能希望得到高的吞吐量（每个单位时间读出或写入的块的数量）。这种类型的吞吐量不一定对高并发性和低反应时间起反作用。一个批处理（生产）系统一般需要较低的背景时间。例如，每个人都希望按时完成工资单应用。

一般考虑两条主要的调整目标：

使你的投资得到最大回报。把你的时间和精力用于解决很有可能产生最大的利益的问题。

使争用减到最小。瓶颈的特点在于延迟和等待；尽可能地消除或减少它们。

最后，考虑如下的通用数据库调整目标：

将需要存取的块的数量减到最小；必要时检查并重写代码。

只要可能的话，使用高速缓存、缓冲区和队列弥补机电缺点（内存比磁盘更快）；预取数据。

将数据传输率（用于读写数据的时间）减到最小；快速磁盘、RAID和并行操作有助于完成这项工作。

调度程序尽可能非竞争性地运行；它们可以并行，但大部分时间仍然是非竞争性的。

16.5 使用投资的回报

投资的回报（ROI）是一个用来查看性能调整过程的组织管理严密的、合算的方法。它帮

助你发现解决特定应用调整问题的最好方法。你已经看到 ROI作为一个主要的性能调整规则（优先次序）和作为两个主要的性能调整目标中的一个出现。现在你把它牢记在心并把它作为高级、逐步的方法用于性能调整。

你想按一定的顺序执行性能调整，希望这个顺序使你所花费的时间和精力获得最大的收益。在随后的 ROI策略中，注意到步骤 1~3 相当于逻辑与物理设计，这已在第 1 章与第 2 章中讨论过。逻辑设计是 DBMS 厂商独立进行的。因此，在处理逻辑设计时，没有 Oracle 的具体说明（逻辑设计工具不包含在内）。你将再次访问 16.6 节的应用类型并运用一些用于物理设计的具体的 Oracle 建议以及用于这些应用的性能调整建议。在后面的小节中，将更加仔细地研究这些 Oracle 的性能调整建议，查看调整应用，调整内存和调整输入 / 输出。

步骤 4 和 5 相当于调整应用，你将在第 17 章中遇到。步骤 6 和 7 相当于调整内存，它包含在第 18 章中。步骤 8 和 9 相当于调整输入 / 输出，它出现在第 19 章中。步骤 10 和 11 研究对于数据库来说比操作系统更外部的组件：网络与客户。步骤 12 提供了一些高级的和不太常用的解决方案，这些方案仅在已经使用了标准方法（步骤 1~11）但想得到的性能收益没有实现的情况下尝试使用。第 17、18 和 19 章包含 Oracle 的具体说明以帮助你完成正确的性能调整。

现在，按照投资回报的逆序详细地回顾这些步骤。

16.5.1 步骤1：进行正确的逻辑设计

像第 2 章中所讨论的那样，你可以减少存储容量。实际上，这经常意味着更多的表，其中每张表所包含的行数更少。这又意味着快速查找的能力。不考虑查找技术，存储的且必须完全查找的行越少，发现你要寻找的东西就越快。

16.5.2 步骤2：进行正确的物理设计

在第 3 章中，你看到一个通用的硬件布局。在稍后的 16.5.12 节中，你将看到一些对于此通用硬件布局的具体应用方法。

16.5.3 步骤3：如果必要，进行重新设计

假如在数据库生命周期的任何阶段，设计出现问题的话，在资源和管理允许的情况下，考虑重新设计它。有时不必进行大范围的调整纠正一个数据库的最好方法是重新分析与重新设计。例如，在没有初始设计、有一个仓促设计或多个数据库需要合并的情况下考虑重新设计。

16.5.4 步骤4：书写高效率的应用代码

一个程序仅能和形成它的算法运行的一样快。假如一些 SQL 代码使用效率低下的查询或排序例程，不管 Oracle 优化程序尽多大努力，应用程序仍然缓慢地运行着数据库。假如你投入一项工作中，代码已经编写但是写得很差，那么请考虑下一步，而不是试图每次修理这部分代码。

16.5.5 步骤5：如果必要，重写代码

假如应用代码效率低下并且资源和管理允许的话，那么就重新分析并重新编写更加有效

的代码。了解Oracle优化程序和用于更有效代码的Oracle SQL重写规则。数据库应用与任何一个普通的软件代码段没有区别，它可以被编写得很好也可以很差。在设计结束后，应用是一个获得投资回报的最大的调整机会。毫无疑问，时间和精力应该花费在这里。

16.5.6 步骤6：调整数据库的内存结构

在第6章中，你看到SGA包含了所有的可以调整的、共享的内存结构。当需要时，Oracle通过调整数据库缓冲区缓存提供真正的增长。这个高速缓存在读出或写入时缓存所有由Oracle存取的数据块。还有，共享池通过库缓存组件保存SQL代码并通过适当命名的数据字典缓存保存数据字典。更多的回滚由上述的数据库缓冲区缓存保存，但它不是独立可调的。重做日志缓冲区是SGA中分别定义的可调区域。所有这些内存结构所具有的充足的容量减少了争用的发生并弥补了机电的缺点。

16.5.7 步骤7：如果必要，调整操作系统内存结构

在Windows NT中，不一定像在诸如VMS和UNIX那样的大型多用户操作系统中那样要通过内存的使用进行调整。在UNIX中，已命名的Oracle用户（通常是“oracle”）一般必须有不受限制的外壳进程限制。这必须通过人工或在一个注册脚本中完成。UNIX交换区域充当一个操作系统临时存储区、用户临时存储区和操作系统虚拟内存后备存储区。因此，在UNIX中，交换区域可以成为一个瓶颈，尤其是在数据库服务器上。

在UNIX系统的Oracle数据库中，系统管理员和数据库管理员必须协同工作以便UNIX提供足够的共享内存和信号量给Oracle进程足够的空间去有效地运行。附录A和附录B进一步讨论这些重要的主题。

16.5.8 步骤8：调整数据库输入/输出

数据库输入/输出受到RDBMS和操作系统内存结构的影响。但是步骤8想要做的事情是通过重新定位数据库逻辑和物理结构调整输入/输出以减少争用。当然，数据库管理员应做到这一步，尤其是在他以正确的顺序执行这些步骤的情况下。

例如，如果你到达这个调整点，你就已经调整了数据库缓冲区缓存。现在你把注意力重新集中在物理设计上。也就是说，你进行更多的物理上的再设计工作，如果必要的话，把数据库输入/输出牢记在心。你的物理再设计比以前完成的更特别。你想要在所希望的方向上提高输入/输出时间评估，例如减少有繁忙插入的应用程序的背景时间。

16.5.9 步骤9：如果必要，调整操作系统输入/输出

操作系统通过所有的进程（包括诸如DBWR和LGWR的Oracle后台进程）完成所有的读写请求。操作系统缓冲这些请求，执行读写操作，然后在完成时进行确认并把数据返回给该进程。文件系统是包含所管理的文件的元数据的数据结构，例如每一个文件起始扇区的地址、扇区长度、目录树位置、属性（例如权限、大小和时间标记等等）以及其他信息。

在UNIX中，文件系统也有它们自己的逻辑块大小，它大于或等于物理块容量（512个字节），缺省值通常是8K字节。Oracle块容量至少是8K字节或8K字节的倍数，例如16K字节。

其他重要的操作系统以及Oracle输入/输出调整问题包括提前读功能、异步输入/输出、多

块读取、RAID数据条大小、磁盘几何结构、控制器和许多其他问题。假如数据库管理员不是系统管理员（事实经常是这样），那么他必须与系统管理员协调工作以适当地使 Oracle和操作系统输入/输出参数相结合。

16.5.10 步骤10：如果必要，调整网络

正如在16.2节中所讨论的那样，一个饱和的网络能够抵消数据库调整带来的改进。数据库管理员必须保证，如果必要的话，通过使用诸如网络监控软件的方法，他的客户 /服务器系统的整体应用性能不会因为网络负载或其他的网络小毛病而过度地受到损害。网络管理员和系统管理员一样必须和数据库管理员密切配合来解决这些问题。

16.5.11 步骤11：如果必要，调整客户端

局域网（LAN）管理员和通用网络管理员（他们有 WAN职责）对网络客户或PC的硬件和软件负责。这些管理员连同数据库管理员必须正确地决定客户的大小和配置客户，如果必要进行升级，在可接受的性能准则下实现客户 /服务器数据库应用程序的运行。例如，当 Oracle已经被调整并且可以在亚秒时间内服务于一条数据库查询（当通过网络从客户端收到时）时，无论从客户到服务器的网络延迟大约是几秒都没有什么关系，反之亦然。

16.5.12 步骤12：如果所有的调整都失败了，考虑更多的特殊解决方案

“特殊解决方案”包括Oracle的多线程服务器（MTS）、事务处理（TP）监控程序、Oracle的并行查询和其他的并行能力、Oracle聚簇、Oracle的位映射索引、MPP机器、固态盘、内存驻留（RAM）盘、硬件加速器和排队系统。

因为这些解决方案不适用后面的小节，所以本节简要地讲述了它们中的一部分。

1. Oracle的多线程服务器（MTS）

正如第3章中所讨论的那样，Oracle的多线程服务器是一个伪多线程解决方案，用以帮助对大量的并发用户或存取单个数据库的程序进行统计。多线程服务器本质上是 TP监控程序的低成本形式，其目标是实现大量的并发用户并对它们进行服务，因此使吞吐量（不是反应时间）达到最大。然而，多线程服务器尽可能地把进程全局区域（PGA）内存移动到SGA中，因此共享更多的内存并且减少维护每个进程的系统开销（它的状态信息）。

2. 事务处理（TP）监控程序

相比之下，事务处理监控程序走得更远，它不维护多线程服务器包含的整个进程状态信息和由普通操作系统或RDBMS所做的永久连接。每一个进程的系统开销进一步减少。操作系统和RDBMS的典型并发用户限制大约是200个用户。这意味着在大部分操作和数据库系统达到这一数字之后，系统性能急剧下降。这被称为系统失效（thrashing），它是由过度的多程序或用户导致的症状。

事务处理监控程序能够支持大约1000个并发用户，而在以前可能仅仅只有几百个用户。这就是为什么事务处理监控程序位于RDBMS的前面的原因。一般认为它们存在于后端（RDBMS）和前端（客户或PC）之间的中层。在本书编写时两个主要的商业产品是Novell/BEA公司的Tuxedo和IBM/TransArc公司的Encina。

3. Oracle的并行查询选项（PQO）

像在第3章中提及的那样，Oracle的并行查询（PQ）通过划分与克服规则可以加快查询的速度。然而，Oracle也允许并行索引创建、并行 SQL*Loader装入和并行导入/导出。所有这些功能通过几个因素减少了实际的背景时间，这对数据库管理员的工作来说大有好处。尽管后面的并行操作趋向于主要帮助数据库管理员，因为他完成（或将要完成）这些任务的大部分，但并行查询可以使任何用户受益。

4. Oracle聚簇

借助Oracle，你可以使用索引簇或哈希簇将经常在一起连接或存取的两个或多个表结合在一起。聚簇本质上是在物理层将子表合并到它们的父表中。换句话说，父表的主关键字不仅为某一个主键值存储唯一的行数据（和通常情况一样），而且存储由子表的行组成的重复组，这些子表通过它们的外键对值进行引用。因此，数据在逻辑层保持规范化，但是被认为在物理或存储层不是这样的。这完全可以接受并且没有回避或违背任何关系模型规则或标准化理论。簇可以被索引（B*树）或哈希处理。第19章将进一步讨论这个问题。

5. Oracle的位映射索引

Oracle还提供位映射索引，该索引一般与OLAP系统有关，但是经常与其他系统一起使用，尤其是DSS。位映射索引能很好地代替基于具有不同值的小子集列的常规索引（B*树）。它们比常规索引占用更少的空间并且可以高效地使用复杂的 where子句。因此，它们是DSS和OLAP系统的首选，因为它们有大量的数据和较高的分析要求。假如一个系统适度地或频繁地进行更新，那么位映射不是好的选择。

这些索引的工作方式是一列中的所有不同值由具有0和1的索引列进行转化，用每一行来表示那一列是否等于给定的特殊值。表16-1显示了一个叫做STUDENT的简单表，它有2列，分别是STUDENT_ID和SEX，并有3行。该表告诉你位映射索引如何工作。你将在第17章学习有关位映射使用和实现的知识。

表16-1 STUDENT表的位映射索引

STUDENT_ID	SEX	SEX= 'M'	SEX= 'F'
719250751	M	1	0
298674071	F	0	1
347691027	M	1	0

6. 大规模并行处理机（MPP）

大规模并行处理机和SMP机在第3章中涉及过。这两种机器类型都有多个处理器并且都提供了巨大的计算能力。除了结构上的不同，两种机器之间性能上的不同只是程度上的差别而不是性质上的差别。大规模并行处理机提供了更多的处理器（几百或上千），而SMP只提供了一百个或更少的处理器。中档和低档的SMP已随处可见，而高档的SMP和大规模并行处理机却仍未普及。

现代UNIX和UNIX变体的多处理器可以安全地超过大型机的性能。这对大多数数据库系统意味着增加了并行计算能力。这能够和诸如并行查询这样的操作在一起和谐地工作。不幸的是，大部分数据库系统不能利用所有的这些功能，因为它们可能受到输入/输出的限制而不是处理器的限制。更有可能的是，组成RDBMS内核的实际软件结构（例如多线程可伸缩性）仅能支持20个左右的处理器，超过这个限制后不能发挥巨大的性能作用。目前，在Oracle、

Sybase和Informix中的确是这样。

存在不同的硬件解决方案，有些基于相对过时的技术，而有些基于较新的技术。较老的方案中包括RAM盘。假如一个操作系统允许这种类型的排列，你可拥有一部分充当分离磁盘的核心（和虚拟）内存。当然，假如电源断电，除非内容拷贝到一个更为永久的存储器（即磁盘）中，否则该盘将丢失。有些DOS版本允许此项操作，对于小型的xBASE数据库（例如dBASE）可以工作得很好。UNIX仍然允许这种操作。特别是，Sun Solaris允许实际是UNIX RAM盘的临时文件系统的配置。

总之，这些盘的使用对数据库系统来说是有问题的，不仅仅会遇到硬件失效或电源丧失，你还会遇到一种情况：不管什么理由，无法使用RAM盘并且RDBMS必须对那张盘中的数据做时间标记。结果：数据库需要恢复！这是一个坏消息。然而，假如系统证明是比较可靠的，那么恰当地使用这张盘可以使性能有所提高。在Oracle中，一种可能方案是把TEMP存储在一张RAM盘上。

7. 固态盘

尽管固态盘的想法和原形已经存在了大约15年，但相对来说它还是比较新的。它们一直没有得到商业上的使用，直到最近5年情况才有所改变。硬件SMP厂商（例如Sun公司）和第三方的供应商目前提供这些盘。

固态盘实质是一块专用的永久内存卡。它的明显优点是消除了机电缺陷，因为有了要分解的磁盘存取时间。存取时间接近于内存速度。缺点是价格偏高、所有权/兼容性问题以及这些产品缺乏较长的产品周期。后面这个问题表明该技术仍然相对比较新并且还不可靠。

8. 硬件加速器

硬件加速器是专用的输入/输出卡，它提供了充足的高速缓存空间、专用的固件以及有时是专用的微处理器用来加速它们支持的输入/输出通道。它们已经比固态盘早几年投入商业使用，但是并不很成功。它们的价格较为合理，但也存在所有权/兼容性问题。因为它们不是替换固态盘而仅是扩充它们，所以它们的技术稳定性不如固态盘那么重要。总之，在合理配置后，它们已被证明相当可靠。Database Executor (DBE) 就是一个例子。

9. 排队系统

排队软件，例如IBM公司的消息队列中间件（Message Queuing Middleware，MQM）系列，能够提供异步消息传送能力。这种软件有时也称为面向消息的中间件（Message Oriented Middleware）或MOM。该软件所完成的工作基本上是队列所要做的；以先进先出方式（FIFO）缓冲某些东西。它有时替代或辅助TP监控程序以帮助应付在维护高并发系统中的大量用户时的系统开销。

除了FIFO缓冲以外，这种类型的软件经常可以保证把客户请求发送到服务器。因此，通过既缓冲了网络系统中的客户/服务器请求，又防止了对同一请求的不必要的二次传送，这种软件间接地加速了处理过程。

16.6 再看应用类型

第3章中涉及到3种主要的应用类型（OLTP、DSS和批量），还提及了一些较少使用的类型（OLAP、VCDB）。现在重新考虑Oracle对于这些不同应用类型的调整建议并看一些通用硬件布局上的新的、特殊应用方法。首先，对两种主要的应用类型OLTP和DSS进行对比。

16.6.1 OLTP问题

OLTP系统具有高的并发性（大量的交互式用户）并且是更新密集型的（含有大量的插入、更新和删除语句）。因为OLTP应用可以根据插入和删除而较大地增长或收缩，所以支持大部分易变的表和索引的表空间中的区间应被预先分配到它们最大的期望容量。在这种情况下，动态扩展仅损害性能。基本上，在达到那个最大值以前你就必须计划容量和执行查找。

你必须有足够的重做和回滚来处理事务的要求。假如事务相对短或者修改相对少量的数据，你需要许多较小的回滚段。因为你有许多并发用户，所以你需要许多回滚段来把争用减到最小。假如系统有紧急需要的话，你应有许多重做日志来处理频繁发生的检查点。回滚段和重做日志将被分隔并且可以使用 Oracle 的诊断工具来测定其大小（参见 16.8 节）。

与把这个逻辑放置在应用代码中相比，数据库检查约束和引用完整性约束就计算时间而论花费更少的系统开销。总之，确保应用代码尽可能地重复使用以促进共享。当此代码被多次调用时，Oracle 使用共享池的库缓存来重复使用它。对于 OLTP，通过使用绑定变量而不是直接量可以减少应用进程的系统开销。假如语句中有少量符号要分析，给出少量分析可以更好地共享语句。

此外，在 OLTP 系统中，需要更多的索引，因为你正试图用近乎随机的方式存取表中相对少量的数据。然而，你不想让索引太多以致于实际上降低了应用的速度。一条规则是，在你的所有主键和外键中建立索引并保守地增加其他东西。记住，更新和重构索引代价非常大，高级 DML（插入、更新和删除）活动触发这种索引重构的排序。

正如 16.5.12 节中提及的那样，当你的数据库系统在极端过载的情况下或需要以每日为基础时，可以使用多线程服务器（MTS）选项。尽管多线程服务器的主要目标是提高高度并发系统的吞吐量并且许多 OLTP 系统的并发度是可变的；然而，最大化吞吐量使反应时间延迟变长。所以使用多线程服务器是一种策略。也就是说，假如高吞吐量给你带来了高的性能，那么就使用它。假如你需要快的反应时间，就不使用它。

16.6.2 DSS问题

使用 DSS 时，数据库经常是大的、历史的和只读的。因此，查询（选择）是 DSS 活动的目标。Oracle 的并行查询能够加速缓慢查询的速度，这些缓慢查询可以通过一些物理（表空间和数据文件）重构并行执行。像以前所讨论的那样，希望 Oracle 块一次读取尽可能多的行。因为 DSS 查询一般触发本质上连续的整个表扫描，所以希望通过读取多个连续的块使你的块读取达到最大值。把 DB_BLOCK_SIZE 和 DB_FILE_MULTIBLOCK_READ_COUNT 参数设置得尽可能高些。这些参数设置将在第 19 章中进一步解释。

然而，这里有一些警告：

确保你的区间大小是 DB_FILE_MULTIBLOCK_READ_COUNT 的倍数。

确保 DB_FILE_MULTIBLOCK_READ_COUNT 是 DB_BLOCK_SIZE 的倍数。

假如你正在使用 RAID，确保你的 RAID 数据条大小是区间大小的倍数并且上述的两条仍然满足。

你有 3 种方法使用 DSS 索引。因为大部分查询返回大表的全表扫描，你也许想知道为什么要使用 DSS 索引（事实上，方法 1 中并不使用它们）。3 种方法如下：

方法 1：根本不使用任何索引，尤其是在你没有任何高度选择性查询或买不起用于在每

张表中创建常规索引的辅助存储器的情况下。

方法2：用一个小的DSS查询率保持那些具有选择性的表中的少量索引。

方法3：像16.5.12节中所讨论的那样，使用位映射索引。因为大部分时间里，DSS查询并不趋向于高度的选择性而且即使在常规索引存在的情况下也将触发整个表搜索，所以把位映射索引作为一种可供选择的方法。

当然，在实际中数据库管理员会不时地使用所有这些方法。

不要在DSS应用代码中使用绑定变量。这直接与OLTP的建议形成对照，因为在OLTP下，你希望最小化应用进程开销（语法分析）。然而，随着相对长时间地运行DSS查询，语法分析占用整个查询时间的比例会更小。你要保证优化程序选择最好的存取计划。假如使用绑定变量，优化程序不能调用它所存储的统计信息（通过ANALYZE命令）以选出存取数据的最好方法。任何本应当通过直接量获得的选择性将不存在。这些直接量有助于减少中间表和返回的最后输出的规模。

在OLTP下，希望有许多相对较小的回滚段。伴随着DSS和长时间运行的查询，类似于长时间运行的批量生产系统，应有一到两个大的回滚段。这仅仅实现一致性读取和大量回滚数据的并发视图。假如DSS是100%只读，这不会成为一个问题。实际中，我发现没有100%的只读系统。

16.6.3 OLTP与DSS都需要考虑的其他问题

OLTP与DSS都需要附加考虑的问题是Oracle的块级参数INITRANS和PCTFREE。对于OLTP，因为每个块必须要有增长和变化的空间，所以把PCTFREE设置得相对低些（或仅仅把它设置为缺省值）。对于大多数的只读DSS，把PCTFREE设置得高些，因为块中的数据通常保持不变。对于OLTP，希望把INITRANS设置得相对高些，因为多数并发用户会触发动态事务头槽扩展，随之而来的是块级重构，这可能逐步影响到区间级。

对于DSS，不太可能出现由于锁定而必须等待的并发用户，因为大部分事务是选择，更不用说DSS存储的大量数据块了。因此，假如你的DSS也支持适当级别的插入、更新或删除的话，那么把INITRANS设置为中等水平。另外，它也可以保持缺省值。

16.7 理解基准测试

基准测试（benchmark）是代表一个应用实际工作的准则。在第3章中提到的作为事务分析结果的数量测量也可以用作基准测试。在基准测试上运行的应用的性能通过一些专门选择的数量测量方法来测定。当基准测试运行时，你通常已经预先设置了条件或配置以便当在运行中的某个时刻改变一个变量时（独立变量），你可以保留常量（初始条件或从属变量）。这使你能够分别研究这些变量的影响。因此，基准测试仅是科学或实验方法的一个变种。假如正确处理的话，你可以对原因和效果，或者最起码是效果的相关性进行评定。假设是改变从属变量（例如，块大小）将影响到独立变量（例如，吞吐量）。

基准测试可以提供各种便利。通常，基准测试与仿真有关。系统仿真是对系统的实际工作进行模拟的功能原型，用来推算系统的性能或测试系统的功能。然而，基准测试也提供运行基线的能力，基线是你在发布一个产品系统之后或开始性能调整以前所采取的相关数量基准。它给数据库管理员一个用于比较的基础，用来测验他或她的性能调整是否已经带来了改进。

最著名的数据库系统基准测试是由事务处理性能委员会（TPC）提供的。这个委员会已经制作了大量的基准测试，它创建的第一个基准测试是具有代表性的银行出纳员操作（TPC-A与TPC-B）。对于这些基准测试，要创建一个小的数据库，包括一张银行表、一张转帐表和一张帐目表。所执行的操作包括对这些表的更新和对历史记录表的插入。性能由每秒钟的事务量（tps）来测定，每秒钟的事务量是吞吐量的一种数量度量。因为TPC-A与TPC-B在几年之前就已经发布，因此事务处理性能委员会已经推出较新的基准测试TPC-C与TPC-D，TPC-C与TPC-D针对不同种类的应用进行基准测试。

TPC-A（1989）使用一个相对简单的、多任务的事务对一个具有高并发性和重要的输入/输出的OLTP环境进行测试（执行不止一个的选择、插入、修改或删除的事务被称为是多任务的）。TPC使用事务类型（transaction type）这个术语来指代一个特殊的多任务事务。在TPC-A中，事务有多个更新和一个插入。

TPC-B（1990）除了取消了高并发性需求以外，基本上与TPC-A相同。它被用作数据库耐压测试（stress test），基本上仅测试重要的输入/输出。然而当使用仿真软件中的虚拟终端时，这个差别是模糊不清的，所以这两个基准测试经常被放置在一起或彼此替代。

TPC-C（1992）是一个OLTP基准测试，它是5个不同的、更复杂的事务类型和9个数据库实体的混合。比较起来，TPC-A和TPC-B使用单一的事务类型和4个数据库实体。其业务目前是基于具有一个定单记录系统的供应商数据库。

TPC-D（1995）是一个DSS基准测试，主要用于繁重的、现实领域的事务。它包括17个针对大量数据的复杂查询。这是目前最新的TPC基准测试。与较轻负载的TPC-A和TPC-B以及相对中等负载的TPC-C完全不同的是，TPC-D适合于长时间运行的重负载查询。

当基准测试在一个无偏的独立环境下执行时，这些基准测试提供了一个在RDBMS软件厂商之间进行比较的基础。数据库系统可以在已调整和未调整的情况下比较。应用是相同的，如果网络 and 平台（硬件和操作系统）是相同的话，那么剩余的唯一可变物是RDBMS软件。通常没有两个厂商会在相同的平台上进行相同的基准测试（至少在同一时期内）。

使用好的仿真软件（例如LoadRunner）可以产生虚拟用户和进程来模拟在基准测试期间数据库系统的负载。并发用户的数量可逐渐增加，同样每个用户的事务数也可以增加。总之，用这两种方法，事务的总数都会增加，你可以有效地创建并观察数据库耐压测试。当确认了运行并且收集了统计数字（tps）后，你可以研究ROI策略并在必要时使用性能调整规则来解决常见的调整问题，即使是在一个仿真系统中也要这样做。

16.8 使用Oracle诊断工具

主要的Oracle诊断工具如下所示：

SQL_TRACE与TKPROF。

EXPLAIN PLAN。

使用V\$动态性能视图和服务器管理器行方式。

服务器管理器监控（GUI）。

企业管理器的性能包。

utlbstat/utlestat与report.txt。

第三方产品。

16.8.1 使用SQL_TRACE与TKPROF

要调整应用，你可以使用 SQL_TRACE与TKPROF。为了得到定时信息，你必须设置 init.ora参数TIMED_STATISTICS=TRUE。时间可以定到0.01秒。实时数据库系统必须使用代替的定时方法，例如编写你自己的定时器或者用数据库系统内部的实时操作系统时钟采样。你可以随意地将 USER_DUMP_DEST设置为你所选择的目录，也可以将 MAX_DUMP_FILE_SIZE设置为你希望跟踪文件增长到的最大字节数。既可以在系统层（对于所有的会话）通过 init.ora参数设置 SQL_TRACE=TRUE，也可以在会话层通过 ALTER SESSION SET SQL_TRACE=TRUE；设置SQL_TRACE=TRUE。

执行这个应用，如果需要，把 SQL_TRACE设置为FALSE。实际上，在短期不会为了计时的目的而再次运行这个应用。使用 TKPROF格式化跟踪文件（从命令行运行 TKPROF工具）。收集到的统计数字包括查询执行周期的组成时间（语法分析、执行和获取数据）以及逻辑与物理读取。SQL_TRACE和TKPROF的使用、语法和解释将在第 17章中详细讨论。

16.8.2 使用EXPLAIN PLAN

另一个应用调整工具是 EXPLAIN PLAN。你必须运行 utlxpln.sql来创建PLAN_TABLE，或者你自己交互式地创建它。不用实际执行 SQL语句，EXPLAIN PLAN FOR SQL语句；就能解释该 SQL语句的执行计划（这类似于其他翻译语言的 NO EXEC选项）。不管有没有 SQL_TRACE和TKPROF，你都可以使用 EXPLAIN PLAN。我建议你先应该检查一下代码（把它读一遍），接下来运行SQL_TRACE与TKPROF，最后假如你仍然对运行较差的应用感到有困难，那么使用 EXPLAIN PLAN。连同Oracle优化程序是如何工作的，这些内容可以在第 17章中找到，对Oracle优化程序的介绍有助于解释 EXPLAIN PLAN是如何工作的。

16.8.3 使用V\$动态性能视图

要有助于调整内存（参见第 18章）与调整输入/输出（参见第 19章），所有的 Oracle产品（例如SQL*DBA，服务器管理器与企业管理器）都依靠 V\$动态性能视图。这些视图被划分为实例、数据库、内存、磁盘、用户、会话和争用几个方面。它们基于内部 X\$库基础表。DESCRIBE V\$FIXED_TABLE并从V\$FIXED_TABLE表中选取你想要的列以得到 V\$视图的列表。一般不直接查询X\$表。它们的名称和内容在不同版本之间以及同一版本内是不同的。

V\$视图之所以被称为动态的是因为它们实例启动时被填充，在实例关闭时被截取。V\$视图（和X\$表）还形成标准Oracle调整脚本utlbstat/utlestat的基础，utlbstat/utlestat使用SQL脚本查询V\$视图并格式化返回的输出值。因此，假如 utlbstat/utlestat没有为你提供你所需要的东西，你可以使用服务器管理器与 V\$视图辅助或者取代那些工具。所有的 Oracle平台，不管是老的还是新的，它们的共同特性是 SQL脚本、V\$视图以及SQL*DBA或服务器管理器行方式（例如，在UNIX上是svrmgr1，在NT上是svrmgr23.exe）。这意味着假如你已经开发了你自己的性能脚本，你可以在这些命令行工具中运行它们。除此以外，还有 SQL*Plus。

16.8.4 使用服务器管理器监控器

由于企业管理器的出现，服务器管理器监控器正变得越来越过时。它提供了与性能有关

的几个屏幕。这些屏幕监控多线程服务器、逻辑与物理输入 / 输出、处理信息、锁与闩、共享池信息和回滚信息。屏幕被实时更新，并且它们基于 V\$视图与X\$表。

16.8.5 使用企业管理器性能包

企业管理器性能包极为有用。它提供了几个组件，这些组件可以节约命令行方法与脚本的很多时间，对有经验的数据库管理员或那些准确知道所要查找的东西的人来说更是如此。企业管理器使你能够很快地接触到那条信息。统计数字基于 V\$视图和X\$表，但是被重新格式化为更容易理解的GUI表格形式。

性能包中的组件有助于分析你的逻辑与物理设计。它们还通过跟踪监控锁、各种性能问题（吞吐量、重做、回滚、输入 / 输出、内存等等）、关于资源消耗的顶端用户会话、表空间存储（数据文件、存储碎片等）以及应用事件。

16.8.6 使用utlbstat/utlestat与reportl.txt

到目前为止最常使用的 Oracle 诊断工具是utlbstat/utlestat对。一个数据库管理员在运行他的应用或仿真以前先要运行 utlbstat。utlbstat.sql脚本构造了必要的开始表用以收集和存储性能数据。然后数据库管理员运行 utlestat.sql，utlestat.sql构造了结束表和差别表用以计算 utlbstat 运行与 utlestat 运行之间（实际上是应用持续时间）的性能差异，格式化输出数据（包括注释和一些解释）并把它写入缺省文件 report.txt中。该文件必须由数据库管理员直接或间接地解释（通过采用一些给定的输出值并把它们作为简单公式的输入使用）。

数据的解释意味着把这些最终数值与建立的准则作比较，把 ROI策略牢记在心，对于那个给定的性能区，把发现的结果分类为可接受的或不可接受的。report.txt中的输出几乎直接或间接地涉及到所有的基础。所以一个数据库管理员只须应用 ROI策略、已建立的准则以及他的应用知识并确定是否可以接受内存性能和输入 / 输出性能。

16.8.7 使用第三方产品

本章对主要的Oracle诊断工具做了概述。目前还有一些第三方性能监控工具并且在已建立的Oracle顾客群中它们占用相当大的市场份额。这些工具包括来自 BMC、Platinum和 Compuware公司的产品。你将使用utlbstat/utlestat和Report.txt作为第18章和第19章的诊断工具。你将在下一章中使用EXPLAIN PLAN 和伴随TKPROF的SQL_TRACE。