

## 第39章 并行服务器管理

本章要点：

- 理解并行服务器的优点
- 单实例与并行服务器数据库的比较
- 决定何时并行服务器能够解决一个商务需要
- 为失效设计一个并行数据库
- 为可伸缩性设计一个并行数据库
- 并行服务器创建的特殊考虑
- 怎样监控与调整并行服务器
- 附加提示与注意

### 39.1 理解并行服务器的优点

Oracle并行服务器用于两种基本的应用类型：那些需要高可用性与那些可以与底层硬件一起移植的应用。为了让并行实例运行，需要比单实例数据库更多的组件。本章将介绍这些附加的组件都是什么，及怎样使用它们管理你的并行服务器数据库。

并行服务器是一个选项，允许两个或更多的实例同时装配在相同的数据库上。使用多实例，可以增加数据库资源的数量或净值数量。这些资源包括：

- 系统全局区，包括共享池空间与数据库调整缓存。

- 后台进程如数据库写（DBWR）、日志写（LGWR）与并行处理伺服器。

- 用户进程。

- SQL\*net或Net8监听器。

图39-1显示了这些资源在并行实例中是怎样成倍增长的。如你可以想像的，这些增长允许数据库为更多的用户处理更多的工作，这也是多数人归因于并行服务器的最主要的好处。只要你使用了长时期伸缩性机制，当你增加更多的硬件结点时，你也可以启动更多的实例。

**提示** 并行服务器可以帮助增加你正在做的任务数，而对相同数量的用户，不会增加每项工作的持续时间或减少同步工作的总时间。如果工作可以在可用的实例之间分散以避免对可用资源的竞争，就可以实现这个目标。

可以使用不同的初始化参数配置在并行服务器数据库上运行的各种实例。这使你可以根据你的用户需求，为进行繁重事务处理的用户调整一些实例，以最好地支持批报表程序并优化联机决策支持活动。如果实例运行在具有不同能力的结点上，如内存大小、CPU数量或CPU能力，也需要分别配置它们。

另外，附加的实例可以为用户与数据库管理员提供安全覆盖。有许多事件（有计划的与无计划的），会导致Oracle实例停止运行。在单实例数据库中，一个停止的实例对用户及他们的应用关闭了所有对数据库的存取。在并行实例中，运行在一个停止实例上的用户可以重新连接到一个仍在运行的实例上。对Oracle 8的用户来说，如果应用使用了特定的Oracle调用接

口 (OCI) 调用, Net8 配置文件具有正确的条目, 这个重新连接可以自动实现。可以设计你的系统使用具有一个或多个富余实例的并行服务器, 以便在主实例不可获取时支持你的所有用户。

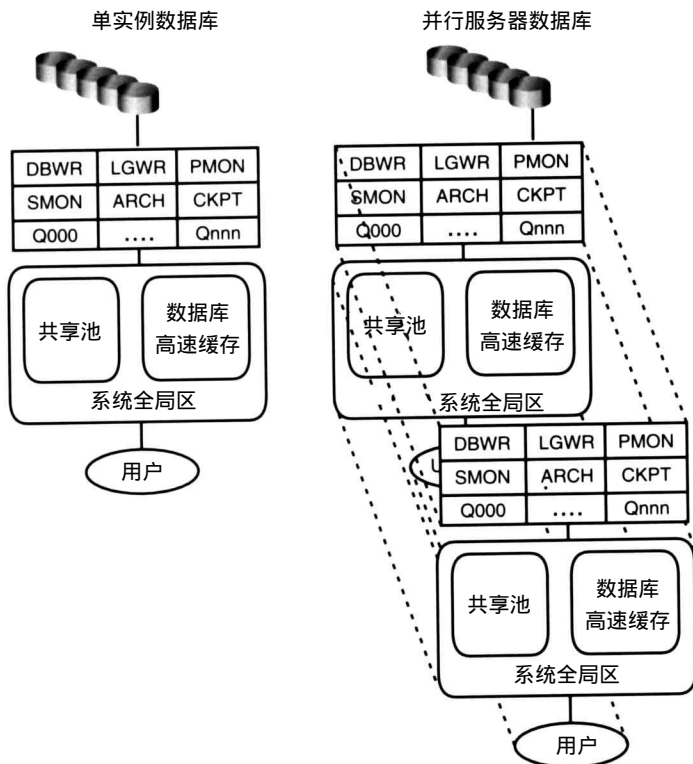


图39-1 单实例与并行服务器实例和资源

如果硬件可以支持足够的结点, 可以在相同的数据库上使用并行服务器的伸缩特性及故障恢复能力。只需简单地在—个或多个结点上保留实例, 用于故障恢复的情况, 而不用为性能的提升而同时运行它们。在设计多实例数据库时, 在这两个功能之间进行明确的区分是非常重要的。将工作转移到因为故障恢复的原因才保留的实例上, 在失败实例的用户需要使用它时, 将会导致竞争与性能问题。

## 39.2 单实例与并行服务器数据库的比较

为完全理解并行数据库的组织结构, 需要很好掌握数据库与实例的区别。数据库是由存储在硬盘上的一套文件组成的, 包括数据文件、重做日志文件及控制文件; 实例包括一套内存结构与进程, 主要的结构称为系统、或共享全局区 (SGA), 需要的后台进程是数据库写 (DBWR) 进程、重做日志写 (LGWR) 进程、系统监控 (SMON) 进程与进程监控 (PMON) 进程。如果运行 Oracle 8 (版本 8.0 或 8.1), CKPT 进程也是一个强制性的后台进程, 还有许多可以通过设置初始化文件中的参数启动的选择性进程。

Oracle 并行服务器需要附加的后台进程。其中一些由初始化参数启动, 另外一些自动启动, 与非并行实例的强制性进程一样。需要哪个进程, 可以使用初始化参数控制哪个进程, 依赖

于正运行的 Oracle 的版本,但至少总是需要一个负责管理实例之间封锁的锁 ( LCK ) 进程。在本章稍后,将看到 Oracle 并行服务器版本与控制特定后台进程参数之间不同的细节。

提示 需要做好准备修改单实例数据库使用的 init.ora 参数,以使并行服务器使用的多个实例可以成功地相互作用。一些参数在每个实例中都是相同的,另一些需要映射每个单独的实例的使用。

还需要知道如何在并行方式下启动一个实例。一个通常的排它的 Oracle 实例在启动时锁定控制文件,禁止任何其他的实例使用。为设置这个锁,实例必须以并行方式或共享方式打开。在 7.3 版本中,通过在命令行状态的 STARTUP 语句中包括 PARALLEL 或 SHARED 关键字,或在 GUI 数据库管理工具的实例管理屏上使用 SHARED 单选按钮达到这个目的。在 Oracle 8 中,需要将初始化文件中的 PARALLEL\_SERVER 设为 TRUE。

注意 实例管理 GUI 屏幕是 Oracle 的企业管理器产品的一部分,只能运行在 NT 平台上。也存在专门的并行服务器管理 GUI 工具,但它们也被限制在 NT 平台上使用。然而,使用其他操作系统的数据库也可以使用这些工具,只要为数据库服务器将 NT 平台配置为客户端。

### 39.2.1 厂商接口

并行服务器使用两个或更多的并发实例运行一个单独的数据库。硬件必须支持这种配置:允许独立的结点共享存储有数据库的硬盘。许多厂商在 UNIX 簇中支持多个结点,有一些早已提供了 NT 下的两个或四个结点簇。一些厂商支持使用专有操作系统的簇。在所有情况下,厂商必须提供软件去管理簇及对共享硬盘的存取。这可能是专有操作系统的一部分,或是一个或多个另外的产品。Oracle 保证并行服务器可以运行在可以成功与这些软件交互的平台上。有些情况下,Oracle 会限制并行服务器可以运行的结点的数量,即使厂商可以配置更多的结点。

为达到不同的实例可以相互交谈的目的,必须使用硬件的结点之间的连接与磁盘管理软件。Oracle 代码与厂商代码之间的接口随时间改变。在 7.3 版本中,Oracle 需要硬件厂商提供分布锁管理器 ( Distributed Lock Manager, DLM ), 提供 Oracle 在不同结点的实例之间传递消息的机制,这种体系结构如图 39-2 所示。有些情况下,Oracle 与厂商紧密配合工作为 DLM 编写所需要的代码,其他早已具有簇管理软件的厂商可以与 Oracle 交谈而不需要进行任何修改。在 NT 平台上,没有发布 7.3 版本的并行服务器选项,直至 Oracle 8 产品在它的开发中加入了这个选项。因此,为避免为新的版本重写代码,并行服务器的 NT 版本是基于 Oracle 8 体系结构的。

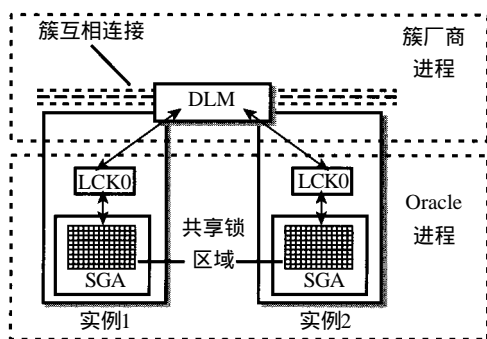


图39-2 7.3版并行服务器与DLM的接口

在 Oracle 8, 8.0 版本中, Oracle 将 DLM 集成为它自己的代码。集成的 DLM ( IDLM ) 通过统称为组成员服务 ( Group Membership Services GMS ) 的一系列服务与厂商软件对话。GMS 负责管理簇、处理共享硬盘存取、代表 IDLM 在不同的结点之间传递消息。一些厂商甚至支持启动与关闭接口以便 Oracle 可以随之在每个结点上启动与关闭 GMS。8.0 版本配置的一个图表

如图39-3所示。

为改进Oracle的性能与能力以指明并诊断问题，设计师们在开发 Oracle8i 8.1版本的并行服务器中决定将大多数GMS函数合并入内核中。这需要对后台进程、LMON、对IDLM的监控进程增加代码。没有合并入IDLM的GMS执行的功能仍作为簇组服务（Cluster Group Services, CGS），由簇厂商提供。这个改进减少了 Oracle对厂商代码的依靠，簇实现只暴露极小部分，同时提供更无缝的产品用于管理。图 39-4显示了Oracle 8i 8.1版本的一张图表，簇存取的系统组件。

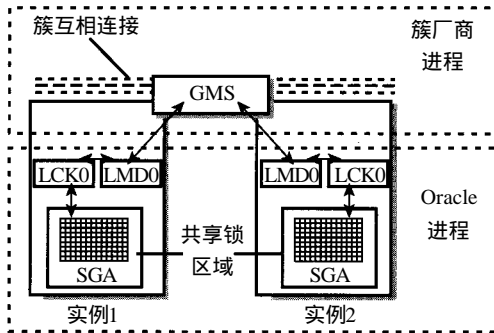


图39-3 8.0并行服务器与GMS接口

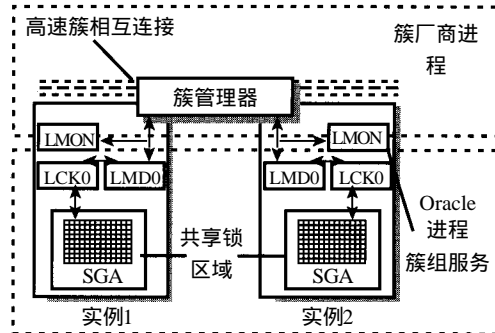


图39-4 8.1版本并行服务器与簇组服务的接口

不管正使用哪个版本，Oracle/厂商接口的详细资源依赖于厂商的体系结构。需要阅读与特定硬件、软件及Oracle版本相关的Oracle并行服务器安装手册，以找出这个配置的详细内容。也许必须运行一些特殊的厂商提供的代码，并在初始化文件中设置参数以配置特定版本所需要的后台工作。

如果运行在UNIX或NT操作系统上，当计划并行服务器时，必须考虑的最后一点是在这些平台上，必须将数据库文件旋转在裸盘分区上以便它们可以被多个实例成功地共享，包括数据文件、控制文件、联机重做日志文件。有些情况下，这非常有利，因为在硬盘写方面有一些性能改进，但对系统管理员管理而言，裸盘通常更难于管理。除了用于系统的操作系统文档之外，还可以在Oracle安装与配置指南中为特定的数据库版本找到裸盘的细节。

### 39.2.2 使用并行缓存管理锁进程

并行服务器使用并行缓存管理（Parallel Cache Management, PCM）控制不同实例对数据库块的存取。实例只有在以正确的状态拥有必要的PCM锁时才能存取一个块。为了使用INSERT、UPDATE或DELETE操作改变一个块的内容，实例必须以排它方式保持这个块的PCM锁。要查询一个块的内容，实例以共享方式保存必需的PCM锁。PCM锁根据基本需要在实例之间传递，从模式名中你可能会猜出，排它模式一次只能由一个实例拥有，而共享锁可以由多个实例同时拥有。PCM锁与行级锁是完全独立的，Oracle使用与单实例数据库相同的方式继续使用行级锁以确保在某一时间只有一个用户可以改变一行的内容。

每个并行实例控制一个LCK进程以追踪该实例PCM锁的状态，并在实例与DLM之间传达锁请求。就每个实例来说，每个PCM锁都有一个状态：

NULL 当锁没有使用时。

SHARED 当锁以共享模式持有时。

**EXCLUSIVE** 当实例当前拥有这个锁只为它自己单独使用时。

每个锁的状态存储在 SGA 区中的一个特定区域，称为共享锁区域。当用户需要一个锁，而他当前没有拥有这个锁，或是要求以排它模式拥有而当前以共享模式拥有这个锁时，LCK 进程负责将锁状态改变的请求送给 DLM。与此相似，当 DLM 接收到锁状态改变的请求时，它发出一个请求放弃实例上的 LCK 进程的锁，那里当前持有 PCM 锁。

DLM 负责所有实例的锁的全局分配，并管理不同实例发出的锁的管理请示。DLM 可以分配没有使用的锁或从另一个实例重新调用锁，并将它们传递给发出要求的实例。DLM 还维护所有锁的当前状态，不管它们是否已被分配给一个实例。在 Oracle8 以前，DLM 是由操作系统厂商或 Oracle 提供的，是与平台有关的。因而，你用于启动或停止 DLM 的命令以及需要配置它的任何文件或参数在不同的平台互不相同。一个 Oracle DLM 配置文件示例如清单 39-1 所示。

清单 39-1 DLM 配置文件示例

					SERVICE
#NODE NAME	IP ADDRESS	NODE ID	DOMAIN	PORT NUMBER	
apple	123.45.678.001	1	0	1544	
orange	123.45.678.002	2	0	1544	
pear	123.45.678.003	3	0	1544	

无论你正以分离的 DLM 使用 Oracle 7 或以 IDLM 使用 Oracle 8，都需要查询与平台有关的 Oracle 安装/配置指南获得特定的 DLM 配置信息。在 Oracle 7 中，这个选项趋向于依赖平台，因为 DLM 是由厂商提供的，而 IDLM 完全由初始文件中的参数配置。由 DLM、LMD0 与 LM0N 使用的两个后台进程接大小排列时需要这两个参数。

向数据库数据文件分配 PCM 锁是多实例数据库系统管理员的主要任务。由于共享锁区域与 DLM 结构需要内存，过多的锁会导致性能退化。如果没有分配足够的锁，会导致大量的块碰撞询问 (pinging)。

#### 确定并阻止过量的块碰撞询问

当一个实例需要一个 PCM 锁，而这个 PCM 锁当前正由另一个实例以排它模式持有时，就会发生块碰撞询问。在释放这个锁之前，拥有这个锁的实例唤醒 DBWR 进程，检查由这个锁覆盖的块。DBWR 查看所有含有改变但还没有向硬盘刷新的块，并将它们写回到数据库文件。这个写脏块的进程作为 PCM 锁向下变化的部分称作碰撞询问。当增加由一个单独的锁覆盖的块的数量时，也增加了另一个实例可能需要这个锁而导致更多块碰撞询问的潜在可能性。

在碰撞询问的另一端是需要这个锁的实例。当接收到这个锁时，这个实例必须将它需要的数据块都读入缓冲器缓存，不能依赖于内存中早已存在的这些块的映射，因为其他的实例在它持有排它锁时可能已修改了这个块。如果接收实例需要在硬盘询问所有的块，这些询问被认为是真实的碰撞询问，但是，如果接收的实例不需要或只需要询问到硬盘的块，不必要的块写操作被认为是假的碰撞询问。参见图 39-5 查看这些不同类型的碰撞询问。

当实例需要定期存取确实相同的块时，真的碰撞询问可能会是无法避免的。假的碰撞询问组成不必要的额外开销，需要被检测出来并减少它们。你所经历的碰撞询问与假碰撞询问的总数依赖于数据库设计及初始化文件中 PCM 锁参数的设置。

**注意** 碰撞询问并不是坏事，而是并行服务器体系结构的一个必要的组件。过量的碰撞询问是一个问题，因为它降低了性能，本章讨论怎样设计一个维护可接受级别碰撞询问的数据库。

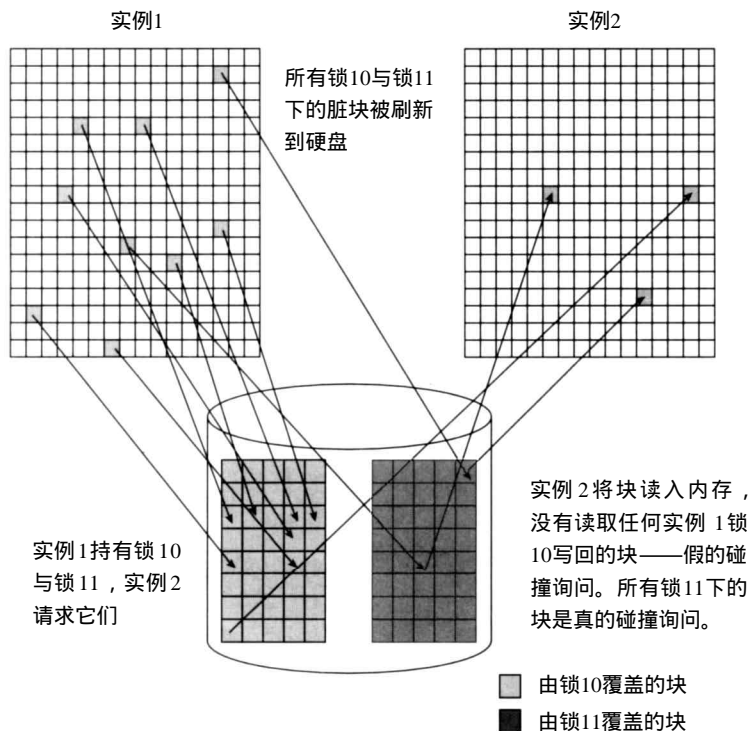
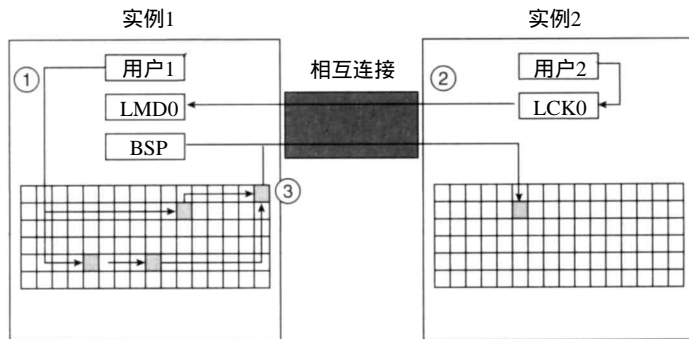


图39-5 碰撞询问在实例之间是怎样发生的

对那些正使用或计划使用 Oracle 8i，8.1版本的人而言，一个新的特性——高速缓存融合（Cache Fusion），阶段I将帮助减少pinging。在Oracle 8i，8.1版本中，DLM接受PCM共享锁的请求，并可以容易地构建请求的实例需要的读一致性块映象，它可以避免碰撞询问。与其说是将另一个实例要读的块写回硬盘，倒不如说是拥有排它 PCM锁的实例将使用它的块服务进程（BSP）后台进程建立所需要的块映象。它将使用任何所需的信息建立与请求实例与 IDLM 联系时刻一致的块映象。BSP会将这个块映象通过调整结点相互连接直接传送给请求的实例。这不只避免了资源仪器的硬盘读写，而且块映象拷贝可以不需要 IDLM改变原有实例上锁的排



实例1上的用户1在一个排它PCM锁下改变一个块及相关的回滚块  
用户2请求块用于实例2的查询，实例2上的LCK0进程联系LMD0进程将PCM降级为共享状态。  
代替改变锁，BSP进程在实例1的内存调整缓存中建立了一个重做块，使用所有必要的回滚信息，  
并通过相互连接直接将整个块发送给实例1。

图39-6 高速缓存融合，阶段I怎样避免碰撞询问。

它状态就被传递。图 39-6 显示了高速缓存融合怎样在 Oracle 8i，8.1 版本上工作。

### 39.2.3 使用并行缓存管理锁参数

使用一套全局高速缓存管理参数规定实例的 PCM 锁的环境。全局高速缓存管理参数名以字符串 GC 为前缀。反映先前讨论过的每个 Oracle 并行服务器版本中体系结构的改变，这些参数在不同版本之间略有不同。表 39-1 列举了参数名与描述信息并指明它们在哪个版本中是合法的。

表 39-1 PCM 锁管理参数

参 数 名	描 述	7.3	8.0	8.1
GC_DB_LOCKS	哈希 PCM 锁的数量	X(3)		
GC_DEFER_TIME	在响应 ping 请求之前的等待时间 (1/100 秒)	X(3)	X(4)	
GC_FILES_TO_LOCKS	对数据文件分配 PCM 锁	X(3)	X(4)	X(5)
GC_FREELIST GROUPS	空闲列表组标题块的哈希锁的数量	X(3)		
GC_LATCHES	为门管理保留的锁的数量 (只在 8.0.4 版本)	X(3)		
GC_LCK_PROCS	锁进程的数量 (LCK0, LCK1, 等等)	X(3)	X(4)	
	Start			
GC_RELEASABLE LOCKS	好的精细 PCM 锁的数量	X(3)	X(4)	X(5)
GC_ROLLBACK_LOCKS	用于回滚段的锁的数量 (只用于版本 8.0 以前的标题块的哈希锁)	X(3)	X(4)	X(5)
GC_ROLLBACK_SEGMENTS	回滚段重做块的哈希锁的数量	X(3)		
GC_SAVE ROLLBACK LOCKS	延迟回滚段块的哈希锁的数量	X(3)		
GC_SEGMENTS	用于非回滚段标题块的哈希锁的数量	X(3)		
GC_TABLESPACES	用于 ALTER ONLINE/OFFLINE 表空间命令的哈希锁的数量	X(3)		

可以将这些参数中的大部分设为它们的缺省值，如许多 DBA 所做的那样。当然，需要监控数据库性能以确保没有创建不必要的开销，这些开销可以通过改变这些值避免。并行服务器最严重的问题趋向于由于选择不合适的 GC\_RELEASABLE\_LOCKS 与 GC\_FILES\_TO\_LOCKS 参数值导致的过量的 ping 与锁变换。而且，每个锁在 DLM 中消耗空间，同时也在每个实例的 SGA 区内的共享锁区域消耗空间。如果定义了过多的锁，可以使用用于 PCM 锁管理的内存，这是其他数据库操作需要竞争的区域。

如在表 39-1 中所见的，有两种类型的 PCM 锁：哈希锁与精细锁，也称为数据块地址 (dba) 锁。这两种类型的锁有一个重要的不同之处，当一个哈希 PCM 锁是一个实例必需的时，它并不将它释放回 DLM，除非它是另一个实例必需的。因此，一个实例很有可能含有一个哈希锁并直至实例关闭之前从不释放它。然而，一个精细锁只在实例使用该锁管理的块时才由该实例持有，在此之后，锁返回给 DLM 供另一个实例甚至是同一个实例使用 (如果需要)。

通常，对查询集中的应用应该使用哈希锁，因为实例可以共享模式获得相同的锁。只要块没有被任何实例更新，这些锁便被每个执行查询的实例不确定地持有。可以定义这些哈希锁跨越许多块 (这减少了锁的总数)，因而减少了这个实例需要的内存。如果也定义了锁覆盖连续的块，将通过读相邻的一套块改进全表扫描的性能。在事务密集型数据库中，应该考虑使用精细锁，这些锁的每一个覆盖一个单独的块或可能是很少的一些相关块。因而 ping 的可能性，特别是假的 ping 的可能性，明显地降低。

**警告** 不应该自动使用精细锁，除非预期在实例之间会有严重的 PCM 锁竞争。精细锁产

生附加的消耗，因为它们在实例完成相关块后自动释放，即使另一个实例不需要它们。

当然，如果为了故障转移的目的在两结点系统中使用并行服务器，其中在任何时间只有一个实例会真正含有活动的用户，可以使用最小数量的哈希锁，而不管数据库活动的类型。锁在用于故障转移的目的而单独使用的并行环境下，不会提供它们通常的实例之间的功能。

怎样设置GC\_RELEASABLE\_LOCKS与GC\_FILES\_TO\_LOCKS参数的值以达到要求的PCM锁分布呢？首先，应该清楚在 Oracle 7中，缺省的状态与 Oracle 8和Oracle 8i略有不同，在Oracle 7，7.3版本中，缺省的机制是哈希锁，参数 GC\_DB\_LOCKS决定文件可以获得多少个哈希锁，这些文件上没有使用 GC\_FILES\_TO\_LOCKS参数分配的PCM锁。这个参数中没有列出的文件将使未分配的锁在它们之间平均分配，图 39-7显示了一个只有10个哈希锁的数据库配置——在该例之外似乎没有这样的数据库存在，将这 10个锁由缺省的状态进行分配。图 39-7已简化用于这个示例。数据文件中的第一个块没有必要由分配给它的最小号的 PCM覆盖，而是使用了一个哈希算法。

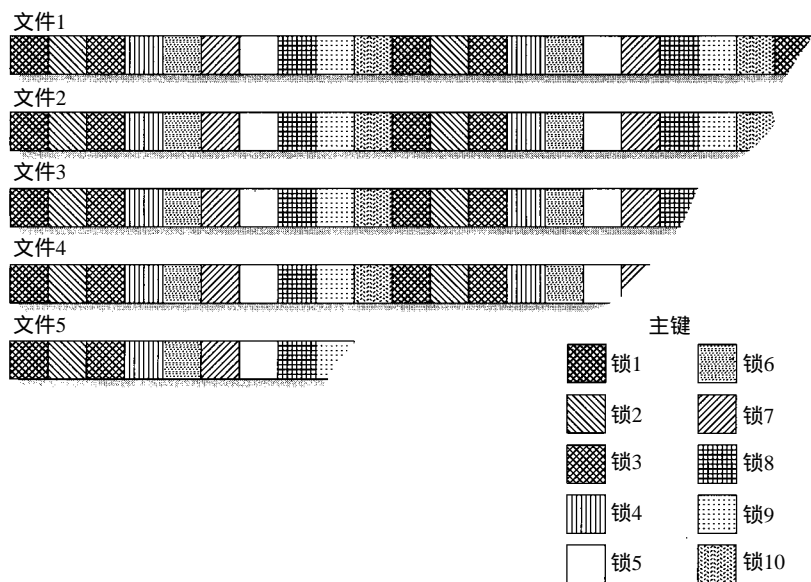


图39-7 哈希PCM锁的缺省分配

在Oracle8 8.0版本中，缺省的锁机制是精细锁，具有可释放的锁的数量与 DB\_BLOCK\_BUFFERS值相同。缺省情况下，Oracle8与Oracle8i的数据文件上没有预分配的锁。每当实例需要一个块时，可释放池中的一个精细锁分配给并覆盖这个块，当不再需要这个块时，锁被释放并可被下一个需要它的块获得。

当分配PCM锁时，不需要给以下种类的表空间分配任何锁：

只读模式。

临时模式。

只含有一个回滚段。

只含有一个临时段。

将这些表空间的文件ID号排除在GC\_FILES\_TO\_LOCKS参数之外，以避免分配不必要的锁。

提示 通过将所有的只读段放入它们自己的表空间,可以将这些表空间改变为只读的,因此减少所需要的PCM锁的数量。与此相似,可以在为临时段定义表空间时,通过使用TEMPORARY关键字减少必须分配的锁的数量。这可以阻止在这些表空间中创建任何其他类型的段,以便安全地对它们的数据文件分配 0个PCM锁。

另外,应该预定表空间只含有回滚段,因为你不需要对与其相关的数据文件分配PCM锁,只需要对回滚段自身分配 PCM锁。然而,没有特殊的关键字确保这些表空间摆脱其他类型的段。

### 1. 在7.3版本中的PCM锁

如果在7.3版本中使用GC\_DB\_LOCKS参数定义哈希锁,它们将在数据文件之间平均分配(参见图39-7)。你也许非常确定想要忽略这种缺省的组织,并按最适合你的应用与数据库设计的比例,对你的数据文件分配这些锁。使用初始化文件中的 GC\_FILES\_TO\_LOCKS参数完成这项任务。必须留有一些未分配的锁,以便 Oracle具有一池备用锁以分配给没在GC\_FILES\_TO\_LOCKS中指定的数据文件,或分配给也需要增加的数据文件。GC\_FILES\_TO\_LOCKS参数的语法如下:

```
GC_FILES_TO_LOCKS =
  ─"file_list=locks:file_list=locks:file_list=locks"
```

这里

```
file_list    is a file id number, or multiple file id numbers separated by
commas for individual files, or dashes for inclusive sets
locks        is the number of locks.
```

其中, file\_list是一个文件ID号,或多个以逗号分隔的独立文件的文件 ID号,或破折号,用于包括的集; locks是锁的数量。

可以包括任何需要的不同的文件列表,以冒号分隔每个文件列表及锁计数值。注意条目的完整集必须括在双引号之内,并且在条目之内还能有空格。在此是一个合法的分配了总数为1 005个锁的参数示例:

```
GC_FILES_TO_LOCKS = "1=100:2-5=500:6,8,12=400:7,9-11=5"
```

文件1接受100个锁,文件2~5共享另外的500个锁,文件6、8、12分享另400个锁,文件7、9、10、11共享5个锁。图39-8显示了文件7、9、10、11上的五个锁是怎样分配给这些文件的,每个文件的第一个锁依赖于哈希算法。这个参数中没有列出的任何文件将由备用锁覆盖。这将是 by gc\_db\_locks指派的但不是由 GC\_FILES\_TO\_LOCKS分配的锁平衡。在前面的例子中,将有 GC\_DB\_LOCKS减去1 0005的备用锁,这些备用锁可以如图39-7所示以缺省循环方式分配给文件。

如果想让这些锁覆盖连续批的块,可以使用组运算符,惊叹号(!),指出每组块的数量。例如,为使前例中的文件7、9、10、11上的每个哈希锁覆盖10个块,重新将参数写为:

```
GC_FILES_TO_LOCKS = "1=100:2-5=500:6,8,12=400:7,9-11=5!10"
```

图39-9显示这些块是如何分配给这些文件中的头几个块的。

另一个可以减少输入的参数串长度的选项是使用关键字 EACH,它对文件列表中的每个文件分配相同数量的锁。将前例重写为:

```
GC_FILES_TO_LOCKS = "1=100:2-5=500:6,8,12=400EACH:7,9-11=5!10"
```

你将分配总数为1 805个锁,文件6、8、12每个含有400个锁。

如果喜欢在7.3版中的数据库中使用可释放的锁,需要稍微修改一下全局高速缓冲区的参数缺省值。首先,需要设置 GC\_RELEASABLE\_LOCKS参数需要的精细锁的数量。它的设置

值不能小于DB\_BLOCK\_BUFFERS参数的值。如果希望，可以将GC\_RELEASABLE\_LOCKS的值设得比DB\_BLOCK\_BUFFERS参数的值稍高一些。下面决定你是否想要使用所有的精细锁或是想要将精细锁与哈希锁混合使用。前者通过将GC\_DB\_LOCKS设置为0即可实现，第二个选项需要在将GC\_DB\_LOCKS与GC\_\_RELEASABLE\_\_LOCKS设置为非0值后，再分别设置每一类锁的各自的参数。

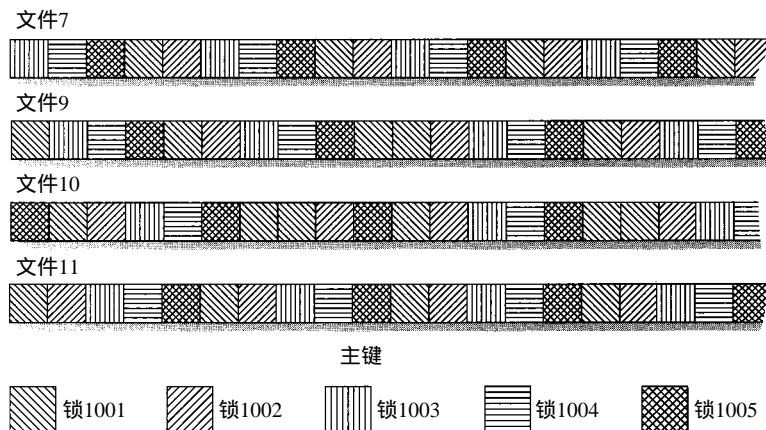


图39-8 GC\_FILES\_TO\_LOCKS="...7, 9-11=5"的哈希PCM锁分配

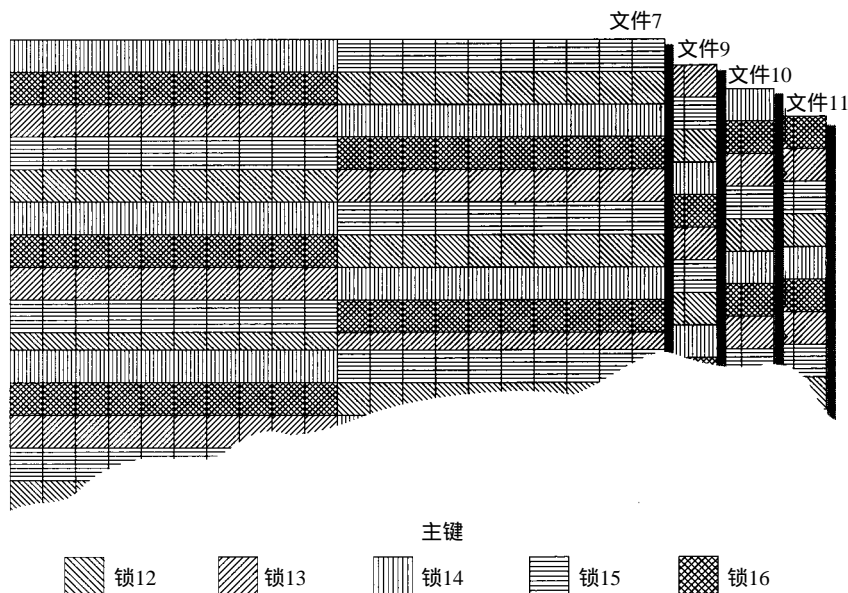


图39-9 GC\_FILES\_TO\_LOCKS="...7, 9-11=5!10"的哈希PCM锁分配

要在一个特定类的锁上使用精细锁代替哈希锁，将相关的参数设为 0。表39-1列出了参数及可以控制的锁的类型。可以通过将GC\_FILES\_TO\_LOCKS参数中的数据库段上的锁分配为 0，为数据库段分配精细锁。清单 39-2是将所有段标题块上与文件 13~17一起设置为精细锁的参数文件摘录。

清单39-2 一个Oracle 7并行实例参数文件的示例条目

---

```

DB_BLOCK_BUFFERS = 1000      # Total block buffers
GC_DB_LOCKS = 500            # Total hash locks
GC_RELEASABLE_LOCKS = 1000   # Total fine grain locks
                                # Assign fine grain locks to:
GG_SEGMENTS = 0              # (a) segment header blocks
GC_FREELIST_GROUPS = 0       # (b) freelist header blocks
GC_ROLLBACK_SEGMENTS = 0     # (c) rollback segment header blocks
GC_FILES_TO_LOCKS = "13-17=0" # (d) datafiles 13 through 17
GC_ROLLBACK_BLOCKS = 200     # Assign hash locks to rollback blocks

```

---

清单39-2中没有包括的数据文件将分享哈希锁，有 500多个哈希锁，因为 Oracle使用内部算法增加了GC\_DB\_LOCKS中设置的值后面的数量。在这些锁中，200个分配用于回滚段重做块，只留下300多一点的锁以普通的循环方式分配给未列出的文件。

精细锁也许是对段标题块的最好的选择，如清单 39-2所使用的，除非具有一个非常繁忙的数据库，拥有大量的活动实例，例如一个运行联机事务处理（OLTP）系统的大量并行处理（MPP）。然而，分配给特定锁类型的哈希锁的缺省数量可能不够。可能改变的两个参数是GC\_SEGMENTS与GC\_ROLLBACK\_SEGMENTS。前一个应该设定为数据库中的表与索引的总数，后一个设为回滚段的数量。

在7.3版本中，可以如先前关于哈希锁的讨论一样，分配好的精细 PCM锁覆盖连续批的块。为在GC\_FILES\_TO\_LOCKS中设置精细锁的组特性，必须对锁的数量使用 0值，否则文件会回复为哈希锁定。下面一行是从一个参数文件中提出的，其中同时使用了哈希锁与精细锁，一些有组选项，一些没有组选项中。等号右边的 0值强制文件9~14与18上用精细锁，“！”导致锁在文件9~14中的20个连续块集及在文件19~22上的10个块集上被分组。

```
GC_FILES_TO_LOCKS = "1=50:5-8=100EACH:9-14=0:20:15,18=0:16-17,19-22=1000:10"
```

## 2. 在Oracle8中的PCM锁

如在表39-1中所看到的，在Oracle8数据库中可处理的全局高速缓存参数比 Oracle7数据库要少得多。然而，在怎样分配锁方面确实具有更多的选项。首先，应该注意到从版本 8.0开始，缺省的锁机制是精细锁而不是哈希锁，Oracle承担将锁分配给许多锁种类的能力。第二，哈希锁可以如在Oracle 7.3中一样运转，或者它们也可以承担精细锁的一些特性。第三，在回滚段重做块上的锁使用与GC\_FILES\_TO\_LOCKS参数相似的语法进行分配。在一个特定的锁类型分配给一个锁或一组锁之后，它的运转将与在 7.3版本中一样。一个非释放的哈希锁（缺省类型）不会被一个实例放弃，除非由代表另一个实例的 DLM要求这样做；一个精细锁或可释放的哈希锁在实例使用它完成工作后，立即释放。

在Oracle 8 8.0版本中引入的新的参数GC\_DEFER\_TIMER，如果设置正确可以帮助减少pinging。在Oracle 7中，或者将GC\_DEFER\_TIMER设置为它的缺省值0，一个实例对PCM锁的需求以他们可以处理的速度被很快满足。在有些情况下，两个实例试图同时使用相同的块，或相同的一套少数块时，将一遍又一遍地请求必需的 PCM锁，以完成一个单独的事务。结果的锁变换与块pings阻止了这两个事务以快速方式完成工作。通过将GC\_DEFER\_TIME设为一个正数，数量单位是1/100秒，可以允许实例在满足DML请求释放一个PCM锁之前等待这个数量的时间，这使实例在请求实例释放锁与任何修改的块之前，可以有时间在锁下完成它的工作。

表39-1中列出的 Oracle 8 参数的其他改变在表中是不明显的。首先，有一个新的选项，允许指明想要释放哪个哈希锁，这需要在 GC\_FILES\_TO\_LOCKS 参数中包括字母 R 以指明这些锁（或者由它自己，或者作为 EACH 关键字的前缀）。下例中的条目显示了这个新特性的合法的使用。

```
GC_FILES_TO_LOCKS = "1=100:4-6=500R:7,9=200EACH"
```

提示 如果想要减少一个实例在启动过程中，或在计划内或计划外的实例关闭之后的重新启动过程中的时间，对哈希锁使用 R（可释放的）选项，这可以帮助你满足服务级的协议。然而，应该认识到，一个用户进程的每个可释放哈希锁的初始获得需要花费更长的时间，所以与在缺省方式下获得哈希锁相比，会有轻微的性能下降。

可以看到文件 4~6 分离的 500 个哈希锁是可释放的，与文件 7~9 分配的 400 个哈希锁一样，文件 7~9 中每个文件具有 200 个锁。

对哈希锁使用 R（可释放的）选项不会导致这些锁与好的精细可释放锁表现得完全一样。一旦获得，这些锁是不可释放的，但在实例的生命期中被保留，并在实例之间传递，就像任何其他哈希 PCM 锁一样。这个选项的目的在于允许锁在第一次需要时获得，而不是在实例启动时获得。

警告 如果同时需要 R 与 EACH 选项，它们必须按所示的正确的顺序连接，表达式 EACHR 是非法的。

在 GC\_FILES\_TO\_LOCKS 参数中的其他选项在所有当前的 Oracle 版本 7.3、8.0 与 8.1 中没有改变。

第二个在 Oracle 8 中以新的形式显示的参数是 GC\_ROLLBACK\_LOCKS。代替为这个值提供一个单一的数，需要使用一个与 GC\_FILES\_TO\_LOCKS 相似的格式为每个回滚段分配不同数量的锁。此处有一条约定：回滚段不能共享锁。当使用这个参数时，如果指明了回滚段的范围，必须使用 EACH 选项，如下所示：

```
GC_ROLLBACK_LOCKS = "1=50:2=100:3-10=200EACH:11-15=30EACH"
```

在这个参数中，等号左边的值是回滚段的 ID 号，不是如 GC\_FILES\_TO\_LOCKS 中的文件 ID 号。可以通过查询 DBA\_ROLLBACK\_SEGMENTS 或 GV\$ROLLNAME 将 ID 号改为数据库中的回滚段的名字。这两个查询结果将显示重做段号（USN）或回滚段 ID 号及回滚段名。

最后一个在 Oracle 8 8.0 版本中引入的对现存全局高速缓存参数的改变是 GC\_RELEASABLE\_LOCKS。这个微小的改变允许将值设置为比 DB\_BLOCK\_BUFFERS 小。现在可以将这个参数值设为在各个实例中是相等的，即使为了减少锁的开销，这些实例中的 DB\_BLOCK\_BUFFERS 的值是不同的。

警告 建议不要将 GC\_RELEASABLE\_LOCKS 设置为比 DB\_BLOCK\_BUFFERS 小的值，除非确定不需要所有的锁。只有两种可能情况：首先，如果已使用 GC\_FILES\_TO\_LOCKS 定义了所有可释放的锁同时覆盖内存中的多个块，第二，如果调整缓存中的块平衡总是被哈希锁覆盖的。如果没有足够可用的可释放的锁，所有的锁都在使用，应用将被迫处于等待状态，直至一个锁成为可用的，即使调整缓存中具有需要这个锁的块。

## 39.2.4 并行服务器初始参数

与前面几节讨论过的全局高速缓存（GC\_）参数一样，为成功地管理并行数据库，需要在init.ora文件中使用一些附加的参数。表 39-2显示了可以在哪个版本中使用这些参数。

表39-2 PCM锁管理参数

参 数 名	版本		
	7.3	8.0	8.1
ALLOW_PARTIAL_SL_RESULTS			X(4)
CACHE_SIZE_THRESHOLD		X(3)	X(4)
DELAYED_LOGGING_BLOCK_CLEANOUTS	X(2)	X(3)	X(4)
DML_LOCKS	X(2)	X(3)	X(4)
FREEZE_DB_FOR_FAST_INSTANCE_RECOVERY		X(3)	X(4)
IFILE	X(2)	X(3)	X(4)
INSTANCE_GROUPS		X(3)	X(4)
INSTANCE_NAME			X(4)
INSTANCE_NUMBER	X(2)	X(3)	X(4)
LM_LOCKS		X(3)	X(4)
LM_PROCS		X(3)	X(4)
LM_RESS		X(3)	X(4)
MAX_COMMIT_PROPAGATION_DELAY	X(2)	X(3)	X(4)
OPS_ADMIN_GROUP			X(4)
PARALLEL_DEFAULT_MAX_INSTANCES		X(3)	X(4)
PARALLEL_INSTANCE_GROUP		X(3)	X(4)
PARALLEL_SERVER		X(3)	X(4)
PARALLEL_SERVER_INSTANCES			X(4)
PARALLEL_TRANSACTION_RESOURCE_TIMEOUT		X(3)	
SERVICE_NAMES			X(4)
THREAD	X(2)	X(3)	X(4)

## 1. ALLOW\_PARTIAL\_SN\_RESULTS

如果这个值设为TRUE，将允许一个全局事务表的查询结束，即使不是所有组的实例都在联机。如果设为FALSE，不能获得查询组的一个或多个实例的查询将失败。应该将这个值设为TRUE，以在批处理程序中获取全局视图（GV\$\_）以便它们不会失败。将它设为FALSE以使用这些视图来接收任何实例失败的警告。

## 2. CACHE\_SIZE\_THRESHOLD

任何表可以被缓存入一个单独实例数据库的缓存中的块的最大数量。缺省将它设置为DB\_BLOCK\_BUFFERS值的1/10，通常是一个推荐值。

## 3. DELAYED\_LOGGING\_BLOCK\_CLEANOUTS

当事务提交时，该事务改变的块将被更新以指明在一个进程中的事务的完成，这通常称为块消除。有些情况下，清除与生成相关的重做信息被延迟直至块再次存取。当这种延迟清除发生时，块是脏的，如果另一个实例需要这些块，需要一个排它PCM与潜在的ping到硬盘。通过将DELAYED\_LOGGING\_BLOCK\_CLEANOUTS设为TRUE，可以阻止查询执行延迟块清除。因而，只有需要排它PCM锁的DML命令将执行延迟的清除。如果将这个参数设为FALSE，可以执行延迟块清除，因而产生潜在的pings。

## 4. DML\_LOCKS

这个参数控制因并发事务修改的表中的排队。将所有实例中的这个值设为 0 将提高性能，但下面的语句将被禁用：DROP TABLE，CREATE INDEX 及显式锁声明（LOCK TABLE）。当不需要使用前述命令时，可以将这个值分配为 0。

#### 5. FREEZE\_DB\_FOR\_FAST\_INSTANCE\_RECOVERY

这个参数控制在实例进行恢复的过程中是否冻结所有的实例。如果设为 TRUE，Oracle 禁止所有的硬盘读写，除了那些与恢复过程相关的硬盘读写。这允许恢复更快地完成，但在恢复完成前将有效地冻结所有的用户活动。当这个值设为 FALSE 时，用户可以继续在实例失效没有影响到的数据库部分上工作，但他们将与恢复过程发生硬盘存取竞争。Oracle 选择这个参数的缺省值的方法基于哈希锁与精细锁所覆盖的文件的比例，如果所有的联机数据文件都使用哈希锁，这个值就是 FALSE。

#### 6. IFILE

IFILE 参数指明当前文件中包括的另一个参数文件。使用这个参数包括一个常用的含有所有参数的文件，这些参数在所有的实例上是相同的。你的特定版本的文档将指明哪些参数必须在所有的结点上具有相同的值。理想情况下，将这个文件放在一个共享文件系统中以使所有的实例都可以读取一个唯一的拷贝。

**警告** 所包括的参数文件将被逻辑地插入到当前 init.ora 文件中包括 IFILE 参数的地方，如果在这个包括文件中的一个参数在当前文件的后面又重复定义了一遍，实例将使用当前文件的值，而不是所包括的文件的值，因为它是最后一个读到的值。

最好将 IFILE 参数放在实例的 init.ora 文件的最后部分。如果使用了一些逻辑的参数安排组织参数文件，如按功能、按字母或按日期的改变顺序排列，需要包括 IFILE 参数两次；一次在它的逻辑位置带有注释，该注释在文件末尾被重复，另一次是文件的最后一项。

#### 7. INSTANCE\_GROUPS

INSTANCE\_GROUPS 命名实例将分配的一个或多个实例组，组名可以是在 PARALLEL\_INSTANCE\_GROUP 参数中显式或隐式的标识。只有在 PARALLEL\_INSTANCE\_GROUP 中命名的实例，不管是在 init.ora 文件中还是使用 ALTER SYSTEM 或 ALTER SESSION 命令命名的，将被选中用于执行并行操作。

#### 8. INSTANCE\_NAME

当想要指明需要连接哪一个实例时，这个参数设置在 Net8 连接中使用的实例名。它的缺省值是在操作系统中设置的实例标识符（SID）。参见 SERVICE\_NAMES 参数的描述以获取更多的连接选项的信息。

#### 9. INSTANCE\_NUMBER

INSTANCE\_NUMBER 与这样的实例有关：这个实例具有手工分配的段中的空闲列表及在分配给表和索引的特定的空闲列表组的实例。如果表或索引具有比 INSTANCE\_NUMBER 值分配的少的空间列表组，模数算法将用于指明使用的空闲列表组。缺省值是 0，在这种情况下，在启动时分配的实例号基于这个实例相对于其他实例的启动顺序。

#### 10. LM\_LOCKS

这个参数指明了 IDLM 使用的锁的数量，这个值不能小于：

资源 + (资源 \* (结点数 - 1)) / 结点数

如果应用在试图执行 DML 时冻结了，需要通过在上述建议的最小值上增加这个参数的值，

来为DML增加更多的锁。

提示 Oracle8与Oracle 8i并行服务器概念与管理手册含有用于计算一个合理的LM\_LOCKS与其他的IDLM参数(LM\_)值的计算的详细解释。它们也提供了可以用于执行这些计算的示例工作表。

#### 11. LM\_PROCS

LM\_PROCS控制可以使用锁管理服务的Oracle进程的数量,这个值至少为(进程数+活动实例的数量)。

#### 12. LM\_RESS

LM\_RESS定义了DLM层使用的资源的数量,这个值应该为

$$2 * ( ENQUEUE_RESOURCES + GC_FILES_TO_LOCKS \\ GC_RELEASABLE_LOCKS + GC_ROLLBACK_LOCKS )$$

建议不要将这个值减小到比上述表达式的结果小。如果不知道这个计算中的一个或多个参数,应该启动一个实例并查询V\$PARAMETER表以找到所需要的值。

#### 13. MAX\_COMMIT\_PROPAGATION\_DELAY

这个参数设置了任何对的实例在同步它们的系统修改序列号(SCN)之前需要的最长时间。可以选择降低这种同步的频率而减少实例之间的消耗,但如果这样做,将会冒这样的风险:在一个实例中已提交的修改在另一个实例中却未被提交的,这是因为SCN用于确定在修改发生之前还是之后开始查询。通常,缺省值已经足够了,应该使用缺省值,除非查询看不到应用所要求的足够的当前的修改。更快的传播比例会伤害性能。

#### 14. OPS\_ADMIN\_GROUP

这个参数用于向实例分配一个实例组名,以允许为系统管理与报表的目的指明实例的子集。使用GV\$\_VIEW的查询仅返回与当前实例具有相同组名的实例的结果。

#### 15. PARALLEL\_DEFAULT\_MAX\_INSTANCES

这个参数指明了当具有INSTANCES的DEFAULT值,在并行方式下处理的查询时,查询协调器执行一行查询所使用的实例的缺省号。这个参数的缺省值出自不同的操作系统的值,对大多数数据库来说,这个缺省值是最佳的。

#### 16. PARALLEL\_INSTANCE\_GROUP

PARALLEL\_INSTANCE\_GROUP指明了以并行方式跨越多个实例执行的操作所使用的实例组。只有伺服器进程属于请求组,如由INSTANCE\_GROUPS参数所设置的,将用于执行并行操作。当数据库被分成相关实例的子集时,这个参数是有用的。但如果需要,可以使用ALTER SYSTEM或ALTER SESSION命令覆盖它,以指明一个替代的实例组。

#### 17. PARALLEL\_SERVER

这个参数指明实例可以作为并行方式与其他实例共同打开,它替代了7.3版本中STARTUP命令的关键字PARALLEL,或选项SHARED。如果设为FALSE,这个实例必须是排它的,是在数据库上运行的唯一的一个实例。

#### 18. PARALLEL\_SERVER\_INSTANCES

把这个参数设置为希望在数据库上以并行方式运行的实例的总数。Oracle使用这个值将SGA调整为这个数的大小。在多数情况下,比起使用缺省值来,这将会对内存进行更好的使用。

### 19. PARALLEL\_TRANSACTION\_RESOURCE\_TIMEOUT

这个参数设置了在并行方式下执行的操作，在等待另一个实例的 PCM 锁或其他共享资源被释放或降级为一个兼容的级别时所需要的秒数。如果在这个指定时间内，不能获得指定资源，这个操作将成为过时的，这种过时将由实例之间的竞争表示，包括潜在的死锁。可以为这个参数分配 0 值，它将把过时时间设置为无限大，阻止了这种过时的发生。

### 20. SERVICE\_NAMES

Net 8 使用服务名帮助在多个实例之间均衡用户的负载。这个参数指明了与一个实例相关的一个或多个服务名，用户请求 Net 8 连接到这些服务名中的一个，而不是连接到一个指定的实例。Net 8 将这个用户的连接分配给与命名服务相关的一个实例，使用一种算法决定哪个可用的实例是负载最小的。

### 21. THREAD

THREAD 指明实例使用的重做日志线程的数量。如果数量为 0，实例将试图获得一个空闲的公有重做线程。如果这个数是非零值，实例将试图获得这个数量的重做线程，不论它作为私有线程还是公有线程启用。如果需要的线程是可用的，实例需要获得它供其专用；否则实例将不能打开。关于重做线程以及私有和公有选项的讨论将在随后章节出现。

## 39.2.5 并行服务器的回滚段考虑

并行实例将与数据库上的其他实例操作共享系统回滚段。至少需要为每个实例增加一个附加的回滚段用于专用，否则实例将不能启动。在创建数据库之后，需要在一个专有实例中所做的是：必须使用这个实例增加所需的回滚段。当然，可以为每个实例增加一个以上的回滚段，遵循与在单实例数据库中相同的方针。

当为并行服务器创建回滚段时，可以使用私有段或公有段。虽然私有回滚段需要更多的初始化工作，但它们给予更多的控制并在长时间的运行中更易于管理。如果使用私有回滚段，这是缺省类型，需要在与它们相关的初始化文件的 ROLLBACK\_SEGMENTS 参数中列出它们的名字。而对公有回滚段，需要在 CREATE ROLLBACK SEGMENT 命令中使用 PUBLIC 关键字，还需要把 TRANSACIONS 与 TRANSACTIONS\_PER\_ROLLBACK\_SEGMENT 两个参数设置为要求的值。如果每个实例都需要相同数量的回滚段，可以对所有的实例使用一个包括参数文件，简单地将所有实例的这些参数设为相同的值。也可以控制每个独立的实例试图获得的公有回滚段的数量。Oracle 计算两个不同的参数——TRANSACIONS 与 TRANSACTIONS\_PER\_ROLLBACK\_SEGMENT 的比例，通过将结果值上舍入到与其相近的最大整数（CEILING 函数），决定实例应该获得的专有回滚段的数量。

```
Number of public rollback segments required =  
CEILING ( TRANSACTIONS / TRANSACTIONS_PER_ROLLBACK_SEGMENT )
```

当使用公有回滚段时要非常小心，要为每个实例创建它需要数量的足够的回滚段。只要有一个回滚段是可用的，实例就可以启动。这会导致实例需要比预期的要少的回滚段，并产生一个性能较差的实例。如果由于一些实例需要比预期更多的回滚段而没有可用的足够的空闲回滚段，一个或多个实例会不能启动。

这种实例会以比需要的活动回滚段少的数量启动的倾向是推荐私有回滚段的原因。因为必须命名私有回滚段，你的实例只有在获得他们设计的回滚段时才会启动。如果不小心在两

个参数文件中命名了一个相同的回滚段，只有第一个启动的实例可以获得它，第二个实例在没有被分配到回滚段的情况下将不能启动。

推荐私有回滚段的第二个原因是它们的位置。当创建回滚段时，可以将它们分配给一个特定的表空间，将每个实例的回滚段放在它们自己的表空间中或一套表空间中是很有用的。当底层硬盘驱动器不可用时，这很容易识别哪个实例将受到影响。而且，当使用非共享体系结构时，应该将回滚段放置在实例的本地硬盘上。如果使用公有回滚段，你不能控制一个实例需要本地硬盘的回滚段还是非本地硬盘的回滚段。

**警告** 只有很少的Oracle并行服务器工具只支持公有回滚段。如果你正使用这样的系统，你的特定平台的文档将会指明。

### 39.2.6 重做日志与并行服务器实例

在并行服务器环境下的每个实例运行它自己的 LGWR进程。在大多数平台上，这些实例写它们自己的一套联机重做日志文件。有两个体系结构，其中日志将它们的输出写向一个集中的进程，这个进程交错将它们插入到一套单独的日志文件中。这种平台很少，可以从安装指南中轻易地决定你的平台是否属于这种平台。本节的其余部分重点介绍写它们自己的重做日志文件的实例。

一个实例使用的一套日志文件称为 thread。通过使用回滚段，可以将线程设为公有的或私有的。与回滚段相似，重做的公有线程由实例在启动时从可获得的资源池中获得，还能控制哪个线程由哪个实例获得。与公有回滚段应用相关的相同的考虑也适用于重做的公有线程。

当使用公有重做线程时，不需要知道哪个线程正由哪个实例写，因而，当经受硬盘失效时，也很难确定哪个用户将受到影响。在数据库扩展时，也不能为所有实例启用足够的公有线程，因而，在启动时会有比预期的要少的实例是可用的。如果正在一个什么也不共享的平台上使用并行服务器，应该使用私有重做线程将相关的日志文件放在每个实例结点的本地硬盘上。

必须在一个活动的实例中创建并启用重做线程。达到这个目标的命令是选择使用 ALTER DATABASE命令，如程序清单 39-3所示的那样。也应该注意到重做的一个线程必须在它的相关实例可以启动之前创建和启用。为每个实例的重做日志选择一个唯一的线程号，注意必须在整个数据库中分配唯一的重做日志组号，而不只是在每个线程中。

清单39-3 定义一个重做的线程并使其可用

```
SQL> ALTER DATABASE ADD LOGFILE THREAD 2
2      GROUP 3 ('/ora/testdb/redo1/redo3a.log', '/ora/testdb/redo2/redo3b.log')
3      SIZE 1M
4      GROUP 4 ('/ora/testdb/redo1/redo4a.log', '/ora/testdb/redo2/redo4b.log')
      SIZE 1M;

Database altered.

SQL> ALTER DATABASE ENABLE PUBLIC THREAD 2;

Database altered.
```

与回滚段不同，使一个线程成为公有的或私有的选项是 ALTER命令的一部分，而不是

CREATE命令的一部分。为使一个线程成为私有的，在 ALTER DATABASE命令中省略 PUBLIC关键字。如果需要改变一个已启用进程的状态，例如将它从公有线程转变为私有线程，首先必须关闭任何当前正在使用这个线程的任何实例，并从另一个不同的实例中进行这个改变。改变自身需执行 ALTER DATABASE命令两次，首先使用 DISABLE THREAD选项，然后使用所需的 ENABLE THREAD选项。

如果实例不是做相似的工作，不必使所有的线程都是相同的。例如，可以在一个处理许多事务的实例的一个线程中具有更大的或更多的重做日志，面对读密集型实例使用更少的或更小的重做日志。如果决定使用这样一种变化的重做结构，必须启用私有实例，以确保每个实例获得适当的线程。如果计划你的实例相互之间作为故障转移实例，应该考虑为每个实例使用相同的重做线程。

### 39.2.7 使用空闲列表组以避免竞争

Oracle使用空闲列表识别在一个 INSERT操作中，段中的哪个块含有用于新行插入的空间。空闲列表中的第一个可用的块被段标题块的条目识别，其他的块在空闲列表中由指针将每一个块链接到下一个块。缺省情况下，一个段的所有空闲列表开始于相同的标题块。当一个段含有多个空闲列表组时，为这个段创建一个附加的、特定的标题块。这些标题块的每一个含有起始指针，指向属于一个特定空闲列表组的空间列表上的块。

**警告** 虽然大多数关于空闲列表的讨论指明了它们用于 INSERT操作，还应该清楚 UPDATE也需要获取一个空闲列表。如果由于一行或多行长度的增长，行不再适合它的起始块，它将迁移到一个不同的块。对这种迁移行来说，如果没有给段分配块，将不得不从空闲列表中找出一个空的块。

并行服务器使用多个空闲列表组有两个主要的原因：第一是避免对空闲列表访问的竞争。在多用户数据库中，在某一时间只有一个用户可以控制一个空闲列表，所以需要多个空闲列表以允许对相同段的并发插入。基于数据库块的大小，有一个可以分配给段的空闲列表数量的上限。对每个活动的数据库与段，也许需要通过包括多个空闲列表组，每个具有多个空闲列表来增加更多的空闲列表。

由于一个常用段的空闲列表的信息存储在段的标题块中，所以也会发生空闲列表的竞争。当块从空闲列表中增加或删除时，标题块中的这条信息必须被更新。当不同实例上的空闲列表经常改变时，这个活动产生消耗，并由于标题块的 pinging而导致严重的性能下降。

第二个使用空闲列表组的原因是控制实例插入时使用哪个块。如果两个或更多的实例共享空闲列表，它们很有可能试图将新的行放入相同的块中，由于每个空闲列表头上的块在实例之间传递，这将再一次导致过量的 pinging。通过使用空闲列表组，可以显式地或隐式地将一批块分配给指定的一空闲列表并确保那些块是由不同的 PCM锁覆盖。然后每个实例只执行在它自己的空闲列表组的空间列表中的块的插入，避免使用分配给另一个实例的块。

第一种方法使用一个表，在这个表中，区间被手工地划分并分配给一个实例号，它隐式地将它们与一个空闲列表组相关联。当查找可用的块时，每个实例将使用与它在启动时获得的实例号相关的空闲列表组。对任何特定实例的插入将指向一个特定区间的块。通过将这些区间放置在分离的数据文件中，使用 GC\_FILES\_TO\_LOCKS参数，可以确保为每个实例区间使用不同的PCM锁。结果是，PCM锁与块在实例之间不再是必须共享的，相关的块也不需要

被ping。用这种方法建立的有四个空闲列表组的表如图 39-10所示。

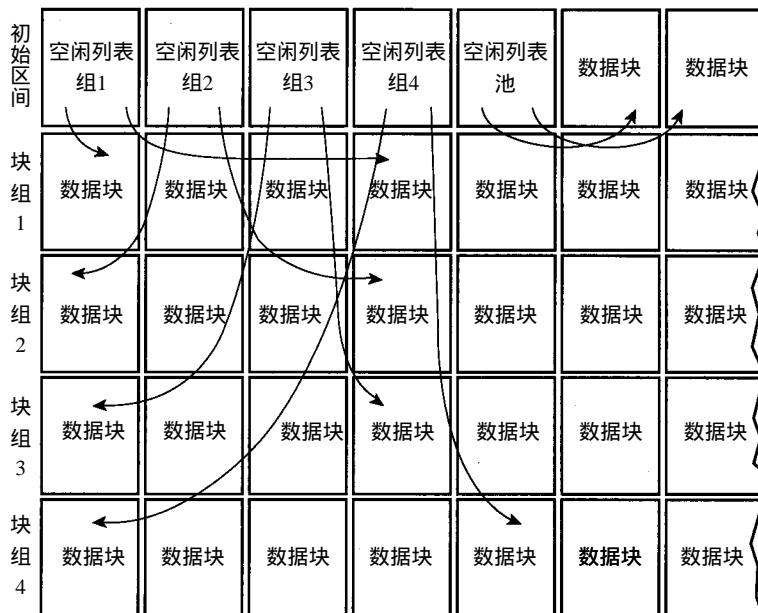


图39-10 具有四个空闲列表组的表

为了使用这种空闲列表组分配的方法，应该使用 STORAGE子句的FREELIST GROUPS选项建立这个表，并使用ALTER TABLE ... ALLOCATE EXTENT命令增加实例指定的区间。这个命令给予你将每个划分的区间分配给一个特定的数据文件与实例号的能力，需要监控这种方案中的空间使用情况，确保 Oracle没有动态地划分新的区间，因为你不能控制它们在哪个数据文件中创建，它们有可能会被分配给存储在主表标题块的普通空闲列表。也应该使用一个最小的初始区间，因为你不愿记录插入到表的这个常用部分，应该将这个区间与实例的特定区间分开。图39-11显示了一个已建立的表仅支持两个实例在两个空闲列表组上的插入。注意如果两个实例需要访问这个区间，初始区间不只在它自己的数据文件中，而且使用精细锁以减少竞争。

为避免你没有指明分配给一个空闲列表组的块的竞争，应该考虑以下方针：

- 1) 创建只足够用于保存段标题块与空闲列表标题块的初始段。
- 2) 使用STORAGE子句为每个实例/空闲列表限制初始区间的 + 1的区间数。
- 3) 只为每个实例在它自己的数据文件中分配一个区间。
- 4) 监控空间的使用，当一个实例需要更多的空间时，只增加 MAXEXTENTS存储子句，在这种情况下，将MAXEXTENTS加1，并立即分配一个实例特定的区间。

在程序清单 39-4中，可以看到怎样初始建立表，并为两个实例的每一个增加一个区间。

清单39-4 创建一个表用于两个实例共享

```
CREATE TABLE my_table (id NUMBER(10), . . . )
  TABLESPACE user_data
  STORAGE (INITIAL 10K MAXEXTENTS 3 FREELIST GROUPS 2);

ALTER TABLE my_table
  ALLOCATE EXTENT (SIZE 1M DATAFILE 'user_data_instance1_01.dbf' INSTANCE 1);

ALTER TABLE my_table
  ALLOCATE EXTENT (SIZE 1M DATAFILE 'user_data_instance2_01.dbf' INSTANCE 2);
```

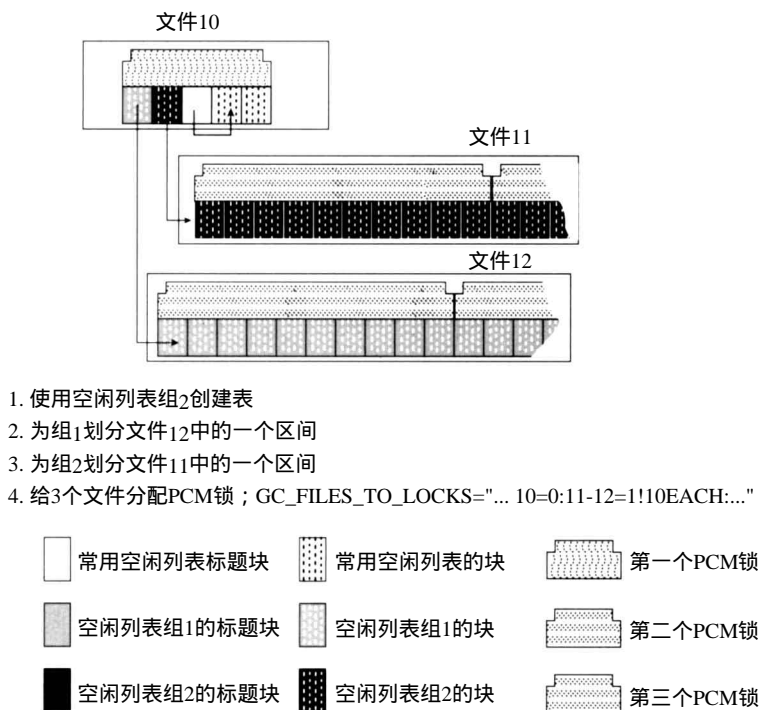


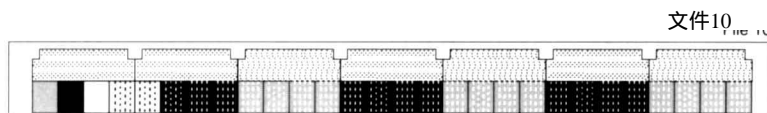
图39-11 空闲例表组的手工分配

第二种可以用于为不同的实例划分插入所用的块与 PCM锁的方法是使用精细锁上的组选项。为理解这点，需要一些背景信息。当 Oracle向段中插入项时，它将移动最高水位，一个段标题块中的指针到段中的最后使用的块。做这个工作以降低空闲列表必须跟踪的块的数量；任何最高水位之后的块逻辑上是不能使用的，所以不必包括在空闲列表中。通常，Oracle一次移动最高水位五个块。

然而，如果给数据文件分配了精细锁，段被划分到多个空闲列表组，Oracle改变了它的缺省状态。当达到当前的最高水位并需要移动这个水位时，Oracle通过为这个数据文件在 GC\_FILES\_TO\_LOCKS参数中定义的数量移动它。另外，它为实例的空闲列表组分配了一个单独的可释放的PCM锁，覆盖所有最高水印之后增加的块。这保证了这些块可以被看到并只被它们分配的实例使用。

另一个需要表中空间的实例将再次移动最高水位，一个进程将为它的空闲列表组获得相同数量的附加块，并且一个单独的可释放锁覆盖住它们。这种每个实例最高水位的移动，及伴随的空闲列表与PCM锁分配，保证了插入将只发生在与特定实例的空闲列表组标题块相关的块上及早已分配的PCM锁上。这个行为如图39-12所示，再次限制为两个实例的数据库。

当使用空闲列表组时，假设块总是只属于一个实例是不安全的。在块分配给一个空闲列表组之后，它还呆在它的相关的空闲列表中直到它已被填满到 PCTUSED的值，此时块将返回给进行这个修改的实例的空闲列表组。这个实例不必是开始在它的空闲列表中含有这个块的实例。与此相似，如果卸出然后载入一个具有多个空闲列表组的段中的行，行将被放入属于执行这个载入的实例的空闲列表组中。因而，它们不必要，也不会返回到它们的初始空闲列表中。



1. 使用空闲列表组2创建表
2. 给文件分配PCM锁；GC\_FILES\_TO\_LOCKS=" 10=100!4"

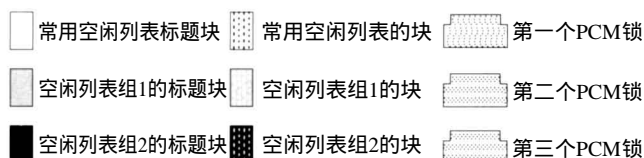


图39-12 自动划分空闲列表组与PCM锁

### 39.3 决定何时并行服务器可以解决一个商务需要

为决定并行服务器是否很适合你的超越故障的需要，需要将它的成本和复杂性与其他的选项进行比较。如果需要数据库每天 24 小时，每年 365 天都处于可用状态，将适合使用并行服务器。当至少有一个实例是活动的时，用户可以继续存取数据库。应该考虑使用并行服务器作为增长应用、数据库大小或用户团体的手段。

下面两节讨论在并行服务器上建造可用的与灵活性的应用所需要的工作。通常按下面各节中的描述进行工作，你应该可以决定你的环境是否适合使用并行服务器。一些问题很适合并行服务器的体系结构，而一些问题不适合使用它。为了做出最好的决定，应该完成这种替换策略的相似性分析，并比较它们的正面与负面结果。

### 39.4 为超越故障设计一个并行数据库

最简单的超越故障设置的组成是：一个两结点的簇，Oracle 实例运行在每个结点上，而数据库存储在一个共享磁盘系统上。缺省情况下所有的用户连接到相同的实例，同时具有连接到替代数据库的第二条路径。在客户 / 服务器环境下，SQL\*Net 或 Net8，( 依赖于 Oracle 版本 ) 可以配置提供替代的、超越故障地址。在三层或 n 层体系结构中，一个 TP 类型的监控器可以路由用户到活动的实例。

这个模型可以被扩展，包括多个活动的结点与实例同时具有一个超越故障的实例，在此需要指出的是一个单独的结点与实例不能提供充足的故障保护。如果同时有两个实例失效——当结点增长时可能出现的增长的事件——一个单独的超越故障实例将不能处理所有失效实例的工作量。决定将哪个用户移动到哪个超越故障结点，它是衡量你的应用需要采取的分区分活动的子集。

不管有一个还是多个超越故障的实例，仍然需要建立 TNSNAMES.ORA 文件以指明主实例与超越故障实例，或在中间层组件中路由模式的其他代码，如 TP 监控器。为达到前一个选项，需要使用 Oracle 版本所允许的最好的选择。在最简单的所有版本中都具有的级别上，需要在 TNSNAMES.ORA 文件中 DESCRIPTION 子句使用一个 ADDRESS\_LIST 及具有相同名字的实例。程序清单 39-5 列出了 TNSNAMES.ORA 文件的一个条目示例。

清单39-5 在TNSNAMES.ORA条目中指明主结点与超越故障结点

```
PROD.world=
  (DESCRIPTION=
    (ADDRESS_LIST=
      (ADDRESS=
        (COMMUNITY=tcp.world)
        (PROTOCOL=TCP)
        (HOST=apple)
        (PORT=1521)
      )
      (ADDRESS=
        (COMMUNITY=tcp.world)
        (PROTOCOL=TCP)
        (HOST=pear)
        (PORT=1521)
      )
    )
    (CONNECT_DATA=
      (SID=PROD)
    )
  )
```

Oracle 8增加了并行服务器的超越故障的能力。首先，具有启动一个为每个主数据库的用户连接到超越故障实例的并行、哑连接的选项。如果主实例失效，重新连接到备用实例将比如果没有创建预先产生的连接要快得多。如果不希望允许在主实例运行时，其他任何到备用实例的连接，应该考虑使用这个选项。

第二，Oracle调用接口（OCI）语言被扩展以允许断开与主实例的连接的用户——因为主实例失效的原因——可以自动连接到超越故障实例。另外，如果用户正在执行一条查询的处理，查询将在这种重新自动连接之下对它们重新启动。TNSNAMES.ORA文件含有一个FAILOVER\_MODE子句以指明需要哪种类型的自动重新连接。新的关键字如表 39-3所示。

表39-3 为TNSNAMES.ORA提供重新连接选项的关键字

关键字名	功 能
BASIC	不工作在替代实例上的预先创建的连接
PRECONNECT	当切换到替代实例时，使用一已存在的连接
SELECT	在重新连接到替代实例之后，继续任何被打断的查询
SESSION	在实例失效之后，重新连接到替代实例
NONE	在任何主实例失效后，不试图进行自动重新连接

清单39-6展示怎样为Net 8建立一个TNSNAMES.ORA文件以提供一个从失效实例到替代实例的自动故障切换。

清单39-6 在TNSNAMES.ORA文件中定义超越故障结点

```
(DESCRIPTION=
  (ADDRESS=
    (PROTOCOL=TCP)
    (HOST=apple)
    (PORT=1521)
  )
  (CONNECT_DATA=
    (SIDGRP=fruit_group)
    (SERVER=DEDICATED)
  )
  (FAILOVER_MODE=
```

```
(TYPE=select)
(METHOD=basic)
```

如果决定利用OCI提供的超越故障选项的优势，应该清楚它们也可以用于实现一个软件超越故障。如果需要关闭一个实例，可以对用户透明地将活动的连接转移到它们的超越故障实例，使用如下命令

```
SHUTDOWN FAILOVER
```

停止当前的事务并用一个错误信息通知应用关闭告警。对没有计划的超越故障，这个技术依赖于OCI程序捕捉一个错误信息并调用超越故障连接。也可以使用如下命令将故障切换到一个单独的连接。

```
ALTER SYSTEM DISCONNECT SESSION 'sid,serial#' POST-TRANSACTION;
```

这个命令将再次使用一个错误通知应用，这个错误可以被 OCI程序捕捉并管理，在当前事务完成后，导致用户重新连接到超越故障实例。

不同于设置重新指向到替代实例的连接所必需的组件，为支持一个简单的故障切换选项不需要作任何数据库设计的修改。由于在某一时刻只有一个结点是活动的，所以不需要这样做的块与PCM锁共享。理论上，一个单独的锁对整个数据库已经足够了。然而，由于内部算法，将永远也不可能将锁的数量减少到这么少。通过将参数设置如下：

```
GC_DB_LOCKS = 1
GC_RELEASABLE_LOCKS = 0
```

将GC\_FILES\_TO\_LOCKS留作它的一个空串的缺省值，将允许 Oracle分配可能的最小数量的锁。

## 39.5 为可伸缩性设计一个并行数据库

可伸缩性指的是系统具有按需要增长的能力，不论是支持更多的用户、处理更多的数据还是执行更复杂的任务。可伸缩性系统设计阶段的主要目标在于确保在实例之间具有最小的竞争。这对查询密集型 Oracle数据库来说是一个很容易的问题，因为所有的 PCM锁都可以由所有需要它们的实例以共享模式同时获得。对于事务密集型数据库，特别是 OLTP系统，这个挑战比较艰巨。不得不将数据库分区以便每个实例使用块的一套子集，而另外的实例使用它们自己不同的子集。第二个目标是将工作量在实例之间尽可能地平均分布，从而不会使一台服务器的资源工作过度而在另外的服务器上却没有充分使用它们。

有一些怎样分区数据库的选项，包括应用、功能、部门 /线型商务、物理表与事务分区。以下几节解释这些选项。

### 39.5.1 应用分区与功能分区

应用分区与功能分区具有相似的步骤，但在开始分区的商务级别方面有明显的区别。

应用分区包括将不同的应用运行在不同的实例上，如金融与制造业。通常，应用使用完全不同的表集，所以很少需要将它们连接到同一个数据库中。如果在应用之间的确具有一些相同的数据，需要在相同的数据库中同时运行这些应用。为并行服务器设计具有这种特性的数据库，需要使用与功能分区相同的步骤。

功能分区包括将一个应用的不同功能分区。例如，需要将人力资源应用分为四个功能——招募新人、雇佣历史、薪水册与奖金。如果公司很大，需要它的雇员和征募的新人可以通过电话线与 Web 存取本公司的数据库，这将导致一个具有高度事务比率的巨型数据库，这种应用需要运行在一个并行服务器上。

对应用分区与功能分区，需要考虑数据的实现而不是功能的定义。这需要分析数据库中的每个表将如何被每个应用或功能使用。为简化这项工作，可以在表中记录下你的发现。以人力资源为例，可以按如下进行：首先，创建一个与表 39-4 相似的表，定义每个功能使用哪个表。

表39-4 分区表——功能使用的表

招募新人	雇佣历史	薪水册	奖 金
Table 1	Table 1	Table 1	Table 1
Table 2	Table 2	Table 3	Table 2
Table 3	Table 4	Table 5	Table 4
	Table 6	Table 7	Table 5

下面记录下公用表——那些由多个实例使用的表。为简化这个例子，只考虑表的薪水册与奖金子集。下一步表的迭代如表 39-5 所示。

表39-5 分区表——功能使用的公用表

薪水册	表 公用表	奖 金
Table 3	Table 1	Table 2
Table 7	Table 5	Table 4

如果在应用中发现了一些公用表，需要重新考察这些表的定义以确定它们是否需要重新定义。目标应该是最小化公用表列中表的数量。如果没有公用表，就具有一个完美的分区应用。在这种情况下，将不同功能的表放到分离的表空间中，并为所包括的每个文件分配一个单独的哈希锁，甚至是与每个功能性表空间或表空间相关的文件集分配一个单独的哈希锁。每个功能的用户将连接到它们的相应实例。

对仍处于公用列中的表，下一步是调查并记录下每个实例在表上的活动类型及存取的频率。在表 39-6 中，插入了人力资源示例的值。

表39-6 分区表——公用表活动

公用表名	薪水册存取与容量	奖金存取与容量
Table 1	SELECT 100/seconds	SELECT 300/seconds
Table 5	SELECT 200/seconds	SELECT 10/seconds
	INSERT 10/seconds	INSERT 0s
	UPDATE 20/seconds	UPDATE 200/seconds

现在可以准备设计表空间以减少 PCM 锁竞争与块 pinging 了。不是多个实例公用的数据库表可以被放置在它们自己的表空间中，只需要最小数量的 PCM 锁。这些表空间中的文件可以使用哈希锁覆盖，而不是使用精细锁，因为它们的竞争不会被其他实例共享。

在公用表列中的只读表——即，所有实例对它们只执行查询——也可以放置在它们自己的表空间中。应该在将数据加载之后就这些表空间改变为只读的，因为 Oracle 对不能被更新的数据不需要使用任何 PCM 锁。在 GC\_FILES\_TO\_LOCKS 参数中略去只读表空间的数据文件，以避免分配不必要的锁。

现在可以处理由一个或多个功能使用，并至少有一个功能修改的表了。首先，需要将它们分类为低活动、中活动及高度活动的表。基于事务率的范围，应该可以定义哪个表属于哪个类。例如，如果最繁忙的表每秒有 100 个插入与 600 个更新，它的事务率为每秒 700 个操作。如果最不稳定的表每秒有 40 个操作，事务率范围为 660 个操作（700-40）。将它分为三等分，即 220 个操作，把它们如下分类：

具有最多每秒 260 个事务的表是低活动率表。

具有每秒 261 个事务至 480 个事务的表是中等活动率表。

具有每秒 481 个事务至 700 个事务的表是高活动率表。

对高活动率表，应该使用分配了哈西锁或精细锁的表空间。如果唯一的活动是插入新的行，应该使用表及索引段上的多个空闲列表组，然后分配或者使用与手工分配区间联合的哈西锁，或者使用相关数据文件中 GC\_FILES\_TO\_LOCKS 参数的组因素分配精细锁。应该将中等活动率与低活动率的表分离到它们自己的表空间中，这些表空间相关的数据文件上具有哈西锁。

虽然以后会需要进行一些更好的调整，但最初应该选择使用三个简单的值，计算你应用到想要使用哈西锁的文件上的锁的数量。这些值将代表哈西锁的粒度，可以基于一些简单的比例选出。例如，对高活动率到中等活动率的数据文件，可以使用 1 到 10 的比率，对中等活动率到低活动率的数据文件使用相同的比率。这些粒度数被数据文件中块的数量除，以计算覆盖这个文件所需要的哈西锁的数量。使用所提到的比率，可以选择高活动率文件的粒度为 10，中等活动率文件的粒度为 100，低活动率文件的粒度为 1000。

对任何给定的文件，使用文件中的块数除以文件内容中活动级别的相应粒度数。表 39-7 显示了使用前表中的值，对粒度的每个级别计算不同大小示例文件的锁的数量。

表39-7 基于活动级别与数据文件大小的哈西锁分配

块的数量	哈西锁的数量		
	高活动率 粒度=10	中等活动率 粒度=100	低活动率 粒度=1 000
1 000 000	100 000	10 000	1 000
5 000	500	50	5
1 000	100	10	1

如果使用前述步骤，仍需要一个大数量的锁以覆盖数据文件，就具有其他的选项以试图减少它们。可以对属于中等活动率与高活动率表或所有公有表的表空间的数据文件使用可释放的锁。也可以试用一种不同的分区机制或在早已具有的功能分区上应用一个附加的分区类型。如果具有太多的锁，系统无法处理它们，或者由于太多的锁而使实例无法在合理的时间启动，你无疑需要使用上述这些技巧之一。

**注意** 如果增加了 PCM 与相关的 DLM 锁的数量，也许想要测试实例恢复的时间。锁的数量越多，DLM 重新配置的时间越长，特别是如果一个整个的结点失效，停止了所有的实例和 DLM 在这个结点上的处理时，更是如此。

如果用户具有可以在故障之后空闲时间的最大数，需要减少所分配的锁的数量以满足他们的恢复时间的需要。

### 39.5.2 部门与线型商务分区

如果具有一些用户，他们试图获取他们自己的数据子集，即使这些数据处于同一个表中，也需要使用部门与线型商务（LOB）分区。为扩展上节中的人力资源示例，公司需要有一定数量的部门，其中人员由一个本地人力资源部门管理，或需要一个 HR 组管理本地雇员，另一个 HR 组管理国际雇员。

如果存在这种固有的分区，可以对每个部门或 LOB 使用不同的实例。由于这个分区模式的本质，很有可能所有节点上的用户都几乎执行所有的 DML，及对它们自己的数据的查询。继而，对每个共享表，只需要极少数的 PCM 锁。达到这个目的的最好的方法是使用多个空间列表组，并为每个实例分配它自己的数据文件或数据文件集中的区间。有些情况下，你会发现在有些文件上只需要一个单独的锁，虽然有些行只是偶而被其他实例共享，但这不会提供足够好的性能。

### 39.5.3 物理表分区

如果不具有用户的固有子集，或有一个或多个所有用户都要修改的关键表，需要求助于表的物理分区。这不像是部门或 LOB 分区那样一流的解决策略，因为还必须控制用户怎样连接到数据库。使用部门或 LOB 分区，用户早已工作在不同位置或不同系统管理单元上，一个新的用户将属于这些组之一，并且可以被分配给相应的实例。为实现物理分区，需要一种机制以确保用户连接到正确的实例，依赖于他们需要存取数据的什么部分。

物理分区的一个例子是创建多个雇员表代替一个单独的表。对四个实例的数据库而言，需要拥有一个姓在字母 A~E 范围内的表，一个姓在字母 F~J 范围内的表，一个姓在字母 K~Q 范围内的表，一个姓在字母 R~Z 范围内的表。可以将这些表放在不同的数据文件中，甚至放在不同的表空间中。在 Oracle 7 中，需要使用分区视图及完全分离的表以实现物理分区。在 Oracle 8 中，可以利用分区选项建立一个具有多个分区的单独的表，尽管分区视图与物理分区表仍是可用的。

如果用户可以使用如表一样的分区范围进行分区，可以轻易地实现这个机制，这与部门或 LOB 分区极为相似。如果不同的人力资源部门雇员按照雇员的姓负责雇员，正是这种案例。一个分区模式与刚刚讨论过的一样进行工作，这个案例与 LOB 分区的最主要区别在于当查询需要报告所有的或至少超过一个分区的雇员情况时。如果使用 Oracle 8 的分区选项，Oracle 8 解决了这个问题，因为只需要查询一个单独的表。在 Oracle 8 之前，将需要使用 UNION 或 UNION ALL 操作创建的分区视图检索所有的记录。

如果配置了并行服务器数据库在运行这种查询时在多个实例上使用伺服器进程，在物理分区之上的查询可以是非常有效的。分区选项也允许使用多个实例上的伺服器越过所有的分区执行并行 DML 命令。在 Oracle 8 8.0 版本之前，不能执行并行 DML。

如果不能按表的标准分区用户，需要应用基于正在处理的数据，将用户发送到正确的实例上，这甚至需要用户为每个事务进行一次新的连接以便它们总是基于有趣的记录工作在要求的实例上。然而这在一个标准的客户/服务器体系结构中是不切实际的，使用三层或更多层的体系结构建立这种应用的模型是可能的。在多层体系结构中，事务处理（Transaction Processing, TP）监控器位于用户与数据库服务器之间，控制客户的 SQL 语句的路由，并将信息返回给用户。当一个请求没有指定实例时，TP 监控器也可以帮助负载均衡，如果个别的实

例发生故障，TP监控器可以将请求路由到替代结点。

如果可以实现一个物理表分区，也可以应用一个哈西 PCM锁模式。行的初始存储与后继操作将由相同的实例执行，所以可以在含有这个实例分区的数据文件上使用一个单独的锁。另外，不需要多个实例组，因为只有一个实例将创建新的行。然而，使用部门或 LOB分区，将需要更多的锁，即使另一个实例很少需要使用这些行。

#### 39.5.4 事务分区

如果不能如前面讨论的那样，将应用或表手工进行分区，也许需要试一试事务分区。事务分区需要非常复杂的应用，也许需要与 TP监控器一同使用。这一步需要每个事务发送到最好的实例处理它，基于事务的本质与所包括的表。如果所有的表都被分配给一个实例的锁覆盖，编写程序非常简单，但当一个事务包括的表分配了不同实例上的锁时，决策树会非常复杂，这个要求使这个途径成为一个艰难而昂贵的选择。

事务分区的一个好处是如果需要增加更多的实例支持增长，不需要对表结构进行修改。使用其他的分区模式，当增加一个新的实例时，通常需要对表或用户进行重新分区。使用事务分区，你修改程序而不是数据库。这听起来像是一项很繁重的工作，移动大量数据所需要的时间足够达到可用性的服务级协定了。然而，使用事务分区，可以在一个测试环境中预见到一个新事务时，进行程序代码的改变，并可用最小的停工时间实现。

#### 39.5.5 索引与可伸缩性考虑

当为数据库表决定最佳的分区选择时，也需要设计可能的索引。为减少硬盘竞争，强烈建议将索引放在与父表分离的表空间中，使用不同的硬盘存放它们的数据文件。为在并行服务器环境下达到这个目的，使用早已存在的允许表分区的表空间，会使系统管理员非常头痛。因而，在某些情况下，将表与它的索引放在相同的表空间中是非常有益的。Oracle分区选项也为在相同的表空间中，存储表的分区与它们的本地分区索引提供了很好的原因。使用这种方法，相关的表与索引分区可以同时联机或脱机。

需要预见的另一个问题是在多实例数据库中索引的处理与线性增长的主键值的使用相关。主键通常是一个数字值键，由 Oracle序列生成器或其他使用一个增长整数值的机制提供值，如果两个或多个实例使用这种主键机制向一个表中插入记录，它们也需要向相同的索引叶子块中插入条目，这会导致这个块的连续的 pinging。为避免这个问题，有一些可用的机制。

可以创建实例值的指定范围，将值乘以实例的数量，使用如下的方程式：

$$\text{instance\_number} * 10000000 + \text{sequence\_number}$$

这个选项需要修改代码语句以使用索引获取行。当不得不增加行时，如果想要检索它，需要对数据做相同的改变操作。如果不是创建这个条目的实例想要检索它，这并不像听起来的那样简单。

第二个选项是为每个实例使用一个不同的序列。有许多方法定义这种序列生成器，虽然它们都有自己特定的缺点。最简单的两种方法包括对不同的实例使用不同的下限与上限的生成器，或使用不同的开始值与非单元步进值的生成器。例如，考虑一个四实例数据库。

在第一个例子中，实例1将分配的范围是1~1 000 000，实例2的范围是1 000 001~2 000 000，实例3的范围是下一个1 000 000个数，实例4的范围是4 000 001至以上。实例1可以使用一个

序列生成器定义起始为1，增长步长为4，因而会被分配数1、5、9、13等等，与此相似，实例2的生成器起始为2，步长为4，值为2、6、10、14等等。虽然这种方法不会超出现有实例的允许数，但如果在应用增长需要一个附加的实例时，这很难与增长的实例协调。

**提示** 如果想要使用具有不同起始值的序列，还想要为附加的实例作计划，可以使步长比当前的实例数大一些。例如，在我们的四实例例子中，可以将四个序列的步长设为10，实例1将产生值1、11、21等等，与此相似，其他的实例将产生它们之间的值，2、3、4、12、13、14、22、23、24等等。这留下了5至10，增长为10的数用于新的实例使用。

最后一个选项是使用 Oracle8 的保留键索引特性，它将保留索引中每列的字节。这个方法的好处是在 SQL 语句的断言中，仍可以使用列的初始值，因为优化知道索引的结构。创建一个保留键值索引使用如下所示的 REVERSE 关键字。

```
CREATE INDEX busy_table_ix ON busy_table (id) REVERSE;
```

**警告** 虽然索引列中字节的保留将索引条目分配到整个索引叶子块中，并避免了在实例之间最有可能的叶子块竞争，但这个值的分配不是完全随机的。如果每个实例每次缓存100个数，两个不同实例从它们的缓存值中检索出的可用值的最后一个字节数可能是相同的。例如，数4022的最后一位字节的内部表示是27，与数4122的最后一位字节完全相同。如果两个实例增加它们的值，分别使用4022与4122同时分配给表，如果它们使用一个保留键值索引的话，它们也需要同时更新相同的叶子块。

应该使用保留键值索引帮助降低索引重建的频率——它们设计的目的，并且不希望它们会避免并行服务器数据库中的竞争。可以不恰当地分配 PCM 锁，那样你将会对数据库的性能感到失望。

### 39.5.6 序列生成器与多实例

如果决定在并行服务器数据库中使用序列生成器，需要清楚一些限制。序列生成器可以用于加速处理的一个方法是通过它们将一些数缓存入一个实例的 SGA 区。这允许大多数需要序列数的事务从内存中找到它，而不必执行一次硬盘读。当下一个数生成并保存时，缓存也可以避免锁定资源的需要。

序列生成器的 ORDER 选项用于确保数按顺序被分配。这意味着将给第一个要求数的进程一个最小的没有使用的数。在并行服务器数据库中需要使用这个特性，但 Oracle 只能提供从硬盘上的数据字典分配每个数的能力，它不能使用缓存的数，因为实例没有能力在它们自己之间传送缓存的数。一个失效的实例不能将它的缓存的数传送给另一个实例。因此，如果同时使用 CACHE 与 ORDER 选项定义了一个序列生成器，当以并行方式启动一个实例时，Oracle 将忽略 CACHE 选项，这很可能降低序列生成器的效率。

## 39.6 并行服务器创建的特殊考虑

不论是否正在建立并行数据库用以支持超越故障或可伸缩性能力，需要注意与基本结构有关的不同的选项。当准备建立数据库时，应该早已决定需要多少个实例，以及如果需要，准备怎样分区数据。也需要考虑作为 Oracle 数据库设计一部分的数据库结构的其他选项，如

ARCHIVELOG或NOARCHIVELOG状态，以及需要什么样的字符集支持所有需要的字符集。当已决定了这些设计时，就可以使用 CREAT DATABASE命令建立数据库了。

在表39-8中,CREAT DATABASE命令的选项与在建立数据库时应该考虑的问题一同列出。

表39-8 CREATE DATABASE命令选项

选 项	描述与使用
CONTROLFILE REUSE	只在计划重写控制文件时需要。建议不要使用这个选项，因为如果使用了错误的init.ora参数文件启动实例，它会破坏一个好的控制文件
LOGFILE	在单实例数据库中，这个参数允许创建需要的多个重做日志文件组，每个含有要求的多重拷贝的数量。使用这个选项创建的组将属于第一个重做线程，并将自动作为公有线程启动，如果想要，可以在启动另一个不同的实例并关闭这个实例之后，禁止这个线程，并重新允许它作为私有线程启动
MAXLOGFILES	设置可以为数据库创建的重做日志组的最大数量。这个数应该是所有想要创建的实例的重做日志组的数量之和。也许想要将这个值设得足够大，以支持在附加的日志组以免你需要对一个或多个实例增加重做日志组及支持附加的可用的实例
MAXLOGMEMBERS	设置可以分配给任何一个重做日志组的多重重做日志文件的最大数量。通常，日志组成员的数量在所有的实例重做上是相同的，但如果不相同，应该将这个设为一个日志组需要的最大数
MAXLOGHISTORY	决定将存储在控制文件中的归档日志文件条目的数量。当达到这个数时，当新的条目增加时，老的项目被取代。这些条目含有与归档日志名与内容有关的信息，用于自动介质恢复选项以指明它需要使用哪个归档日志文件。如果想要使用自动的介质恢复，应该将这个参数值至少设为在一个单独的恢复循环中所实例将创建的归档日志文件的数量
MAXDATAFILES	设置为数据库创建的数据文件的最大数量。它是一个典型地比单实例数据库要高的数，由于避免实例之间块竞争的分区分表与索引的需要。与使用MAXLOGFILES一样，应该考虑将这个值设为足够大，以容纳将来分区需要的增长
MAXINSTANCES	指明可以同时存取数据库的实例的最大数量。如果为超越故障使用一个或多个实例，需要使用这个总数，即使在通常情况下不需要用户连接它们。这是另一个需要考虑到将来增长设置的参数
[NO]ARCHIVELOG	允许决定数据库在创建时是否处于 ARCHIVELOG模式。虽然建议对大多数产品数据库使用 ARCHIVELOG模式，通常不必要立即将它置为这种模式。缺省值是NOARCHIVELOG
EXCLUSIVE	这是一个选择性关键字，并将在 Oracle8中废除。它加强了第一个实例不能以共享或并行方式打开的事实。建议忽略它
CHARACTER SET	指明了数据库用于存储数据的字符集。这个参数没有特定于并行服务器的特殊的特性
DATAFILE	命名最初组成 SYSTEM表空间的数据文件或数据文件集。这个参数没有用于并行服务器的特殊的特性。与此相似，对已命名数据文件的 AUTOEXTEND选项在单实例与多实例数据库中是完全相同的

需要建立一个SQL脚本文件以包含这条命令，及其他为并行服务器使用用户化数据库的命令。这些包括执行下述步骤的命令。

至少为每个实例增加一个回滚段。

为每个附加的实例增加一个重做线程。

启用重做线程。

运行catparr.sql脚本。

## 39.7 怎样监控与调整并行服务器

需要知道一些特定的动态性能表，以监控并行数据库的性能。在用于任何 Oracle数据库的标准的调整步骤之外，还需要使用这些调整工具与调整步骤。

对调整Oracle并行服务器非常有用的一些动态性能表使用称为 catparr.sql的脚本建立。作为SYS用户运行这个脚本，创建表并当增加新的想要追踪的段或区间时更新它们，在 Oracle动态增加想要监控用于数据库映射视图的区间位置之后，也需要运行这个脚本。

提示 应该养成在开始任何监控或调整工作之前，将运行这个脚本当作理所当然的事的习惯。这样，可以确保当检查需要的各种各样的动态性能表时，看到的是最当前的信息。

在Oracle 8中，下面几节中讨论的表，与大多数其他动态性能表一样，具有一个全局版本显示所有活动实例的各自的值。一个实例数列指明了数据正在报告的是哪个实例。为使用这些全局表，需要在表名前包括一个g。程序清单39-7显示了在全局视图V\$VERSION上所进行的查询。

清单39-7 查询一个全局动态性能表

```
SQL> SELECT * FROM gv$version;
```

```
INSTANCE_ID BANNER
```

```
-----
1 CORE Version 4.0.2.0.0
1 NLSRTL Version 3.3.0.0.0
1 Oracle8 Server Release 8.0.3.0.0
1 PL/SQL Release 3.0.2.0.0
1 TNS for Solaris: Version 3.0.2.0..0
2 CORE Version 4.0.2.0.0
2 NLSRTL Version 3.3.0.0.0
2 Oracle8 Server Release 8.0.3.0.0
2 PL/SQL Release 3.0.2.0.0
2 TNS for Solaris: Version 3.0.2.0.0
```

### 39.7.1 V\$LOCK\_ACTIVITY

需要监控的第一个表是V\$LOCK\_ACTIVITY。这个表显示了在一个单独的实例中，自它启动以来所发生的锁变化的数量和类型。虽然真实的数量自身并没有任何意义，因为每个系统都将以不同的速度处理锁与相关的活动，但可以使用它们比较不同实例的性能，并比较一个特定实例不同时期的行为。在V\$LOCK\_ACTIVITY表上的查询将产生与清单39-8相似的输出结果。

清单39-8 查询V\$LOCK\_ACTIVITY

FROM	TO	ACTION	COUNTER
NULL	S	Lock buffers for read	3595
NULL	X	Lock buffers for write	8111
S	NULL	Make buffers CR (no write)	2763
S	X	Upgrade read lock to write	1046
X	NULL	Make buffers CR (write dirty buffers)	92
X	S	Downgrade write lock to read (write dirty buffers)	1220
X	SSX	Write transaction table/undo blocks	907
SSX	NULL	Transaction table/undo blocks (write dirty buffers)	0
SSX	S	Make transaction table/undo block available share	0
SSX	X	Rearm transaction table write mechanism	907

清单39-8显示了在一个实例中发生的PCM锁变换的不同类型，同时还有变换发生的次数。FROM与TO列中的项目显示了锁的状态：

NULL——不是当前分配给实例的。

S——共享模式。

X——排它模式。

SSX——子共享排它模式。

共享锁用于查询表与索引块，当需要更改块竞争时使用排它锁。回滚段使用子共享排它锁，因为每个回滚段都由一个实例在启动时获得，只有这个实例可以向它写，因而不需要实例放弃写向它的回滚段块的独占能力。然而，其他的实例也许需要读回滚段信息，并当这些读在处理时，使用子共享排它锁在拥有的实例上覆盖回滚段块。应该看到从X到SSX与从SSX到X的转变的两个计数含有相同的值，因为任何借给另一个实例用于读目的的回滚段都不得不返回它拥有的实例。可以忽略从SSX到空与从SSX到S的转变的项目，因为它们来自于在系统启动时或表空间管理活动时所执行的操作。

如果能察觉到PCM锁变换与inging正导致性能问题，或如果想要预先活动并定期监控PCM锁的状态，V\$LOCK\_ACTIVITY表正是你应该开始的地方。你将总会看到一些块变换活动，因为一个实例在实例启动时拥有所有的PCM锁在NULL状态，在实例可以做任何工作之前改变实例需要的锁的状态。应该将注意力放在锁返回为NULL状态的锁转变的值，及回滚段锁活动的值上。转变为NULL状态只发生在不是你监控的实例需要使用这个锁时。如果这些转变的计数高并持续增长，应该猜想到pining是导致任何数据库性能下降的一个原因。你将需要察看其他的并行服务器动态性能表以确定包括哪个块与段。

在从X到SSX与从SSX到X的成对的变化表明回滚段数据正被ping。如果这些数值较高，应该考虑使用GC\_ROLLBACK\_LOCKS增长回滚段锁的数量。如果问题仍然存在，可能需要转变为精细锁。

### 39.7.2 V\$BH

V\$BSH表显示实例的数据库缓冲缓存中的当前内容。它提供了每个缓存中的文件与块的标识数，及该块上的PCM锁的状态。一个查询，连同部分列表一起展示，如清单39-9所示。

清单39-9 查询V\$BSH表

```
SQL> SELECT file#, block#, class#, status, xnc FROM v$bh;
```

FILE#	BLOCK#	CLASS#	STATUS	XNC
8	438	1	XCUR	0
8	467	1	SCUR	0
8	468	1	SCUR	0
8	468	1	SCUR	0
9	192	1	CR	0
9	198	1	CR	2
9	201	1	XCUR	10

列的状态是如表39-9所示的七个值之一。通常在实例运行一段以后，只看到行具有SCUR、SCUR或CR值。此时，所有的缓存至少已使用一次，任何介质或实例恢复应该已经完成。偶而会发生一个缓存正在使用从一个数据文件读出的数据装填。

表39-9 在V\$BH中显示的缓存状态

状 态	解 释
FREE	这个缓存当前没有使用
XCUR	这个缓存中的块由一个排它 PCM锁覆盖
SCUR	这个缓存中的块由一个共享 PCM锁覆盖
CR	这个缓存中的块由降级的 XCUR或SCUR 锁覆盖
READ	块正在从硬盘读入缓存
MREC	缓存中的块处于介质恢复状态
IREC	缓存中的块处于实例恢复状态

提示 如果发现缓存处于MREC或IREC状态，应该立即退出数据库。如果缓存处于任何一种恢复状态，将得不到数据库性能的任何有用的信息。而且会话几乎正在竞争资源，这些资源是由恢复进程以实时方式完成工作的最好利用的资源。

经常会发现在内存中有许多缓存处于 CR状态，同时在V\$LOCK\_ACTIVITY中有过多的X到NULL或X到S的转变。XNC列显示了特定块上X到NULL的转变，该列上的非零值代表这个块当前正被 ping，可以从FILE#与BLOCK#列找出包含了哪些块。通过使用另一个实例的V\$BH表或Oracle 8的GV\$BH表，与另一个实例的缓存中的块相比较，可以找出哪些块正在哪些实例之间被 ping。有些情况下，可能在其他的实例中找不到相同的块，这表明发生的是假 ping，因为锁已被释放，但并不需要这些块本身。

为解决真正的或假的pinging的问题，知道包括哪个表甚至是这些表中的哪些值很有用处。可以使用ROWID伪列查询表，以查看哪些块被包括。清单 39-10是7.0版数据库中这种查询的一个例子。使用这种方法找不到索引项，最多能找出索引的名字。

清单39-10 检查特定块的行内容

```
SQL> SELECT * FROM DEPT
2  WHERE SUBSTR(rowid,1,8) = '00000201'      -- the block id number
3  AND SUBSTR(rowid,15,4) = '0009';          -- the file id number
```

DEPTNO	NAME	LOCATION	MGR_ID
14	Human Resources	Fairview	198
22	International Sales	New York	1356
...			

为找出与V\$BH中指明的特定块相关的段，可以查询 DBA\_EXTENTS视图，但使用V\$CACHE与V\$PING视图会更简单。

### 39.7.3 V\$CACHE与V\$PING

V\$CACHE与V\$PING这两个动态性能表与V\$BH视图基于相同的缓存信息，但它们包括三个附加的列。在清单 39-11中，可以在V\$CACHE上所进行的查询输出中看到这些列。列指明了块的名字、类型及块在缓存中属于哪个段的段 ID号。这些视图在创建之后就不会含有新增加的段的信息，除非运行 catparr.sql。如果NAME、KIND与OWNER#含有空值，再次运行脚本，使用新的信息填充这个视图。

V\$CACHE与V\$PING的不同之处在于：前者为每个高速缓存中的内存含有一项，而

V\$PING中为有可能被ping的内存含有一行。这由在XNC列拥有一个非零值的缓存决定，指明它含有的块至少有一个X到NULL的转变。

清单39-11 查询V\$CACHE

```
SQL> SELECT name, kind, owner#, file#, block#, status, xnc FROM v$cache;
```

NAME	KIND	OWNER#	FILE#	BLOCK#	CLASS#	STATUS	XNC
EMP	TABLE	14	8	438	1	XCUR	0
EMP	TABLE	14	8	467	1	SCUR	0
EMP	TABLE	14	8	468	1	SCUR	0
EMP	TABLE	14	8	468	1	SCUR	0
DEPT	TABLE	14	9	192	1	CR	0
DEPT	TABLE	14	9	198	1	CR	2
DEPT	TABLE	14	9	201	1	XCUR	10
...							

也可以使用V\$FALSE\_PING动态性能表以指明潜在的假ping。这个表含有与V\$PING相同的列，但只含有极有可能多次被假ping的块的行。然而，Oracle不保证在这个表中列出的行即将被假pinging，或这个表含有被假ping的每一个块。仍然应该检查其他的缓存以找出上述的假pinging的确切证据，可以使用讨论过的视图的全局版本轻易实现这个目标。

#### 39.7.4 并行服务器的调整策略

多实例数据库并行实例组件的基本调整策略含有下面四个步骤：

- 1) 决定是否有过多的碰撞询问。
- 2) 指出什么正在被碰撞询问。
- 3) 解析这个碰撞询问是真还是假。
- 4) 解析这个碰撞询问问题。

通过使用前面所讨论过的并行服务器动态性能表，可以找出pinging的总数，正被ping的对象，是否一个或多个实例正在ping相同的块。有了这些信息应该怎么做呢？

对所有的调整活动，如果用户不需要进行数据库性能的调整，只需要进行调整。通过不时地监控V\$LOCK\_ACTIVITY的值，可以发现任何与事务响应时间成反比例增长的值。如果发现这样的值，需要从V\$BH, V\$CACHE或V\$PING中查出对此负责的块。

**警告** 当响应时间或相关事务处于峰值时，使用动态性能表要小心。如果现在查看这些值，会发现它们比一小时前要大多得多，因为Oracle在实例的生命期中累积统计值。在任何数据库做有用工作的时候，它的一些性能统计会增长以反映这个变化。对多个实例需要使用一个与单实例数据库相同的监控机制，这包括固定周期地收集统计数据，以便监控特定时期的变化率而不是依赖于不变的值。

**提示** 如果使用的是Oracle 8，使用全局动态性能表收集与实例有关的统计会很容易。如果从Oracle 7数据库升级为Oracle 8，需要将用于查询V\$\_表的任何脚本替换为查询GV\$\_表。

如果在动态性能表中发现真正的pinging，有一些解决选项。首先，如果只有两个实例，可以考虑将它们的工作负载与用户合并为一个实例。如果由于资源的限制或由于包括了不止

一个的实例，不可能进行这种合并，需要考虑使用前面讨论过的分区选项。目的是将多数对一个给定块的存取限制为只有一个实例存取。这需要仔细检查 ping 块的行以确定分区选项。

如果发现正在发生一个假的 ping，最简单的解决策略是对涉及的数据文件增加更多的哈希锁，或将哈希锁转换为精细锁。前一种将涉及创建过多的锁，以至于当前内存无法处理，在这种情况下，需要使用精细锁。如果这些附加的精细锁与现存的哈希锁合在一起仍然组成了过多的锁，以至于系统无法处理，需要试试将一些由哈希锁覆盖的数据文件释放为精细锁。

如果使用这些方法仍不能解决 ping 问题，这可能意味着数据库设计不合理，需要返回来重新进行规划，并使用前面讨论过的技术找出一个更好的设计方案。Oracle 7.3 以前的版本没有提供精细锁与相关益处，会使一些数据库不适宜使用并行服务器选项。现在，有了精细锁，使用 TP 监控器与 Oracle 8 附加的功能，如全局视图与反向键值索引，应该有能力为超越故障或可升级目的（如果不是两者兼备）建立或监控一个多实例数据库。

### 39.8 附加的提示与注意事项

除非正使用一种非常罕见的操作系统，诸如 nCUBE 与 DEC 的 VAXCluster 的派生系统，必须将 Oracle 并行服务器数据库放置在裸盘上。在 Oracle 8i 版本中，控制文件不再是固定长度的，其中存储了归档重做日志项的状态，用于 Oracle 的恢复管理器使用，既使不使用恢复管理器工具用于备份及恢复。可以通过设置初始化参数 CONTROL\_FILE\_RECORD\_KEEP\_TIME，将其设为一个较小的数，减少存储的项的数量。

也可以通过自己的操作，改变控制文件的大小。例如，可以设置实例的参数文件包括一个 DB\_FILES 的值，它大于在 CREATE DATABASE 命令中设置的 MAXDATAFILES 的值。如果为数据库增加了比 MAXDATAFILES 要求的数量更多的文件，控制文件将增长以容纳它们。应该确保将控制文件放在足够大的裸盘上，因而可以含有一个比 CREATE DATABASE 命令产生的更大的控制文件。如果硬盘管理系统允许裸盘分区的内容比分区容量大，这更为重要。

虽然第一阶段的高速缓存融合减少了查询中使用的块在实例之间传递的开销，但并不能总结出可以避免按实例对数据进行分区的进程，在将块发送给请求的实例时，仍然会有消耗。集成的分布锁管理仍然必须在实例之间传递锁请求，簇交互必须在实例之间传递块映象，而不只是短信息。如果设计需要通过簇交互传送过多的块，会超过簇的带宽，因而降低系统的效率。

也有一些厂商不提供 Oracle 用于传送块映象的交互。在使用高速缓存融合之前，应该检查安装与配置手册。

如果具有一个不需要用户注册到一个指定实例的数据库设计，可以使用新的在 Net 8 文件中定义服务的选项。服务由两个或多个实例组成，在 Net 8 连接串中使用服务的名字而不是一个单独的实例的名字。Net 8 将使用由监听器进程维护的与服务相关的每个实例的统计，指出最不繁忙的实例，用户的注册将路由到这个实例。这允许在实例之间维持动态工作负载平衡。如果运行了几个实例，当然可以创建多个服务，每个服务覆盖实例的一个子集。这允许提供不同用户与实例集的负载平衡。