Oracle 技术白皮书
2009 年 7 月

使用 Oracle In-Memory Database Cache 提高 Oracle 数据库处理速度

ORACLE

1. 简介

Oracle In-Memory Database Cache 可以加快业务流程速度、支持实时业务智能、促进面向客户的应用程序实现个性化。

Oracle In-Memory Database Cache (IMDB Cache) 是一个 Oracle 数据库产品选件，非常适于将 Oracle 数据库中对性能影响极大的部分缓存在应用程序层中。使用 IMDB Cache 可提高应用程序的响应速度和吞吐量。IMDB Cache 由三个重要技术组件构成 — Oracle TimesTen In-Memory Database (TimesTen)，用于应用程序层实时数据管理；缓存技术，用于将频繁访问的表从 Oracle 数据库服务器缓存到应用程序层并维护缓存数据的一致性；事务型数据复制组件，用于确保跨层的高可用性。

TimesTen 是一个内存优化的关系数据库，它为性能要求极高的系统提供了快速的响应时间和极高的吞吐量。其设计目标是为了在应用程序层运行（靠近应用程序），也可与应用程序一起在进程中运行。TimesTen 数据库可用作关系数据库和/或某个 Oracle 数据库的缓存。

应用程序可以在 TimesTen 中创建和管理数据库表，也可以将 Oracle 数据库中频繁访问的内容缓存在 IMDB Cache 中。缓存表和非缓存表可共存于同一个内存数据库中，并且这些表都是可持久存在和可恢复的。应用程序使用 ODBC、JDBC、Oracle Call Interface (OCI) 或 TTClasses 通过 SQL92 或 PL/SQL 对缓存数据和非缓存数据执行查询和更新，也可通过 Pro*C 执行这种查询和更新。

缓存网格由一组共同管理应用程序缓存数据的 IMDB Cache 构成，可用于改善性能和容量，实现水平可伸缩性。缓存数据分布在网格成员之间，缓存网格为应用程序提供位置透明性，从而使应用程序可以有效地使用所有网格成员中缓存的全部数据。通过联机添加（和删除）网格成员，缓存网格可以实现增量可伸缩性。缓存网格可以维护缓存网格成员与 Oracle 数据库之间缓存数据的一致性。

IMDB Cache 跨应用程序层和数据库服务器层管理数据的可用性。无论何处发生故障，它都能确保高可用性，并确保不丢失事务。无论故障是发生在一个缓存节点、一个 Oracle RAC 节点，还是发生在网络级，甚至是整个 RAC 集群都发生故障，均可保证高可用性，并保证不丢失事务。

TimesTen 和 IMDB Cache 在实时企业和响应时间要求极短的行业中部署具有极佳表现，包括网络通信服务、运营支持系统、联系中心、机票预订系统、指挥和控制系统以及证券交易系统。全球范围内的数百家公司在生产应用中使用了 TimesTen 和 IMDB Cache，这些公司包括 Alcatel-Lucent、Amdocs、Aspect、Avaya、Bombay Stock Exchange、Bridgewater Systems、BroadSoft、Cisco、Deutsche Börse、Ericsson、JP Morgan、NEC、NYFIX、Smart Communications 和 Sprint。

2. 应用程序层缓存

通常，应用程序层缓存用于缩短数据访问延迟并减少后端数据库的负载。

已经开发了各种缓存技术来改善数据库访问性能或减少后端数据库服务器上的争用。快速响应时间对实时应用程序和面向客户的应用程序尤为重要。另外，对于那些用户团体持续增长的应用程序（如软件托管服务、电子商务站点或电信服务），降低后端数据库的负载也很重要。

至于缓存哪些信息以及将信息缓存在何处，有多种选择，每种选择都有其优缺点。已经开发的一些缓存技术包括：

- *查询结果缓存。*这通常在应用程序层进行并由一个专用软件管理，该软件向应用程序隐瞒了缓存的存在。这种情况下，缓存软件自动保存提交给数据库系统的查询结果。如果某个查询与之前提交的一个查询完全匹配（包括相同的参数值），则确认一个缓存命中并由缓存提供服务。这种缓存技术的优点是简单，适于可能反复提交相同查询的访问情况。但它的用途有限，因为无法处理对缓存内容的查询处理。

- *对象关系映射工具缓存。*通过提供对象与关系数据之间的透明映射，对象关系映射工具（O/R 映射工具）向面向对象的编程人员隐藏了关系数据库。一旦关系数据映射为一个对象表示，O/R 映射工具就会对其进行缓存，直到不再需要该数据或该数据过期。通过 O/R 映射工具进行缓存是避免编程语言对象模型与数据库关系模型之间高开销映射的一种常用技术。

- *对象缓存。*这里的“缓存”一词某种程度上是个误称，因为放入这些缓存中的对象不一定是存储在其他地方的对象的子集。这些“缓存”是独立于对象源的对象信息库。它们通常对应用程序不透明。应用程序从这些缓存中“put”、“get”、“insert”和“delete”对象。市场上提供这类缓存的产品很少，并且它们所支持的功能水平也有所不同。这些缓存可能是完全的内存驻留，也可能是回到磁盘或其他数据管理系统。某些产品提供了并发控制，某些产品提供了网络中多个节点上的透明分布，还有一些产品提供了高可用性。

Oracle In-Memory Database Cache 具备完整的关系功能和 SQL 功能，能够自动维护数据与 Oracle 数据库的一致性，并能维护实时性能。

Oracle In-Memory Database Cache (IMDB Cache) 采取了一种独特的方法，支持将表或表片段从 Oracle 数据库缓存到应用程序层。通过扩展的 SQL 语法对表片段进行描述，并将其缓存在 Oracle TimesTen In-Memory Database (TimesTen) 中。应用程序使用 SQL、PL/SQL 或 Pro*C 读取和更新缓存数据，IMDB Cache 自动将 Oracle 数据库中的更新传播到缓存中，并将缓存中的更新传播到 Oracle 数据库中。

可以将一组 IMDB Cache 配置成一个缓存网格。缓存数据分布在网格成员之间，缓存网格为应用程序提供位置透明性和并发控制，从而使应用程序可以有效地使用所有网格中缓存的全部数据。随着应用程序性能或容量需求的增加，还可向缓存网格添加更多节点，但不会中断服务。因此，IMDB Cache 除了为应用程序提供了关系数据库的完整功能，还提供了增量可伸缩性及位置透明性，并能自动维护缓存与 Oracle 数据库的一致性，还提供了实时的内存数据库性能。

IMDB Cache 方法有两个主要优点，有助于提高整体性能。第一，TimesTen 的内存中体系结构消除了应用程序层与数据库服务器之间的通信，因此使用 IMDB Cache 的应用程序可以大大缩短响应时间并显著增加吞吐量。第二，该方法减少了后端数据库的负载，从而提高了所有应用程序的总吞吐量。

它所提供的关系数据库的全部优势、实时性能、增量可伸缩性及自动缓存管理，都是 IMDB Cache 的独特功能。它最适于缓存 Oracle 数据库中对性能影响极大的部分，支持读取和更新缓存数据，并且能够自动管理数据一致性。

本文的后续部分将简单介绍 Oracle TimesTen In-Memory Database（可在 [1] 中找到更详细的信息），还将介绍 Oracle In-Memory Database Cache 是如何缓存和管理数据的，并且列举了几个说明性的使用缓存的情况。

3. Oracle TimesTen In-Memory Database

TimesTen In-Memory Database 通过标准 API 提供对数据和关系功能的事务访问。

Oracle TimesTen In-Memory Database 是一个内存优化的关系数据库，它通过 ODBC、JDBC、Oracle Call Interface (OCI) 和 TTClasses[1] API 支持 SQL92 和 PL/SQL，同时也支持 Pro*C/C++。通过对标准接口和流行的 Oracle 接口的支持，TimesTen 确保现有应用程序均可方便地使用它。

虽然 TimesTen 对主内存中的数据进行操作，但 TimesTen 数据库在软件、硬件或电源故障时仍然可持久存在且可恢复。通过检查点和将其记录到磁盘可确保持久性。应用程序可以为其事务选择 ACID 属性，但还提供了其他更优选项以实现更高性能。TimesTen 提供了一个基于成本的查询优化器，应用程序可通过它查看和影响查询计划。可以将 TimesTen 作为应用程序链接库提供，也可以通过客户端/服务器选项提供。通过客户端/服务器选项访问 TimesTen 时，对 TimesTen 的每个请求都会产生进程间通信开销，即使应用程序与 TimesTen 服务器运行在同一台计算机上也是如此。相反，TimesTen 与应用程序链接后，对 TimesTen 的请求就只是开销可忽略不计的本地调用，应用程序与 TimesTen 之间的数据传输就只是开销较小的内存复制操作。通过复制提供了高可用性。还提供了大量实用程序，包括一个交互式 SQL 实用程序，以及一个用于数据库开发和缓存配置、联机备份和恢复以及批量加载的图形化工具。还通过可编程的 API 提供了数据库维护操作。

运行时，主内存中驻留了数据库的一个副本。在一个共享内存段中管理这个副本，所有连接到该数据库的进程均可访问该副本。图 1 显示了 TimesTen In-Memory Database 系统的体系结构。

针对数据所驻留的内存对 Oracle TimesTen In-Memory Database 数据结构和算法进行了优化。

TimesTen 的数据结构和存取算法利用数据库所驻留的内存实现了性能突破。与完全基于磁盘缓存的数据库相比，TimesTen 的内存优化体系结构使用的 CPU 周期要少很多，这是因为它省去了用于管理内存缓冲区并处理多个数据位置（磁盘和内存）的开销。

Oracle TimesTen 内存优化的性能由支持事务属性、持久性机制以及系统故障恢复所完善。有各种方法可用于锁定、多用户隔离和日志记录，从而满足各种应用情形（从临时查找缓存到核心事务性财务交易和电信计费系统）。

[1] TimesTen C++ Interface Classes (TTClasses) 是一个 C++ 类库，提供了关于最常用 ODBC 功能的打包。它比 ODBC 更易于使用，并且在保持快速性能的同时促进了最佳实践。

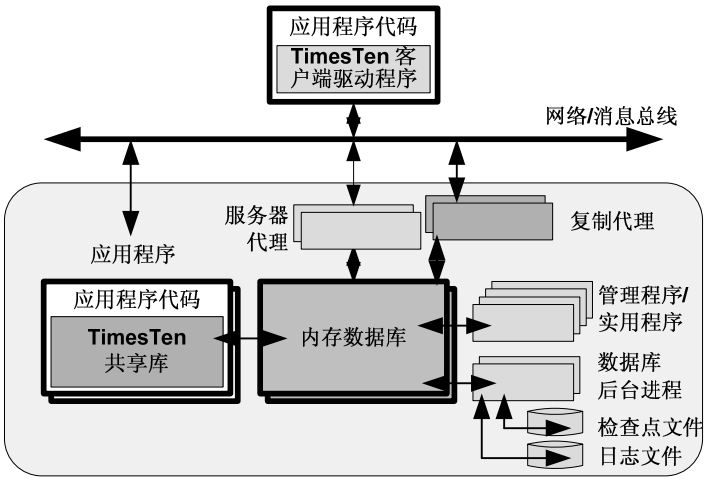

图 1. TimesTen 体系结构

TimesTen 数据库可持久存在且可恢复。

TimesTen 通过将已提交事务中的更改记录到磁盘并通过检查点定期更新数据库的磁盘映像实现了持久性。应用程序可以对日志写入磁盘的时间进行配置（要么与事务的结束时间同步，要么一直延迟到事务完成之后），从而实现更高的性能。在许多情况下，高吞吐量要比同步记录重要，尤其是当事务的货币价值很低或事务数据的生存时间较短时（如，在每几秒钟传输一次移动电话位置的网络中跟踪移动电话的位置时）。

TimesTen 允许应用程序跟踪特定表的变更。这在应用程序对特定事件的发生非常敏感的环境中是很有用的。例如，应用程序可能希望知道某支股票的价格何时超出给定的阈值。当这一变更通知特性不仅能跟踪基表的变更，而且还能跟踪物化视图的变更时，该特性尤为有用。

3.1 Oracle TimesTen 的性能

仅靠添加硬件无法实现极低的响应时间。TimesTen 凭借其独特的体系结构提供了极低的延迟。

TimesTen 凭借其特有的内存中体系结构实现了微秒级的响应时间。使用 TimesTen，读取一条数据库记录的事务所用时间不到 5 微秒，更新或插入一条记录的事务所用时间也小于 15 微秒。

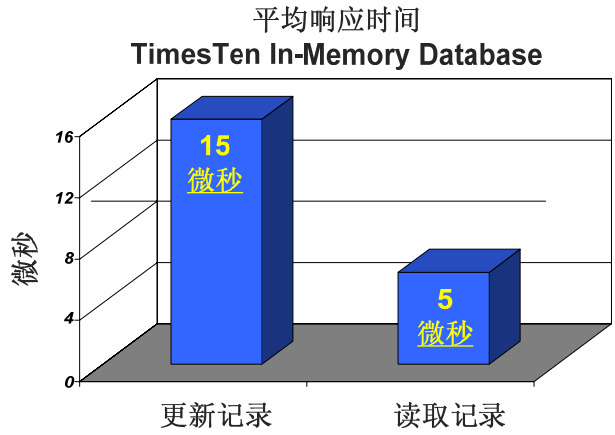

图 2. TimesTen 的响应时间

图 2 显示了应用程序在运行 Oracle Enterprise Linux 5.2 的双 CPU Intel E5450 (8-way@3GHz) 上执行读取和更新事务的响应时间。

4. 使用 Oracle In-Memory Database Cache 进行数据缓存

IMDB Cache 包含 Oracle 数据库表的子集。

使用 IMDB Cache 可将 Oracle 数据库中表的子集缓存到应用程序层。可对缓存表进行更新，并且 IMDB Cache 在 Oracle 数据库与缓存之间同步数据。

管理缓存数据的数据库引擎为 Oracle TimesTen In-Memory Database。其功能有所增强，能够加载和同步缓存数据。与 IMDB Cache 关联的后台进程之一是 Cache Agent，它管理部分数据同步。图 3 显示了 IMDB Cache 的体系结构。

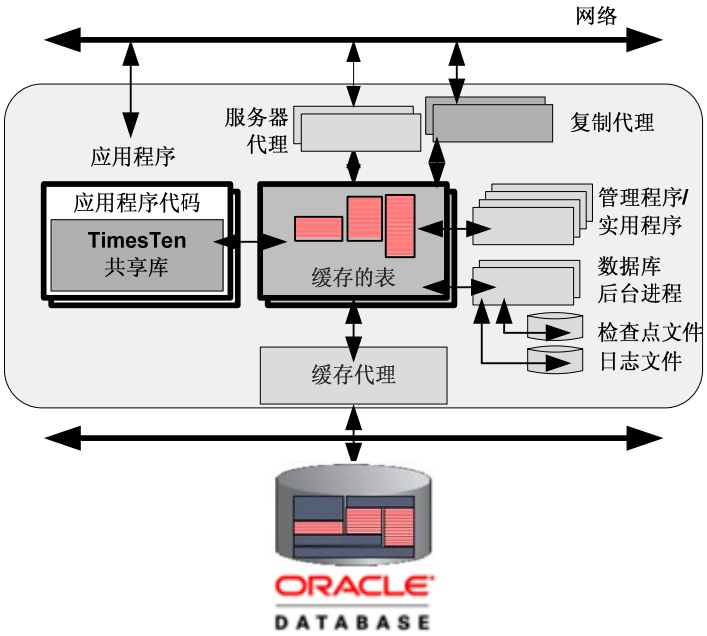

图 3. IMDB Cache 的体系结构

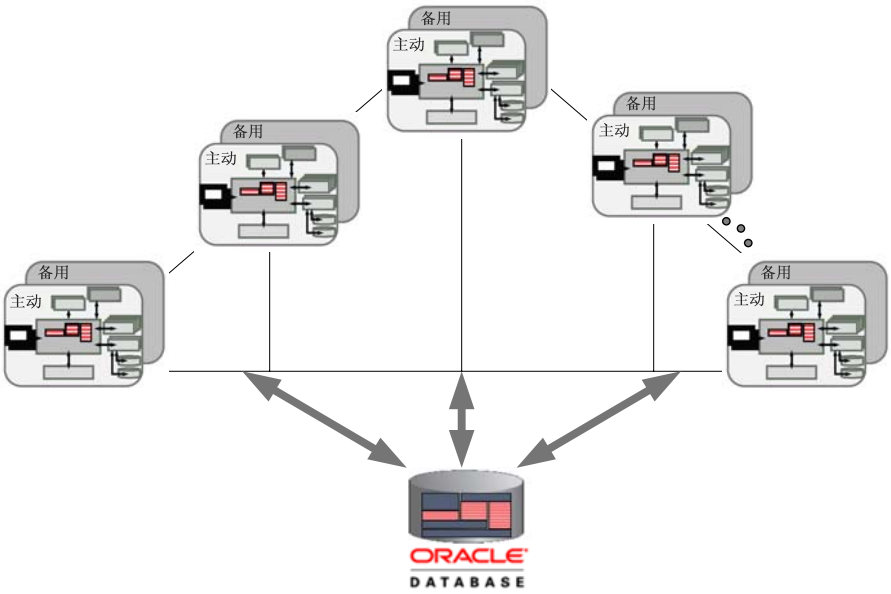

图 4. 包含五个复制网格成员的缓存网格

缓存网格 是共同管理应用程序数据的 IMDB Cache 的集合。缓存网格由一个或多个受 IMDB Cache 支持的网格成员组成。网格成员缓存来自中央 Oracle 数据库和真正应用集群 (RAC) 的表。缓存数据跨无共享存储的多个节点（即 IMDB Cache）分布。缓存网格确保节点间数据的一致性。网格成员可以复制。

图 4 显示了由五个复制网格成员组成的缓存网格。可以逐渐增加网格成员，而不中断操作。必须与 IMDB Cache 一起使用的复制配置为主动/备用对配置。

4.1 定义缓存的内容

使用扩展 SQL 语法定义 Oracle IMDB Cache 的内容。

缓存组 是一组 IMDB Cache 表，它对应于一组频繁使用的 Oracle 数据库表，这些表通过外键约束相关联。使用 SQL 语法定义缓存组，并从 Oracle 数据库表中选择要缓存的列和行。用户可以通过编程方式或者通过交互式 ttIsql 实用程序定义缓存组。

示例：

假设 Oracle 数据库中存在以下表：

- Customer(CustId, Name, Age, Gender, StreetAddress, State, ZipCode, honeNo)
- Order(CustId, OrderId, PurchaseDate, Amount)
- CustInterest(CustId, Interest)

应用程序可能希望对 2009 年 1 月以来下单客户的配置文件进行缓存。为达到此目的，它可能定义以下两个缓存组：

- 第一个缓存组包含上述三个表的部分内容，也就是只包含 2009 年 1 月以来的下单客户，并且这些客户居住在美国的太平洋地区。而且，应用程序可以选择只缓存这些表的部分列。例如，它可能缓存以下列：
  - Customer(CustId, Name, Age, Gender, State)
  - Order(CustId, PurchaseDate, Amount)
  - CustInterest(CustId, Interest)
- 第二个缓存组与第一个缓存组包含同样的信息，但是其中的客户居住在美国的山区。

这两个缓存组可以缓存在运行 IMDB Cache 的不同节点中。

IMDB Cache 使用的另外一个概念是缓存实例。*缓存实例* 是唯一可标识的相关记录的集合，用于建立一个复杂对象的模型。正如下面将要介绍的，缓存实例构成了缓存加载和缓存老化的基本单位。在上面的示例中，Customer、Order 和 CustInterest 三个表中属于给定客户 ID (CustId) 的所有记录都属于同一个缓存实例，这些记录通过外键约束互相关联。CustId 唯一标识缓存实例，因此我们将 CustId 称为*缓存实例键*。

TimesTen 与 Oracle 数据库支持相同的数据类型。

除了支持自己的数据类型外，TimesTen 还支持与 Oracle 数据库相同的基本数据类型，因此无需将 Oracle 数据库数据类型映射为 TimesTen 数据类型。但是，可能需要将 Oracle 数据库数据类型映射为更高效的 TimesTen 实现。例如，应用程序可将 Oracle 数据库的 NUMBER 数据类型映射为 TimesTen 的 INTEGER 数据类型。

注意，应用程序开发人员可以在内存缓存表上创建索引。内存缓存索引可能与 Oracle 数据库的索引相匹配，也可能不同。应用程序设计人员可以利用 TimesTen 的灵活性在同一个表上创建多个索引，可以对多列定义索引。

4.2 加载数据和管理缓存

应用程序必须确定如何将缓存组数据加载到 IMDB Cache 以进行处理。可使用以下方法加载数据：

- 显式加载。这可通过下列几种方式完成：
    - 一次加载整个缓存组。如果一个缓存可以容纳整个缓存组的内容，则适用该方法。还能够卸载整个缓存组。
    - “按 WHERE 子句”加载缓存实例。这种情况下，WHERE 子句用于描述应该进入缓存的部分缓存实例。应用程序也可“按 WHERE 子句”卸载缓存实例。
    - “按 ID”加载缓存实例。这种情况下，使用缓存实例 Id 列表指定应放入缓存的缓存实例。应用程序也可“按 ID”卸载缓存实例。

- 动态加载。该方法用于加载缓存实例。当缓存组太大以至于缓存中无法容纳时，动态加载非常有用，这时缓存中只保留应用程序的工作集。这种情况下，根据缓存缺失自动加载构成缓存实例的记录，也就是说，SQL 语句[2]在缓存中找不到请求的数据时就加载含有该数据的记录。如果缓存实例已经在缓存中，则由该缓存直接处理 SQL 语句。

  动态加载通常与自动*缓存老化* 一起使用。当超过缓存容量时，自动将老化的缓存实例从缓存中换出。IMDB Cache 支持*基于使用情况的老化* 和*基于时间的老化*。当超过缓存容量时，基于使用情况的老化使用 LRU（最近最少使用的）方案将老化的缓存实例换出。基于时间的老化在缓存中授予缓存实例一定时间的*生存期*，这种老化需要缓存组的某个表含有时间戳列。时间戳列的值由应用程序管理。只要缓存实例的时间戳值加上生存期不超出当前时间时，缓存实例就能保留在缓存中。注意，缓存老化可独立于动态加载单独使用，实际上，可以和不是从 Oracle 数据库缓存的常规 TimesTen 表一起使用。

  应用程序可以选择对一些缓存组实施老化，而其他缓存组则保留。例如，应用程序可能希望在缓存中始终保留目录信息，但又希望在用户连接到应用程序时才按需加载用户的配置文件，在用户断开连接后再自动换出这些配置文件。应用程序也可以显式地卸载缓存实例。

已经加载到内存缓存表中的数据可通过 JDBC、ODBC、TTClasses 和 OCI 进行 SQL、PL/SQL、Pro*C 处理。

4.3 在缓存网格范围内共享数据

缓存组可以是本地的，也可以是全局的。对于*本地缓存组*，缓存数据不能在同一个缓存网格的成员之间共享。网格成员可能具有分散的数据或者重叠的数据，由应用程序决定数据在网格成员中的分布方式。例如，为了获得最佳性能，可将只读目录数据缓存在所有网格成员中，而将可更新的客户信息按地理区域分区缓存于不同网格成员中。提交的对缓存表的更新被传播到与其他网格成员没有协调关系的 Oracle 表中。可将本地缓存组定义为显式加载的缓存组或动态加载的缓存组。默认情况下，缓存组都是本地的，除非将它们定义为全局的。

[2] 动态加载缓存实例可用于在缓存实例的任何记录的主键或外部键上带有等式的 SQL 语句。

在*全局缓存组* 中，同一个缓存网格的成员可共享缓存数据。跨整个网格执行并发控制，网格中任意位置运行的事务始终会看到最新提交的缓存实例版本。不同网格成员提交的对同一个缓存实例的更新，按照它们在网格中的提交顺序传播给 Oracle 数据库，以确保数据一致性。

4.4 维护数据一致性

Oracle IMDB Cache 支持对缓存数据的更新，并能自动维护缓存与 Oracle 数据库之间的一致性。

缓存数据可在 IMDB Cache 中更新，也可在 Oracle 数据库中更新。IMDB Cache 能够自动将更新从缓存传播到 Oracle 数据库，也可将 Oracle 数据库的更新传播到缓存。但基本前提是，缓存组主要或仅在缓存或 Oracle 数据库中更新。一个主要设计缺陷是，希望缓存的一组表既可在缓存中大量更新，又可在后端数据库中大量更新。但是，实际应用中存在着缓存和后端数据库中更新都较少的情况。例如，Oracle 数据库中的更新只发生在夜间，其目的是为了维护，而缓存中的更新发生在白天；或者中央数据的更新发生在 Oracle 数据库中，而区域数据更新发生在缓存中。

缓存组可以是*系统管理* 的，也可以是*用户管理* 的。有三种系统管理的缓存组：

- *只读缓存组*。这些缓存组不能在缓存中更新。它们只能在 Oracle 数据库中更新，并且由 IMDB Cache 管理从 Oracle 数据库向缓存传播更新。

- *异步直写 (AWT) 缓存组*。这些缓存组可以在缓存中更新，但不能在 Oracle 数据库中更新。事务提交后，IMDB Cache 异步地将更新从缓存传播到 Oracle 数据库。

- *同步直写 (AWT) 缓存组*。这些缓存组可以在缓存中更新，但不能在 Oracle 数据库中更新。提交事务的同时将内存缓存表中的更新同步传播到 Oracle 数据库。

系统管理的缓存组有定义清晰的语义和执行这些语义的约束条件。相反，用户管理的缓存组的语义都由应用程序负责。例如，用户管理的缓存组既可在缓存中更新，也可在 Oracle 数据库中更新。

只读、AWT、SWT 和用户管理的缓存组都可以是本地缓存组。但是，只有动态 AWT 缓存组可以指定为全局缓存组。

下表汇总了可用的各种缓存组加载、缓存网格共享和一致性维护选项。

维护数据一致	向缓存组中加载数据			
	显式加载		动态加载	
只读缓存组	x		x	
AWT 缓存组	x		x	x
SWT 缓存组	x		x	
用户管理的缓存组	x		x	
	本地缓存组	全局缓存组	本地缓存组	全局缓存组
	在缓存网格范围内共享数据			

通过一个到 IMDB Cache 数据库的连接，IMDB Cache 应用程序可以向缓存组或 Oracle 数据库发送 SQL 语句。此单一连接功能由*直传 (PassThrough)* 特性启用，该特性检查内存缓存表能否在本地处理 SQL 语句，或者是否必须将其重定向到 Oracle 数据库。该直传特性提供了一些设置，使用这些设置可以指定在哪些情况下将哪些类型的语句进行传递。一个尤为有用的设置是，指定将所有更新数据库的语句都传递给 Oracle 数据库。通过该设置，应用程序可以通过一个连接在 Oracle 数据库中执行更新，在 IMDB Cache 中执行读取。

下面几节将介绍用于维护缓存数据一致性的 IMDB Cache 操作。其中部分操作由 IMDB Cache 自动启动；其他操作则由应用程序显式启动。

4.4.1 从 IMDB Cache 向 Oracle 数据库传播更新以及在全局缓存组的所有缓存网格成员之间传播更新

正如我们已经看到的，全局缓存组也是动态 AWT 缓存组。使用全局缓存组的应用程序将连接到一个网格成员。大部分时间，应用程序将访问该网格成员中已经缓存的缓存实例。但是，当应用程序试图访问该网格成员中没有的缓存实例时，IMDB Cache 将从另一个网格成员或从 Oracle 数据库动态加载要访问的缓存实例，具体从何处加载取决于该缓存实例的最新更新版本所在位置。这一过程是完全自动的，无需应用程序干预。IMDB Cache 确定最新副本所在位置，并使用点对点通信与该网格中的其他 IMDB Cache 数据库交换信息。

如果某个事务更新了任一网格成员中的缓存实例，可通过以下机制保持 Oracle 数据库与缓存的同步：

- *传播*。事务提交后，IMDB Cache 将更新传播到 Oracle 数据库。如果不久之后另一事务更新了另一网格成员中的同一个缓存实例并进行了提交，则 IMDB Cache 会确保将该提交按正确的顺序传播到 Oracle 数据库。

图 5 显示了由三个网格成员组成的缓存网格。所有网格成员缓存的是同一个全局缓存组，每个网格成员的缓存中只缓存了该全局缓存组的部分缓存实例。因为这些实例最近被访问过，因此被缓存在它们各自的网格成员中。随着时间的流逝，每个实例要么继续在其网格成员中被访问从而保留在那里；要么在另一个网格成员中被访问，从而将它移动到被访问的成员中；要么永不再被访问，这时就将它从该缓存网格中彻底换出。

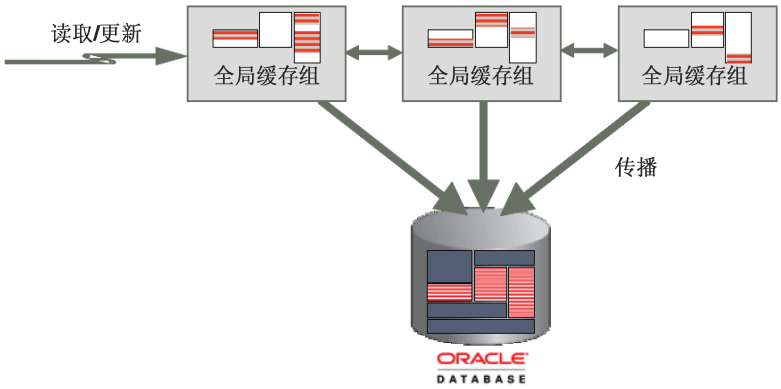

图 5. 全局缓存组的更新传播与缓存一致性

4.4.2 本地缓存组从 IMDB Cache 向 Oracle 数据库的更新传播

对于可在缓存中更新的本地缓存组，使用以下机制保持 Oracle 数据库与缓存的同步：

- *传播*。打开传播选项后，对缓存组的所有修改（即所有插入、更新和删除操作）都自动传播给 Oracle 数据库。SWT 缓存组和 AWT 缓存组传播发生的时间不同。对于 SWT 缓存组，应用程序完成了一个修改一个或多个缓存组的事务后，首先在 Oracle 数据库中提交该事务，然后在 IMDB Cache 中提交该事务。这种方法使得 Oracle 数据库在将任何所需的相关逻辑在 IMDB Cache 中提交之前就先应用于数据。对于 AWT 缓存组，应用程序完成一个事务后，就在 IMDB Cache 中提交该事务并且控制权返回给应用程序。然后，再将该事务所进行的变更异步传播到 Oracle 数据库。

- *数据库刷新*。该操作由来自应用程序的显式请求驱动，可应用于缓存组或缓存实例。只有对关闭了传播选项的缓存组或缓存实例才允许使用该机制。该操作会根据缓存中记录的值更新 Oracle 数据库中的记录。当某段时间内需要对同一个记录集进行频繁更新时，该操作很有用。不是每个更新都实时传播，而是将每个记录的最终映像发送并应用到 Oracle 数据库。

一个应用程序可以配置一个包含若干位于不同网格成员中的可更新本地缓存组的缓存网格。从网格成员向 Oracle 数据库的更新传播将由 IMDB Cache 管理，但是，我们建议不同网格成员中的本地缓存组不要重叠，这样可避免在不同节点中同时发生对同一个数据的不同更新，如果发生这种情况，将在后端数据库上产生不可预测的数据值。

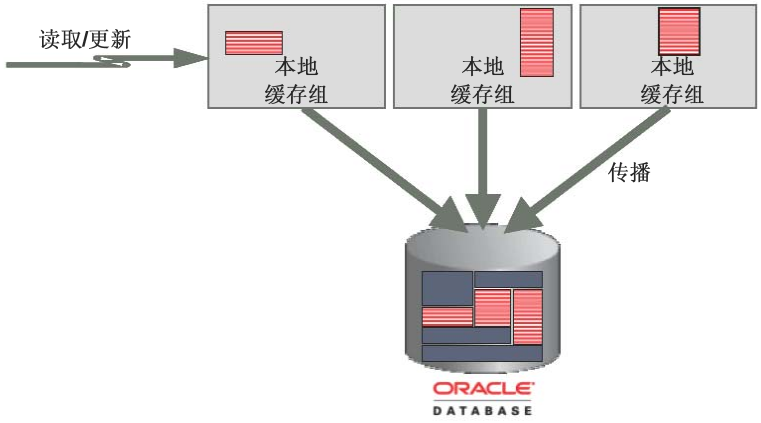

图 6. 可更新的本地缓存组的更新传播

4.4.3 本地缓存组从 Oracle 数据库向 IMDB Cache 的更新传播

对于在 Oracle 数据库中更新的本地缓存组 [3]，可以使用以下机制保持缓存内容与 Oracle 数据库的同步：

- *刷新。*这是一个来自应用程序的显式请求，可以刷新整个缓存组，也可刷新特定的缓存实例。它相当于紧跟在加载操作后面的卸载操作。

- *完全自动刷新。*使用完全自动刷新时，应用程序指定刷新应该发生的频率，而 IMDB Cache 按照应用程序指定的时间间隔自动刷新缓存组。

- *增量自动刷新。*与完全自动刷新不同，增量自动刷新只更新上次刷新后 Oracle 数据库中有所修改的记录。与完全自动刷新一样，应用程序也必须指定刷新频率，IMDB Cache 按照这个频率自动执行增量刷新。

  增量自动刷新可以与基于时间的老化一起使用，以便在缓存中保留一个滚动窗口。例如，客户支持应用程序可能希望在缓存中保留过去 5 天内报告的所有意外事故。这种情况下，应用程序可以指定缓存组应使用增量自动刷新，并且指定基于时间的老化的生存期为 5 天。随着新的意外事故不断插入 Oracle 数据库中，增量自动刷新自动将它们传播到内存缓存表。如果这些意外事故在 Oracle 数据库中有所更新，这些更新也将自动传播到内存缓存表。这些意外事故必须包含由应用程序维护的时间戳。一旦时间戳的值比当前时间早 5 天以上，则其关联的意外事故将会自动从缓存中换出。

上面介绍的三种方法在不同情况下都很有用。假定一个缓存组只需在每天凌晨 2 点刷新一次，因为此时内容提供商网站的活动最少。这种情况下，使用完全自动刷新也许是最佳选择。另一种情况，如果需要每五分钟刷新一次缓存组，则应使用增量自动刷新。最后，对于只需很少刷新但又无法预测刷新次数（次数只有应用程序知道）的缓存组，应选择使用上边介绍的第一种方法 — 刷新。

[3] 注意，全局缓存组不能在 Oracle 数据库上进行更新。

一个应用程序可以配置一个包含若干位于不同网格成员中的只读本地缓存组的缓存网格。不同网格成员中的缓存组也许是完全分散的、部分重叠的或完全相同的。将由 IMDB Cache 管理从 Oracle 数据库向所有网格成员的更新传播。

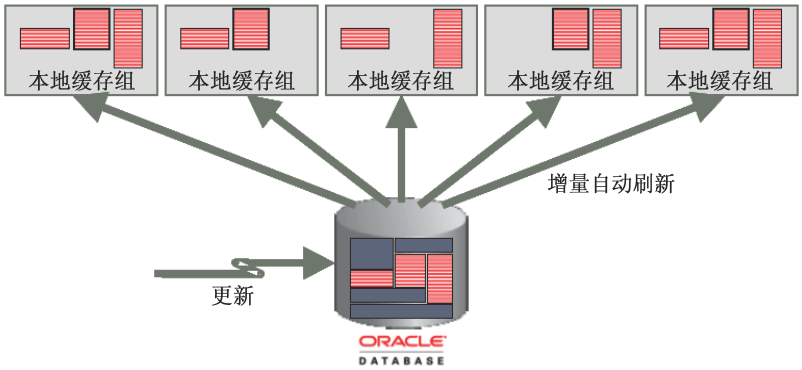

图 7. 增量刷新只读本地缓存组

4.5 高可用性

IMDB Cache 支持跨应用程序层和数据库服务器层的高可用性。

将 Oracle TimesTen 专门用作记录数据库时，与用作 Oracle 数据库的内存数据库缓存相比，它通过复制、各种联机操作功能以及大量支持故障切换、恢复和联机升级的实用程序，更能确保数据的高可用性。通过与 Oracle Clusterware 的集成，提供了数据库和应用程序的自动故障检测和自动故障切换功能。类似地，Oracle 数据库通过包括 Oracle Real Application Clusters (RAC)、Oracle 自动存储管理 (ASM) 和 Oracle Data Guard 在内的一组特性，支持其数据的高可用性。另外，IMDB Cache 的 Replication 组件提供了大量特性，不但可确保缓存数据的高可用性，还可确保分布于应用程序层和数据库层中的 Oracle 数据库中的故障的自动恢复。下面分别介绍这些特性。

4.5.1 内存中缓存节点故障的处理

为防止缓存节点故障并确保缓存数据的持续可用性，TimesTen 复制特性提供了缓存节点的故障切换与恢复处理。具有多个只读使用者的主动/备用对复制配置旨在 将 Oracle 数据库作为配置的一部分，以便提供缓存节点的故障切换与恢复。

使用主动/备用对时，始终将所有更新先应用于主动节点。然后将更新复制到备用节点，接着再将更新从备用节点复制到所有只读使用者节点。这样，备用节点始终在只读使用者节点前头，如果主动节点发生故障，毫无疑问可以确定哪个使用者节点应该成为新的主动节点。

只读缓存组

主动/备用对复制配置旨在让只读缓存组与直写缓存组协同工作。使用只读缓存组时，Oracle 数据库中应用的更新只传播到主动节点。然后，TimesTen 复制将更新先通过备用节点再传播到所有其他节点。

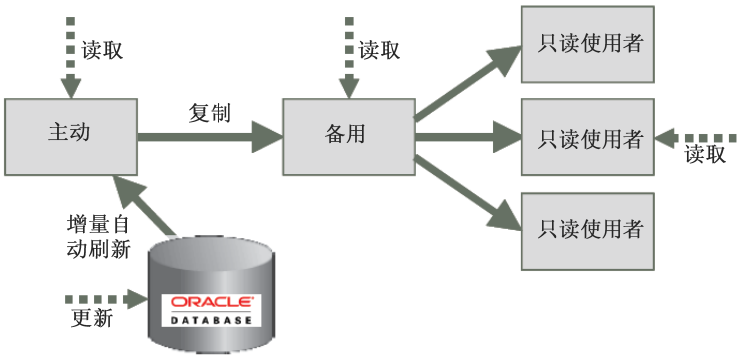

图 8. 使用主动/备用对配置的只读缓存组

如果主动节点发生故障，备用节点就成为新的主动节点。从这时开始，来自 Oracle 数据库的更新将传播到新的主动节点（以前的备用节点），然后再从该节点复制到只读使用者节点。一旦原来的主动节点恢复联机，它将成为新的备用节点。TimesTen 复制特性自动处理从主动到备用的更新传播的切换以及故障节点的恢复。

类似地，如果备用节点发生故障，则复制操作从主动节点重定向到只读使用者节点。一旦备用节点恢复联机，复制特性将确保它补上恢复为备用节点角色前错过的所有更新。

直写缓存组

使用直写缓存组时，更新先应用于主动节点。然后再将它们复制到备用节点。一旦到达备用节点，就可将它们传播到 Oracle 数据库并复制到所有只读使用者节点。

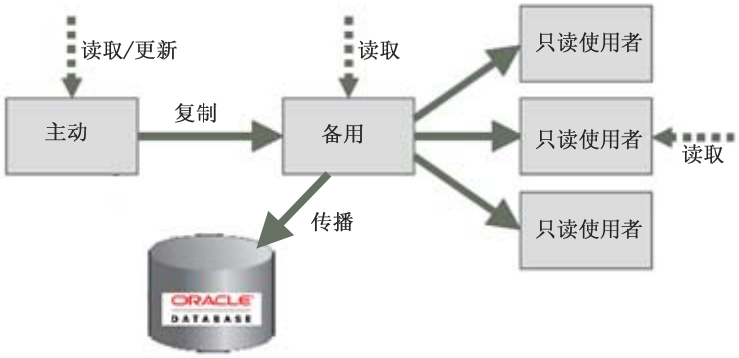

图 9. 使用主动/备用配置的直写缓存组

如果主动节点发生故障，备用节点就成为新的主动节点。从这时开始，必须将所有更新发送到新的主动节点，即，以前的备用节点。对新主动节点的更新将传播到 Oracle 数据库并复制到只读使用者节点。一旦原来的主动节点恢复联机，它将成为新的备用节点。TimesTen 复制特性自动处理新备用节点的恢复，并负责将向后端传播更新的职责转移给新备用节点。

类似地，如果备用节点发生故障，来自主动节点的复制将自动重定向到只读使用者节点，而主动节点将直接开始向 Oracle 数据库传播更新。一旦备用节点恢复联机，复制特性将确保它补上恢复为备用节点角色前错过的所有更新，并让备用节点重新开始向 Oracle 数据库和只读使用者传播更新。

4.5.2 Oracle 数据库中的故障处理

如果由于网络故障、硬件故障或 Oracle 数据库故障等原因导致 IMDB Cache 无法访问 Oracle 数据库，IMDB Cache 的设计已经将这些故障对它的影响降至最低。应用程序仍然可以继续访问内存缓存。而且，在使用 AWT 缓存组的情况下，对缓存的更新将继续记录到 Oracle TimesTen 中，以便在 Oracle 数据库恢复为可访问后，将更新传播给它。类似地，对 Oracle 数据库中的只读缓存组进行的但尚未传播到内存缓存的变更仍将记录在 Oracle 数据库上，一旦 Oracle 数据库恢复为可访问，就会将这些变更传播给缓存。

另外，IMDB Cache 充分利用了 RAC 的高可用性特性。RAC 配置包含一个可供几个节点访问的物理数据库。一个节点上的运行时配置称为一个实例。RAC 提供了跨所有实例的负载平衡、高可用性和数据一致性。

无需用户干预，IMDB Cache 即可从 RAC 节点故障中快速恢复。为此，在可以使用 Oracle 透明应用程序故障切换 (TAF) 特性和快速应用程序通知 (FAN) 特性的场合，IMDB Cache 会使用这两个特性。也就是说，这两个特性是否可用取决于 Oracle 客户端、服务器和 TAF 配置的版本。如果与 IMDB Cache 连接的 Oracle 实例发生故障，则自动将该连接切换到另一个实例。发生故障时，如果正在进行刷新、完全自动刷新或增量自动刷新操作，则该操作将自动回滚内存数据库中发生的变更，并重启该操作。发生故障时，如果正在进行 AWT 缓存组的传播操作，那么，如果需要对发生在 Oracle 数据库中的变更进行回滚，则该事务将自动回滚这些变更，并将重启传播操作。

如果使用同步 Data Guard 将 Oracle 数据库复制到备用数据库，主动 Oracle 数据库万一出现故障，则 IMDB Cache 将自动切换到备用 Oracle 数据库，并且不会丢失数据。

5. 性能

为了测量 IMDB Cache 的性能，我们开发了一个基准测试，它可以模拟蜂窝网络中使用的归属位置寄存器 (HLR) 应用程序。该基准测试由 7 个事务组成，每个事务模仿 HLR 执行的一个典型操作，如建立或删除呼叫转移，或更新移动电话用户的有关信息。

我们用两种不同的配置运行了这个基准测试。在第一种配置中，HLR 基准测试应用程序基于 Oracle 数据库运行，基准测试应用程序运行在一台服务器上，而 Oracle Database 10g 运行在另一台服务器上。在第二种配置中，我们在 Oracle 数据库的前面添加了 IMDB Cache，HLR 基准测试应用程序直接与 TimesTen 缓存数据库链接，运行在一台服务器上，而 Oracle Database 10g 运行在另一台服务器上。缓存数据存储在 AWT 缓存组中，这使得对缓存数据的所有更新可自动传播到 Oracle 数据库。

基准测试应用程序是用 Java 实现的，使用 JDBC 进行数据访问。这四台服务器具有相同的配置：6GB 物理 RAM，两个具有超线程的 Intel Xeon 2.4GHz 处理器，运行 Oracle Enterprise Linux 5.2。

我们分别测量了基于 Oracle 数据库运行和基于 IMDB Cache 运行时的每种事务的平均响应时间。下图显示了使用 IMDB Cache 时，应用程序的响应时间大大减少。

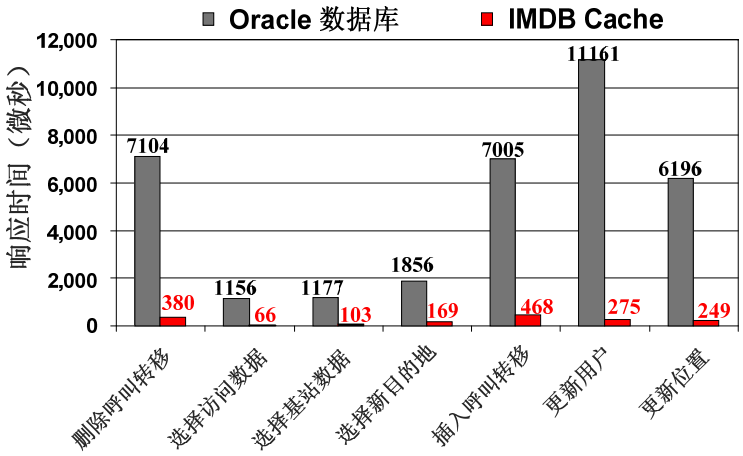

图 10. HLR 基准测试应用程序的响应时间比较

我们也测量了这两种配置中所有事务的综合吞吐量。下图显示了使用 IMDB Cache 时，应用程序的吞吐量大大增加。

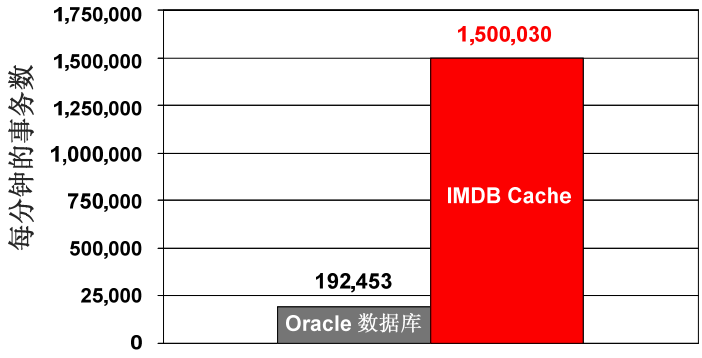

图 11. HLR 基准测试应用程序的吞吐量比较

该基准测试显示了使用 IMDB Cache 的好处。如上面两个图所示，应用程序的响应时间减少至原来的 1/40 到 1/10，总吞吐量提高了 7 倍多。通常，IMDB Cache 改善率随不同的硬件和平台会有所差异。

6. 示例

在本节中，我们将介绍几种缓存的使用情况以及针对这些情况建议的 IMDB Cache 配置和缓存组类型。注意，尽管每个示例使用的是特定类型的缓存组，但同一个 IMDB Cache 中可以同时存在不同类型的缓存组以更好地满足应用程序的需要。

6.1 只读缓存

使用增量自动刷新的只读缓存组最适用于缓存频繁引用的数据。

许多应用程序可通过使用只读缓存受益。这些应用程序的主要特征是需要反复查询某些记录。无论这些记录是否频繁更新，读/写比率都非常高。这类记录的示例包括：在线购物应用程序的价格表、航班预定应用程序的航班时刻表，以及酒店预订应用程序的可用房间表。

适用于此类数据的最佳缓存配置是使用增量自动刷新的（系统管理的）只读本地缓存组。在后端数据库上对数据进行更新。更新将自动传播到缓存。传播的频率由应用程序决定，应根据后端数据库的更新频率及应用程序需要的数据流量决定传播频率。

如果缓存部署于缓存网格的多个网格成员中，则应在这些网格成员上定义本地缓存组，每个成员将直接从后端数据库获得更新。

注意，在线购物应用程序通常无需频繁更新缓存的价格表，因为价格表并不经常变化。但是，航班跟踪应用程序在频繁读取数据的同时还要保证缓存的航班状态及时更新。这种更新最适合使用定义了较小的合理增量自动刷新间隔（如 5 分钟）的 Oracle 数据库，为只读缓存组也定义同样的刷新间隔。IMDB Cache 将所有更新自动传播到包括更新数据的网格成员。连接到某个网格成员的应用程序可以建立一个到 IMDB Cache 的连接，并且在缓存中执行所有读取操作的同时，使用 PassThrough 选项将所有更新路由到 Oracle 数据库。

6.2 只读滚动窗口缓存

使用增量自动刷新和基于时间老化的只读缓存组最适用于缓存落在某个滚动窗口中的频繁引用的数据。

许多情况下，应用程序需要的只读数据是与时间因素有关的数据，这种情况下，新数据的访问比旧数据频繁。对这类应用程序，新数据不断生成，旧数据逐渐变得没有价值。因此，我们可以考虑一个固定长度的时间段，该时间段不断向前推进，使数据从一端进入该时间段，从另一端离开该时间段。应用程序只对该时间段内的数据感兴趣，通常将这种机制称作滚动窗口。

这类需要落在滚动窗口内的数据的应用程序示例包括：股票交易应用程序，它可能需要最近 3 天的交易数据；新闻传播应用程序，它可能需要最近 24 小时的新闻简报。

为了缓存落在滚动窗口中的数据，我们希望自动将新数据放入缓存中，并且还能自动将旧数据换出缓存。我们还希望对后端数据库进行的修改能自动更新到缓存的数据中。适用于此类数据的最佳缓存配置是使用增量自动刷新和基于时间老化的（系统管理的）只读缓存组。

与前一个示例一样，如果缓存部署于缓存网格的多个网格成员中，则应将在这些网格成员上定义本地缓存组，每个成员将直接从 Oracle 数据库获得更新。

6.3 可更新的缓存

异步直写缓存组最适用于可更新的缓存。

有些应用程序需要对缓存数据立即进行实时更新，并最终向 Oracle 数据库传播更新。例如，管理和供应电话预订服务及验证对该服务访问的应用程序通常将预订者信息缓存在 IMDB Cache 中。对用户服务的变更必须立即反映在缓存中，并且应该传播到后端数据库。

适用于这类数据的最佳配置是异步直写缓存组。

如果预订者人数过多，需要将应用程序部署在缓存网格的多个网格成员中，则应在这些网格成员上定义将本地缓存组，每个成员的缓存组拥有自己的预订者子集，并且不同网格成员拥有的子集之间不重叠。例如，预订者有可能是按区号分区的。

6.4 可更新的动态缓存

使用动态加载和基于使用情况老化的异步直写全局缓存组最适用于可更新的动态缓存。

对某些应用程序来说，对活动数据的访问必须非常快，但活动数据集随时间变化并且是一个更大量数据集的子集，而这个大量的数据集因为数据量太大，无法完整地容纳在一个缓存中。这就需要按需将活动数据引入缓存，并且缓存的内容需要是动态的，以便活动数据可以替换陈旧数据。

这类应用程序的一个示例就是呼叫中心应用程序，它管理着大量并发客户会话。通常，这种应用程序部署在几个应用服务器节点上。与呼叫中心联系的客户被自动路由到一个可用的应用服务器节点，理想情况下，该服务器节点也应提供该客户的配置文件。

适用于该情况的最佳配置是使用动态加载和基于使用情况老化的 AWT 全局缓存组，每个应用服务器节点上配置一个网格成员。

使用这种配置，当客户被路由到可用的服务器节点时，会将该客户的配置文件从 Oracle 数据库动态加载到该服务器节点上的网格成员中。客户结束一次呼叫后，对客户配置文件的更改将从 IMDB Cache 传播到 Oracle 数据库。LRU 老化将自动从 IMDB Cache 中删除非活动客户的配置文件。如果同一个客户在首次呼叫后不久又联系了呼叫中心，但这次被路由到另一个服务器节点，则该客户的配置文件会动态加载到这个新节点，这次既可以从 Oracle 数据库加载，也可以从之前加载过的网格成员加载，具体取决于配置文件最新副本所处的位置。由 IMDB Cache 决定最新副本所处的位置。它还负责管理对网格内数据的并发更新。

所有客户数据都存储在 Oracle 数据库中。Oracle 数据库远远大于组合的 IMDB Cache 数据库，最适合那些不要求 IMDB Cache 实时性能但需要访问大量数据的应用程序访问。这类应用程序可能包括计费应用程序和数据挖掘应用程序。

随着客户群的增加以及为更多客户提供服务的要求的同步增加，呼叫中心可能会决定部署更多的应用服务器节点。可以向 IMDB Cache 网格添加新 IMDB Cache 成员，但不会中断该网格中正在运行的请求。类似地，某些节点出现故障或被删除也不会中断该网格其他节点的操作。

6.5 发生率不均的数据捕获缓存

基于使用情况老化的异步直写缓存组最适用于捕获发生率不均的数据。

有一类应用程序，某些时间段的新数据生成率特别高，而其他时间段则一般。在高活动时间段，后端数据库常常无法满足应用程序要求的高吞吐量。这类应用程序可以通过使用缓存受益，实际上，通过使用缓存最终可以使新生成数据的发生率达到“平滑”。

例如，股票行情应用程序的新价格的发生率随时间变化非常大。开市与闭市时数据发生率特别高，其他时间则较低。通常，基于磁盘的数据库通常无法处理峰值发生率，但 IMDB Cache 可以承担这一任务。

适用于此类数据的最佳缓存配置是使用基于使用情况老化的（系统管理的）异步缓存组。插入到直写缓存组的数据自动传播到后端 Oracle 数据库。基于使用情况的老化从内存缓存中自动删除老化的数据以释放空间。

6.6 稳定的高发生率的数据捕获缓存

基于使用情况老化的异步直写缓存组与基于使用情况老化的 TimesTen 表一起使用，这种配置最适用于捕获具有稳定的高发生率的数据。

另一类应用程序，其新数据的生成率也很高，并且高发生率不一定会减退。如果因为没有为后端数据库留出追赶的时间而使发生率无法平滑，那么只是将那些发生率太高而无法被后端数据库容纳的数据进行临时缓存是不能解决问题的。但是，对这类应用程序，在将新生成的数据永久存储在后端数据库中之前，通常将它们集中到一个更精简的表单中。通常这些应用程序会实时分析收集的数据，以检测感兴趣的模式或异常模式。

此类应用程序的一个示例就是从传感器或 RFID 阅读器收集数据的应用程序。数据经常重复且非常容易收集，但常常需要进行实时分析。

适用于此类应用程序的最佳配置就是，当数据到达仅由 Oracle TimesTen 管理的一个或多个表时插入数据，也就是说，后端数据库中没有该数据的映像。非缓存的仅 TimesTen 表配置了基于使用情况的老化。数据一旦被应用程序收集，就会插入到基于使用情况老化的（系统管理的）异步直写缓存组的缓存中。IMDB Cache 自动将收集到的所有数据传播到后端数据库。因为仅 TimesTen 的表和缓存表都配置了基于使用情况的老化，因此最近最少使用的记录将自动换出，以便使内存中的空间用于存储新记录。

6.7 可更新的用户管理的缓存

显式刷新的用户管理的缓存组最适用于数据频繁更新但业务事务不多的应用程序。

某些应用程序为了达到最佳性能需要在缓存中执行多次更新，但只需在 Oracle 数据库中永久记录最终的事务。此类应用程序的一个示例就是电子商务应用程序，这种应用程序可能需要为活动用户维护大量的购物车。并且缓存中的购物车会反复更新。因为这些更新意义不大，因此无需传播到 Oracle 数据库。然而，一旦用户下了定单，就需要将这笔交易永久记录在 Oracle 数据库中。

适用于此类数据的最佳配置是用户管理的可更新缓存组，在这种配置中，每当应用程序需要向 Oracle 数据库中记录一个事务时，它就会发出显式刷新请求。可以与基于使用情况的老化一起使用这种配置，以便自动将丢弃的购物车从缓存中删除。

6.8 只读动态分布缓存

动态加载和基于使用情况老化的只读表最适用于只读动态分布缓存。

某些情况下，一个应用程序可能分布在许多节点上，以便处理单个节点无法处理的吞吐率，并且应用程序需要的活动数据集是动态的，在任何指定的时间，活动数据集都是一个比完整数据集小得多的子集。此类应用程序的一个示例就是贸易应用程序，在这种程序中活动数据就是活动贸易商的配置文件。

适用于此类数据的最佳配置就是在每个节点上配置内存缓存，也就是在 Oracle 数据库的同一组表上配置只读缓存组，并且对这些缓存表使用动态加载和基于使用情况的老化。接下来将发生的是，当每个节点需要贸易商的配置文件时，该节点就会得到需要的配置文件，这些文件是自动加载的，当不再需要这些配置文件时，它们将从缓存中换出，将空间腾给需要的配置文件。

7. 总结

通过将 Oracle 数据库中对性能影响极大的部分表和表片段从 Oracle 数据库缓存到应用程序层，Oracle In-Memory Database Cache 大大降低了应用程序事务的响应时间。与简单的结果缓存机制不同，在这种缓存机制中，应用程序可以对缓存的数据执行 SQL 和 PL/SQL 命令，因为缓存表的管理与 TimesTen 内存数据库中的常规关系数据库表的管理是一样的。不同的应用程序可以共享缓存。更新可应用于缓存，从而使缓存与 Oracle 数据库保持一致。使用 IMDB Cache 缓存数据比使用其他缓存技术有许多优势，因为它引进了完整的关系功能，提供了增量可伸缩性、位置透明性和稳定的性能，还能自动维护与 Oracle 数据库的一致性，同时对运行于应用程序层的应用程序提供了跨层的高可用性。

通过让数据更接近应用程序并且在内存数据库中处理查询，Oracle In-Memory Database Cache 大大降低了响应时间。通过从 Oracle 数据库服务器上分流一些数据处理工作，在不影响后端数据库集中管理的情况下明显提高了应用程序的整体吞吐量。

8. 参考资料

1. 使用 Oracle TimesTen In-Memory Database 实现极限性能。*Oracle 白皮书（2009 年 7 月）。*

甲骨文（中国）软件系统有限公司

北京总部

地址：北京市朝阳区建国门外大街1号，国贸大厦2座2208室
邮编：100004
电话：(86.10) 6535-6688
传真：(86.10) 6505-7505

北京上地6号办公室

地址：北京市海淀区上地信息产业基地，上地西路8号，上地六号大厦D座702室
邮编：100085
电话：(86.10) 8278-7300
传真：(86.10) 8278-7373

上海分公司

地址：上海市卢湾区湖滨路222号，企业天地商业中心1号楼16层
邮编：200021
电话：(86.21) 2302-3000
传真：(86.21) 6340-6055

广州分公司

地址：广州市天河北路233号，中信广场53楼5301&5308室
邮编：510613
电话：(86.20) 8513-2000
传真：(86.20) 3877-1026

成都分公司

地址：成都市人民南路二段18号，四川川信大厦20层A&D座
邮编：610016
电话：(86.28) 8619-7200
传真：(86.28) 8619-9573

大连分公司

地址：大连软件园东路23号，大连软件园国际信息服务中心2号楼五层502号A区
邮编：116023
电话：(86.411) 8465-6000
传真：(86.411) 8465-6499

济南分公司

地址：济南市泺源大街150号，中信广场11层1113单元
邮编：250011
电话：(86.531) 8518-1122
传真：(86.531) 8518-1133

甲骨文软件研究开发中心（北京）有限公司

地址：北京市海淀区中关村软件园孵化器2号楼A座一层
邮编：100094
电话：(86.10) 8278-6000
传真：(86.10) 8282-6455

甲骨文研究开发中心（深圳）有限公司

地址：深圳市南山区高新南一道飞亚达大厦16层
邮编：518057
电话：(86.755) 8396-5000
传真：(86.755) 8601-3837

沈阳分公司

地址：沈阳市沈河区青年大街219号，华新国际大厦17层D单元
邮编：110016
电话：(86.24) 2396 1175
传真：(86.24) 2396 1033

南京分公司

地址：南京市玄武区洪武北路55号，置地广场19层1911室
邮编：210028
电话：(86.25) 8476-5228
传真：(86.25) 8476-5226

杭州分公司

地址：杭州市西湖区杭大路15号，嘉华国际商务中心702室
邮编：310007
电话：(86.571) 8717-5300
传真：(86.571) 8717-5299

西安分公司

地址：西安市高新区科技二路72号，零壹广场主楼1401室
邮编：710075
电话：(86.29) 8833-9800
传真：(86.29) 8833-9829

福州分公司

地址：福州市五四路158号，环球广场1601室
邮编：350003
电话：(86.591) 8801-0338
传真：(86.591) 8801-0330

重庆分公司

地址：重庆市渝中区邹容路68号,大都会商厦1611室
邮编：400010
电话：(86.23) 6370-8898
传真：(86.23) 6370-8700

深圳分公司

地址：深圳市南山区高新南一道飞亚达大厦16层
邮编：518057
电话：(86.755) 8396-5000
传真：(86.755) 8601-3837

甲骨文亚洲研发中心（上海）

地址：上海市杨浦区淞沪路290号，创智天地10号楼512-516单元
邮编：200433
电话：86-21-6095 2500
传真：86-21-6095 2555

公司网址：http://www.oracle.com（英文）
中文网址：http://www.oracle.com/cn（简体中文）
销售中心：800-810-0161
售后服务热线：800-810-0366
培训服务热线：800-810-9931

欢迎访问：
http://www.oracle.com（英文）
http://www.oracle.com/cn（简体中文）