OS 虚存技术与双缓冲结合——Sambase 对象存储系统*

高 媛 庄 严 朱 坎 郑振楣 石树刚

(武汉大学计算机科学系 武汉 430072)

摘要 利用操作系统的虚存技术,在虚存中开辟一块空间(大至整个虚存)作为 DBMS 的缓冲池.这样实现了 DBMS 和 C++之间的无缝连接,无需特殊的编译程序,应用程序利用普通的虚存指针,使用与操作临时对象相同的程序代码来直接快速地操作持久对象,建立在这种虚存技术上的缓冲机制使数据库的大小突破了虚存大小的限制.

关键词 指针搅拌,双缓冲,虚存对象管理.

中图法分类号 TP311.13

针对 GIS 的数据量非常庞大(经常以 GB 计算)和 GIS 中图形显示需要直接操作大量对象这 2 个突出问题,我们设计了虚存技术和双缓冲机制相结合的对象存储系统.

Sam 是武汉大学数据库组设计的用于开发 GIS 的支持系统.它的核心 Sambase 是一个面向对象的 DBMS,在它上面我们开发了各种用户接口:数据库语言接口 Sam. OSQL,程序编程接口 Sam. ProC++,图形接口 Sam. View.[1]

在老版本的 Sambase 中,除 OSQL 外,用户不能直接访问持久对象,必须做从 Sambase 的对象存储格式到应用对象格式的转换.这个问题在 GIS 的图形接口中表现得更为突出.GIS 必须提供方便显示并操作空间数据的图形接口.一块图形的显示涉及到大量的空间对象的操作,因此要求对这些持久对象进行直接快速访问.以前提供图形显示工具与底层数据库管理系统之间数据转换的接口,效率不高、灵活性差,每增加一个空间对象类,就必须增加一套对这个类的转换代码,限制了系统的扩充性.

为解决这类应用与 Sambase 中对象格式的不匹配问题,我们在 Sambase 的存储系统中增加了一个虚存对象管理器,使问题从图 1 的(a)简化到(b).

虚存对象管理器调用 OS 的虚拟存储器的功能,使应用程序可以用普通的虚存地址指针直接对数据库缓冲池中的对象进行访问,应用程序可以用访问临时对象一样的代码对持久对象进行操作.运用这种方法,对持久对象的访问就无需特别的编译程序支持,只需将

---

* 作者高媛,女,1972 年生,硕士,主要研究领域为面向对象数据库.庄严,1971 年生,硕士,主要研究领域为面向对象数据库,地理信息系统.朱坎,1970 年生,硕士,主要研究领域为面向对象数据库.郑振楣,女,1937 年生,教授,主要研究领域为分布数据库,面向对象数据库.石树刚,1936 年生,教授,主要研究领域为面向对象数据库,演绎数据库.

本文通讯联系人:郑振楣,武汉 430072,武汉大学计算机科学系

本文 1996-09-18 收到修改稿

C++的类库链入应用程序,就可使用与操作临时对象一样的代码来操作持久对象.

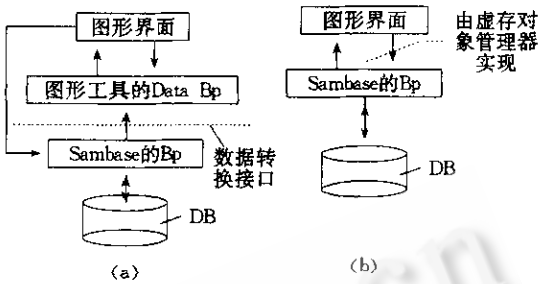

图1

Sambase首先在虚存中开辟一块空间作为缓冲池(BP),由操作系统的软、硬件实现缓冲池的虚存空间到实际的物理内存的映射.Sambase在虚存基础上采用了2种缓冲技术:在服务器上采用页面缓冲;在客户机上采用双缓冲机制,双缓冲包括页面和对象2种缓冲.我们采用的懒惰拷贝和急切重定位相结合的机制,在大多数应用中优于单纯页面缓冲机制.

本文第1节介绍对象存储系统概貌;第2节描述利用虚存技术硬件,实现对象的虚存指针到缓冲池页面映射的虚存对象管理;第3节讨论存储管理的双缓冲机制;第4节是小结.

1　对象存储系统概述

对象存储系统负责在内存缓冲池中查找和装入对象,并负责对外存页面的申请和释放以及内、外存之间的页面交换.

对象存储系统支持对象标识(Oid),本系统的Oid为形如类标识、实例标识的二元组形式.作为存储系统底层的DMOM(外存对象管理器)就是用Oid对外存中的对象进行存取.在对象实例存储结构中(如图2所示),对象之间的引用关系也是利用Oid实现的.

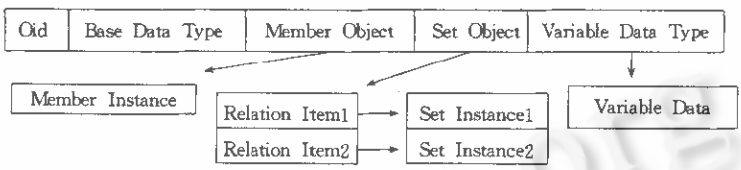

图2 Sambase中对象实例存储结构

在Member Object中存放Member Instance的Oid,表示元组引用;在Relation Item中存放Set-Instance的Oid表,表示集合引用(多值引用).

下面介绍存储系统的大概结构,如图3所示,其中(1)OM:对象管理器.以Oid为句柄,在双缓冲池中访问对象;(2)VMOM:虚存对象管理器.用对象虚存指针在页面缓冲池中访问对象;(3)DMOM:外存对象管理器.在OM,VMOM访问对象失败时,调用DMOM通过Oid读取磁盘中的对象页;(4)DBM:双缓冲管理器;(5)LLM:局部锁管理器;(6)GLM:全局锁管理器;(7)BM:缓冲池管理器.

在Client-Server结构中,Server采用页面缓冲,Client采用双缓冲,Server与Client之间以页面作为数据的传送单位.

执行过程.首先,应用接口(Application Interface)对应用程序中有关对象的操作进行解释.如果是数据库语言接口(DBL Interface),则通过OM直接用Oid在双缓冲池中操作对象.如果是程序编程接口(PL Interface),则通过VMOM直接用指向对象的虚存地址在页面缓冲池中操作对象.如果OM的Oid和VMOM中的虚存地址所对应的对象不在缓冲池

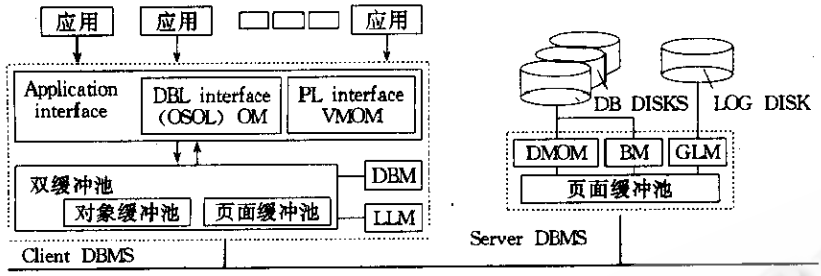

图3 Sambase的存储管理系统

内,则调用Server的DMOM从外存读取数据库对象页,并将该页送到Client.DMOM相当于为Client提供一个本地数据库.在这个存取过程中,事务处理机制(并发控制和恢复)在Server上维护全局锁表,在Client上维护局部锁表,并在Server上记录日志以便进行恢复.

2 虚存对象管理器(VMOM)

我们在应用程序设计语言编写的应用程序和DBMS之间设计了虚存对象管理器,无需增加编译程序,只要将C++的类库链入应用程序,就可以在应用程序中使用指向对象的虚存指针直接在页面缓冲池中访问对象,从而用与操作临时对象一样的代码来操作持久对象.使一次可访问的对象数仅受虚存大小的限制,解决了引言中图形显示的应用中大量访问对象的效率问题.虚存对象管理器为用户提供了一个方便、高效的数据库接口.

2.1 虚存对象管理器的主要思想

VMOM利用Unix中的虚存部分,实现用普通的虚存地址访问缓冲池中的对象,并在命中失败时把对象页面从外存调入内存缓冲池.为了访问持久对象,VMOM维护从虚存页到缓冲页的物理映射,这个物理映射是动态的、多对多关系,一个虚存页面在不同时刻可以映射到不同的缓冲页面.同时,VMOM也维护从虚存地址到对象Oid的逻辑映射,这个映射是静态的,在一个事务中,相同的虚存地址映射到相同的Oid.

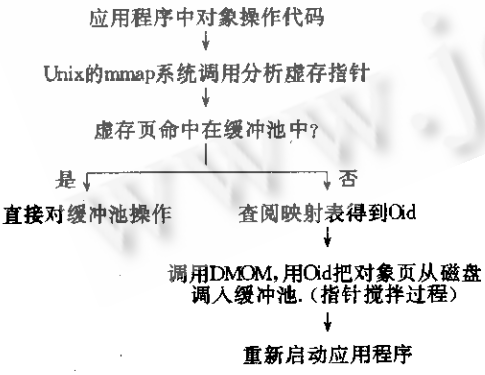

图4 VMOM的处理流程

当应用程序用虚存地址第1次访问在某页中的对象时,记录为页面失败,唤醒命中失败处理进程.命中失败处理进程将查阅虚存地址到Oid的映射表,得到虚存指针相对应的Oid,然后再调用DMOM(外存对象管理器)根据得到的Oid把对象面从磁盘调入缓冲池,并同时维护2种映射,进行指针调整,把虚存页设置成可访问权限.然后重新启动应用程序,这时的虚存指针已经映射到缓冲池中的某页,应用程序可以直接对该页中的对象访问.执行流程参见图4.

在DMOM用Oid把对象所在的面从磁盘调入缓冲池时,如果调入的页a中的对象包含有对未调入的页中的对象的引用,即页a中包含其它页中对象的Oid时,则为未分配虚存地址的引用页分配一个虚存地址.如果这次分配的虚存地址与保存在页a的meta-data中引用对象的Oid相对应的上一次分配的虚存地址不同,则维护磁盘地址Oid与虚存地址的

映射. 这个过程叫作指针搅拌过程(Pointer Swizzling).

2.2　实现技术

VMOM 利用 Unix 系统提供的系统调用 mmap,实现从虚存地址到缓冲池中地址的物理映射,并且控制虚存页的访问权限.

mmap()建立一张从某进程的内存地址空间(pa, pa+len)和外存文件 fd(句柄)从偏移值 offset 起 len 长的空间的映射. mmap()的成功调用将返回与文件地址 offset 相对应的内存地址 pa. 调用时 prot 参数可以规定此页的访问权限. 有关系统调用 mmap()的详细信息及其参数的意义可参见 Unix 的程序员手册.

由于 VMOM 利用系统调用 mmap()实现从虚存空间到实际的内存缓冲池的映射,而 mmap 把文件的偏移值作为虚存指针管理函数,使之能够和 mmap 一起完成虚存空间的管理及其到实际内存缓冲池的映射的格式,所以 VMOM 重写了 Unix 的 malloc 一类存储. VMOM 首先打开一个与缓冲池大小相同的文件,然后调用 mmap 把整个文件作为一片虚存,这样 DBMS 就可以用这个范围内的虚存去访问缓冲池. 应用程序对内存的申请和释放是调用重写后的存储管理函数进行,事实上对内存的管理转换成了对文件的管理.

这里应该特别注意支持缓冲池访问的文件服务器. mmap 决不会为文件相应的虚存空间分配交换空间和实际的物理空间,所以把一个大的虚存范围映射为缓冲池虽然会增加操作系统页面的大小,但并不增加进程大小.

2.3　内存中的页表结构

VMOM 维护一张内存页表记录从虚存地址到 Oid 的映射,在此前引起缺页中断的页面及其所引用页面的描述项都包含在页表中. 我们把磁盘中的数据页划分成 2 种类型:包含对象的大小小于页面的大小的页面叫作小对象页;对象大小大于页的大小时,对象将跨页存放,此页面叫作大对象页. 页描述如图 5 所示. 大对象页的描述管理比小对象页的描述管理复杂. 首先一个大对象包含所有页用一个描述项描述,当应用程序访问大对象的第 1 页时,这个描述项将分裂成为 2 项,一项描述访问过的那页,另一项描述剩下的那些页,以后的过程依此类推.

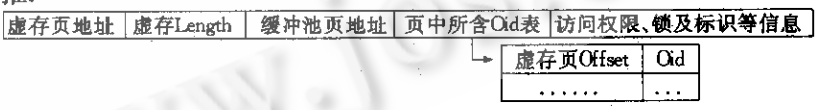

图5　页描述项的格式

这个页表根据虚存地址组织为一棵高度平衡二叉树,这样做,一方面方便了大对象描述项的分裂过程;另一方面方便保持内存地址顺序. 当数据库变得比整个虚存空间还大时,就必须查找二叉树,找一个当前不用的页从虚存中替换出去,这样,同一虚存页可以映射到多个外存页,但这些外存页中只有一页有存取权限,其它页没有. 这样就解决了存取冲突问题.

另一方面,页面也根据 Oid 组织为 hash 表,以维护从 Oid 到虚存地址的映射,在缺页中断处理进程中,利用 hash 表进行指针搅拌.

2.4　指针搅拌(Pointer Swizzling)

对应每张磁盘页面有 2 个辅助信息对象 meta-object 和 bitmap-object. meta-object 记录如下每个对象的每次引用(包括元组引用和集合引用)的 Oid 与上次分配的虚存地址间的映射. 从形式上看,meta-object 是(虚存地址,Oid)元组的数组. bitmap-object(位图对象)记

录了页中所引用的准确 Oid 的位置,以便进行指针搅拌.

在缺页中断处理进程中,检查此页在磁盘上的辅助信息对象 meta-object,查看该页中对象所引用对象的 Oid 上次所分配的虚存地址与这次所分配的虚存地址是否一致,如果不一致,则进行指针搅拌,把 Oid 映射到当前分配的虚存地址,并写回 meta-object 中.

下面介绍数据页第 1 次被调入内存的过程,首先读入数据页,再读出此页的 meta-object,在内存 hash 表中查看是否所有的引用 Oid 所指的对象都调入内存.如果某对象没有调入,则根据 meta-object 中的信息,创建一个页描述项给这个对象所在页,并根据 meta-object中记录的该对象上次的虚存地址为其分配虚存空间,如果该地址还空着,则分配给此页,否则选择新虚存页给它.如果对象均已调入内存,系统也要查看上次的虚存地址与描述项中的虚存地址是否相同.

如果所有的 Oid 都映射到上次的虚存地址,则搅拌过程结束.如果有 Oid 映射到新的虚存地址,则读出 bitmap-object,修改 meta-object 的有关信息.

3 双缓冲管理

双缓冲包括以页面为单位的页缓冲和以对象为单位的对象缓冲,双缓冲既能缓冲整页,也能缓冲单独的对象.对于页面缓冲,当数据库相对于当前进程没有聚簇时,则缓冲池的大部分空间被没有用的对象占据,利用率不高.而如果只有对象缓冲,当数据库对于当前的进程聚簇得很好时,不必对每个对象都拷贝到对象缓冲池中,增加时间开销.只有双缓冲能同时缓冲页和对象,对包含许多应用程序将访问的对象的页面进行页缓冲,而把对象从数据利用率不高的页中拷贝到对象缓冲池中.

对象缓冲池和页缓冲池各有自己的缓冲机制,而双缓冲主要讨论页缓冲与对象缓冲之间的调度机制.

双缓冲的主要动作:①从页缓冲池对象所在页中将对象拷贝到对象缓冲池中;②对象的重定位工作,在双缓冲机制下,对对象的操作既可在对象缓冲池中进行,也可在页面缓冲池中进行,但任何操作都必须保持 2 个缓冲池中数据的一致性.双缓冲中,只有页面缓冲与外存交换数据,对象缓冲只能通过页面缓冲与外存发生联系.例如,对象刷新操作,对象缓冲池中修改过的对象如果要写回外存,必须先使该对象所在页存在于页面缓冲池中,对该页进行刷新,再由页面缓冲机制将修改刷新到外存.

双缓冲机制是对象拷贝时间和对象重定位时间选取的机制.通过下面的分析,可以看出对象懒惰拷贝与急切重定位相结合的机制是最科学的.

3.1 对象拷贝

对象拷贝是把对象从页面缓冲池中所在页面中拷贝到对象缓冲池中,有急切(Eager)拷贝和懒惰(Lazy)拷贝 2 种机制.

急切拷贝是在第 1 次访问对象时,就进行对象拷贝.采用这种机制,应该把对象缓冲池设置得比页面缓冲池大些.急切拷贝的弱点是在 2 种情况下效率不高:第 1 种是应用程序顺序浏览大量数据时,因为页面缓冲池配置的较小,因此增加了缺页中断;第 2 种是访问已经聚簇的对象.为了克服这些缺点,提出懒惰拷贝.

懒惰拷贝是在对象的所在页将从页面缓冲池中替换出去,而对象本身仍在进程的访问

集中时,进行对象拷贝.采用这种机制,对于对象的顺序访问,对象将在所在的页被替换之前从进程的访问集中删除,因此没有拷贝开销;而当应用访问的对象分散在大量的数据页中时,懒惰拷贝也能进行必要的对象拷贝.而且在这种机制下,对象的拷贝时间可与页面替换的内、外存I/O时间重迭,双缓冲的拷贝时间开销大大减少.因此我们采用懒惰拷贝.

3.2 对象重定位

对象重定位是指对象首先被拷贝到对象缓冲池,其所在页面已经被替换出去,而后来由于对其它对象的访问,又将该页面重新调入页面缓冲池中,这时要将原来在对象缓冲池中的对象重定位到其所在的页面中.对象的重定位也有急切和懒惰2种机制.

急切重定位是指只要对象所在页重新被调入缓冲中,就进行对象的重定位,并释放掉对象缓冲池中的对象的拷贝.而懒惰重定位将会尽可能晚地重定位,只有在对象必须从对象缓冲中替换出去时,才进行对象重定位.

采用懒惰拷贝和急切重定位相结合,完全消除了副本,只有当对所在的页面不在页面缓冲池中时,才会出现单独对象.这样也简化了2种缓冲池中的对象一致性维护.

4 小 结

本文主要介绍了Sambase对象存储系统中VMOM部分的思想、技术及实现,VMOM巧妙地利用了操作系统的虚存技术以及对象存储中先进的指针搅拌技术,为应用程序操作数据库对象提供了一个方便、高效的接口.不需要另外的编译程序,可用同操作临时对象相同的代码进行持久对象的访问.并使数据库的大小突破虚存大小的限制,一个事务中可同时操作的数据量仅受虚存大小的限制.此外还讨论了双缓冲机制在Sambase中的实现.

参考文献

1 石树刚,郑振楣.Sambase面向对象数据库管理系统,交通与计算机,1995,13(3):13~22.

COMBINING VIRTUAL MEMORY TECHNIQUES AND DUAL-BUFFER STRATEGY ——SAMBASE OBJECT STORAGE SYSTEM

GAO Yuan ZHUANG Yan ZHU Kan ZHENG Zhenmei SHI Shugang

(Department of Computer Science Wuhan University Wuhan 430072)

Abstract The approach presented in this paper uses VM(virtual memory) techniques of OS to allocate a buffer pool from VM whose size can be equal to the size of the whole VM. The approach is implemented as a C++ class library that can be linked with an application, requiring no special compiler support. Application program accesses objects via normal VM pointers, and manipulates objects using the same compiled code. Dual-buffer strategy based on this VM technique allows the size of database exceed the size of VM.

Key words Pointer swizzling, dual buffer, virtual memory object management.

Class number TP311.13