第6卷第6期
1995年6月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 6, No. 6
Jun. 1995

持久对象访问的自动捕获*

孙建伶　何志均

(浙江大学人工智能研究所,杭州 310027)

摘要　本文提出持久对象访问的自动捕捉技术,该技术是面向对象数据库系统在保持对象级的并发控制、版本管理和约束检查的条件下,实现持久对象与易变对象访问方式的一致性,进而实现面向对象数据库与程序设计语言的无缝结合的关键技术.

关键词　面向对象数据库,程序设计语言,集成.

数据库与程序设计语言的紧密集成是计算机软件领域多年来孜孜以求的目标.人们希望这样的紧密集成系统在保证计算完备性和资源完备性的同时,还消除了以往数据库与程序设计语言松散耦合中存在的编程范型和类型系统两方面的失配,进而从根本上提高数据库应用软件的质量和可维护性,降低开发人员的负担,提高软件生产率.持久程序设计语言(如PS-ALGOL)和数据库程序设计语言(如[illegible])都是在这方面所作的努力[1],但离目标尚远.面向对象为数据库和程序设计语言提供了统一的类型系统及双方共同感兴趣的东西(如模块化、可维护性、软件重用等),起到了数据库、程序设计语言集成的催化剂作用[2].面向对象数据库便是面向对象程序设计语言与数据库紧密结合的产物[3].

面向对象数据库的应用程序尽管也区分两类数据,即易变数据(驻留在易变存储器,即内存中,随建立它的进程的终止而消亡)和持久数据(驻留在持久存储器,即外存中,不随建立它的进程的终止而消亡,可由程序的多次执行而使用,也可由多个程序共享),但要求应用程序对持久数据和易变数据的访问方式保持一致,即对象访问时并不区分对象的持久性或易变性.比如有一函数,以类型T的对象作为形参,那么T的持久对象或易变对象都可以作为实参调用该函数.这就使得编程人员能以一致的观点看待、以一致的方式访问持久对象和易变对象,不仅大大减轻编程人员的负担,而且对于提高数据库应用软件的模块化、可维护性和可理解性都有重要意义.

OSCAR是我们开发的一个面向对象的数据库管理系统原型.OSCAR不仅致力于为新一代应用系统提供诸如复杂对象建模、工程事务管理、版本管理等高级功能支持,而且致力于数据库与面向对象程序设计语言C++的无缝结合.OSCAR利用了硬件及操作系统高级特征的支持自动捕获持久对象上的访问,实现了持久对象与易变对象访问方式的一致性,

* 本文1993-12-03收到,1994-01-17定稿

本研究项目得到国家自然科学基金和863计划CIMS主题的资助.作者孙建伶,1964年生,讲师,主要研究领域为数据库,人工智能,软件工程.何志均,1924年生,教授,博士导师,主要研究领域为人工智能,专家系统,CIMS.

本文通讯联系人:孙建伶,杭州 310027,浙江大学人工智能研究所

为数据库应用开发者提供了一个极为友善的编程界面.

1 持久对象与易变对象访问方式的一致性

持久对象与易变对象访问方式的一致性指的是所有对象,不管是持久对象还是易变对象,均遵从一致的访问方式和规则.比如类 Person 有属性 age,p 是指向 Person 类对象的指针,则"p→age=10"完成 age 属性的赋值,并不区分 p 所指对象是持久对象还是易变对象.

持久对象与易变对象访问方式一致性的实现,其困难之处在于编程方式的一致性隐藏实现上的不一致性.从编程人员角度看,持久对象与易变对象的本质区别仅是生存期不同.但是,从 OODBMS 的实现者的角度看,持久对象与易变对象的实现方法完全不一样.易变对象生成在易变存储空间中,甚至根本就不受 OODBMS 的管理.而 OODBMS 为了实施对持久对象的控制(并发控制和恢复、约束检查以及版本化),却必须在对象访问之前知道对象的访问方式(称为写前通知),以及对象修改后知道对象确实做了修改(称为写后通知).

目前各 OODBMS 进行写前通知和写后通知的方式大不一样. Zeitgeist、ONTOS 都需要通过显式调用系统提供的函数直接声明对象将进行的访问方式(读或写);Zeitgeist 通过调用系统方法 set_modified()、ONTOS 通过调用系统方法 PutObject 来进行写后通知. VERSANT 假定对象的缺省访问方式为读,写前通知通过根类 PObject 的方法 dirty()进行,而且不再进行写后通知(即假定进行过写前通知的持久对象都要进行写访问).这种通过编程人员显式调用函数进行写通知的做法存在着极大的弊端,不仅造成持久对象与易变对象访问方式和规则不一致,给编程人员带来额外负担,而且对应用系统的完整性、一致性也造成极大的隐患,因为如果编程人员忘了写前通知将会造成并发控制的错误;如果忘了写后通知将会造成已修改对象内容的丢失.然而,通过显式函数调用进行写访问通知从某些意义上讲又是必须的,而且是编程人员不得不接受的.因为,除非有硬件及操作系统高级特征的支持,否则不通过函数调用进行通知,OODBMS 就无法知道持久对象的访问方式.

2 硬件和操作系统高级特征的支持

硬件及操作系统的发展一直在提高着数据库系统的性能,改善和增强着数据库系统处理新一类数据的能力,并极大地影响着数据库系统的构造方式[4].OSCAR 实现持久对象访问的自动捕获利用了 SUN SPARC 工作站上的 Unix 系统版本 SunOS 的如下特征(也是 SUN SPARC 工作硬件特征的体现):

SunOS 的进程进址空间由页面组成; SunOS 的虚拟存储空间(virtual memory)由所有可随机访问的物理存储资源(如文件、Swap 区、处理器主存、其它随机存储设备)组成,称为虚拟存储对象; SunOS 提供系统调用将进程地址空间的页面同虚拟存储对象(通常是文件)的页面相映射.对这样映射的进程页面可以施加读保护或读写保护;如果某页上的写访问与该页的读保护方式冲突,硬件能够检测到,操作系统通过发出 SIGSEGV 信号反映这种冲突;SIGSEGV 信号可以由 OSCAR 登记的 SIGSEGV 信号处理程序捕获并处理;在 SIGSEGV 信号处理程序返回后,SunOS 保证重新执行引起冲突的那条指令.

大多数 Unix 版本,如 SVR4、OSF/1,Berkeley BSD 4.3 也提供了这种机制.OSCAR 在

对象缓冲管理中使用这一机制,实现了持久对象与易变对象访问方式的一致性.

3 方法与实现

3.1 OSCAR 对象缓冲管理

OSCAR 对象缓冲管理有两大功能,一是内存驻留对象的管理,包括对象缓冲区分配、对象的调进/调出等.另一个功能就是对象访问的自动捕获.它由 5 部分组成,即内存驻留对象表 ROT、缓冲区、缓冲区页表、缓冲区分配链表、空闲块链表,如图 1 所示.

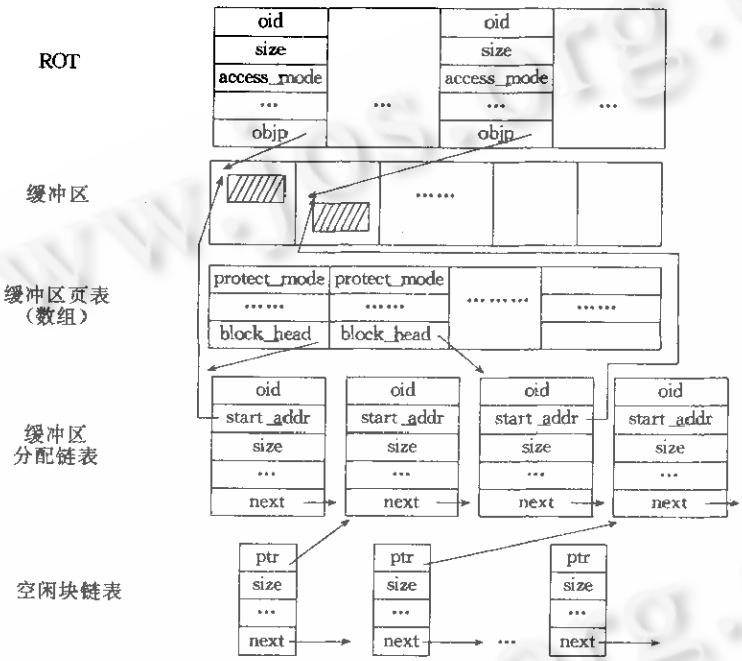

图1 OSCAR对象缓冲管理结构

(1) ROT 表

ROT 表登记了所有驻留在对象缓冲区中的对象,对持久对象的访问都要经过 ROT 表.ROT 表是以对象标识符 oid 为关键字的 hash 表,它的基本元素是内存驻留对象描述符 ROD.ROD 的主要信息包括对象标识符 oid、对象大小 size、访问模式 access_mode、对象在缓冲区中的地址 objp 等.其中访问模式 access_mode 表示对象自调入对象缓冲区后所经历的最高级别的访问模式,即如果经历过写访问,则为 WRITE 值;若未经历过写访问,则为 READ 值;对象最初生成时,其 access_mode 为 WRITE;对象刚调入对象缓冲区时,access_mode 为 READ.有关 ROT 表的调度策略和算法、缓冲区空间的分配与释放、对象数据库格式至 C++内存格式的转换等,不是本文要讨论的内容.

(2) 缓冲区

缓冲区是一块与虚拟存储对象(磁盘空间)相映射的进程的连续地址空间,它是一组连续页面的集合.可以通过 SunOS 的系统调用对其中的任何一页或连续几页施加读保护

(PROT_READ)或读/写保护(PROT_WRITE);缓冲区既是持久对象的驻留区,又是持久对象写访问的"感受区".OSCAR 保证任何对象的第 1 次写访问必定和对象所在页面的读保护方式冲突,从而引发 SIGSEGV 信号,由 OSCAR 登记的信号处理程序处理,从而感知对象的写访问.

(3) 缓冲区分配表

它按缓冲区地址顺序登记缓冲区中的每一块被各持久对象占居的连续地址块或缓冲区中的空闲块,每一个链表项的信息包括对象标识 oid(NULL 表示对应块为空闲块)、在缓冲区中起始位置及长度等.

(4) 缓冲区页表

它是一个数组,数组的元素个数与缓冲区页面数相等.数组中的每个元素的信息包括:对应页面的保护方式 protect_mode(PROT_READ 或 PROT_WRITE)、对应页面所包含的对象块和空闲块在缓冲区分配链表中的起始位置 block_head.由于缓冲区分配表按缓冲区地址顺序链接各对象块和空闲块,同一页面中的各对象块和空闲块在缓冲区分配表中必是连续的几块,因而循着 block_head 能快速判别某页的某一地址所处的块,从而判别该地址所属的对象的 oid.

假定缓冲区中某一地址为 address,缓冲区的起始地址为 address0,页面长度为 PAGESIZE,则地址 address 所在页面的序数 page_num(从 0 称起)的计算公式为:

$$page\_num = (address - address0) \bmod PAGESIZE$$

page_num 作为缓冲区页表(数组)的下标可以迅速访问到地址 address 所在页的详细信息.

(5) 空闲块链表

它按照空闲块的大小次序链接缓冲区各空闲块,这是为了缓冲区分配的方便.空闲块链表的每一项的信息包括空闲块大小 size,指向缓冲区分配链表的相应项的指针 ptr 等.

3.2 算法流程

对持久对象的访问是通过 OSCAR 为 C++提供的通用指针类 Reference 的→操作实现的,如"r→age=20(r 是通用指针类变量)".Reference 类的变量可以指向任何对象(持久对象或易变对象),它封装了对象标识 oid,重载了→等操作,使得 Reference 变量表现得如同一般的对象内存指针一样.

为了完整地描述持久对象访问的自动捕获技术,需要将 Reference 的→操作的流程、操作系统的处理流程、OSCAR 的 SIGSEGV 信号处理函数 handler 的处理流程结合起来考虑(尽管→操作和 handler 函数才是 OSCAR 要实现的):

(1) →操作按照 Reference 中的对象标识 oid 找 ROT 表中的一项.

(1.1) 如果没有找到,则按 oid 从数据库中找到相应对象,并读入内存,在缓冲区中分配一块,在 ROT 表中分配一项(可能引起 ROT 表的调度);将对象所在页(可能不止一页)置为读保护方式,将缓冲区页表中相应项的 protect_mode 置为值 PROT_READ;将对象 ROD 中的 access_mode 置为 READ.

(1.2) 如果能找到,则如果对象的访问模式(ROD 中的 access_mode)与所在页的保护模式(缓冲区页表项的 protect_mode)不一致,则利用 OS 系统调用将对象所在页置为与对

象访问模式一致的保护模式.相应的 protect_mode 值也置为与 access_mode 值一致.

(2) 此时要找的对象一定登记在 ROT 中(假如 oid 对应的对象存在),对应的 ROD 已经定位,则返回 ROD 中的 objp.

(3) C++程序按 objp 访问对象的属性.

(3.1) 如果实际是读访问,那么决不会与对象所在页的保护方式冲突,一次对象访问顺利结束.

(3.2)如果实际是写访问,那么可以区分两种情况:

(3.2.1) 如果对象所在页的保护方式为读/写保护(protect_mode=PROT_WRITE),则对象写访问也不会和对象所在页的保护方式冲突,一次对象访问顺利结束.

(3.2.2) 如果对象所在页的保护方式为读保护(protect_mode=PROT_READ),则对象写访问与对象所在页的读保护方式冲突,硬件能检测到这种冲突,反应在操作系统则是按信号 SIGSEGV 调用 OSCAR 登记的 SIGSEGV 信号处理函数 handler(handler 还同时获得出错地址等信息).handler 的处理流程如下:

(3.2.2.1) 按照出错地址算出地址所在页的页号;以页号作为下标找到缓冲区页表中对应的一项;按此项中的 block_head 找到对象在缓冲区分配表中的一项,得到对象 oid;算出对象所跨越的页面;利用操作系统调用将这些页面设置为读/写保护模式,并将页的 protect_mode 置为 PROT_WRITE;通过 oid 找到 ROT 表中相应的 ROD(必能找到),置 access_mode=WRITE;从 handler 返回.

当从 handler 返回后,OS 保证重新启动那条引起冲突的指令,回到第(3)步状态.此时由于对象的写访问与对象所在页的读/写保护模式一致,写访问顺利进行,一次成功的访问结束.

3.3 分析

从上面的算法流程可以看出,OSCAR 保证持久对象读入缓冲区后,只在第 1 次写访问时,而且一定要在第 1 次写访问时引起对象写访问与对象所在页的读保护方式冲突,操作系统通过调用 OSCAR 登记的 SIGSEGV 处理函数 handler 使 OSCAR 知道对象将进行写操作,而且必定要进行写操作(操作系统保证重新执行引起写操作冲突的机器指令).这等于既获得了对象的写前通知也获得了对象的写后通知.对象的读访问不会引发 SIGSEGV 信号,这是显然的;对象第 2 次及以后的写访问操作也不再引发 SIGSEGV 信号,这是通过在返回对象内存指针前开放对象所在页的读/写保护实现的.OSCAR 对第 1 次写访问的捕捉是通过在返回对象内存指针前关闭对象所在页的读/写模式(只开放读保护模式)实现的.对象所在页的读保护模式等于是一道警戒线,超过了这道警戒线(写访问)就会被捕获.除了第 1 次写访问 OSCAR 要做较多的处理外,持久对象以后访问的额外代价只可能化在持久对象所在页的保护方式的切换上.这种情况发生在同一页中有多个对象,而它们的访问模式又不相同时.消除这一代价的方法之一是将具有相同访问模式的对象分配在同一页中.

4 与 ObjectStore 的比较

ObjectStore[5]也利用了本文所描述的硬件与操作系统的高级特征,而且完全依赖于这

种特征设计其系统结构. ObjectStore 特别关注于系统的性能和从 C、C++到 ObjecStore 的自然过渡;这种自然过渡要求持久对象与易变对象访问方式的一致性. ObjecStore 基于虚存映射的体系结构确实使这目标得到了实现. 但是,ObjectStore 也有其难以克服的缺陷,这种缺陷来源于它的体系结构. ObjectStore 将进程地址页面直接映射到数据库页面,它对写访问的自动捕捉是针对整个页面,而不是单个对象. 因此,ObjectStore 只能以页作为并发控制和日记的粒度,无法实现任意持久对象的版本化,无法实施对象级的并发控制与约束检查.

OSCAR 只在对象缓冲管理模块中利用了硬件和操作系统的这种特征,对总体结构影响很小,易于系统的优化和扩展;OSCAR 实现了对单个对象访问的自动捕获,进而实现了持久对象和易变对象访问操作的一致性,同时保持了对象级的并发控制、版本管理和约束检查. 因此就对象的现有功能和潜在功能而言,OSCAR 都比 ObjectStore 丰富.

参考文献

1 Atkison M P. Types and persistence in database programming languages. ACM Computing Surveys, 1987, **19**(2): 105—190.

2 Moss J E B. Object orientation as catalyst for language—database integration. In: Kim W ed. Object—Oriented Concepts, Databases and Applications, Reading MA: Addison Wesley, 1989. 583—592.

3 Zdonik S B, Maier D. Fundamentals of object—oriented databases. In: Zdonik S B, Maier D eds. Readings in Object—Oriented Database System, San Mateo CA: Morgan Kaufman Publishers, 1990. 1—32.

4 Selinger P S. The impact of hardware on database systems. In: Goos S, Hartmanis J eds. Database Systems of the 90s, Berlin Heidelberg: Springer—Verlag, 1990. 316—334.

5 Lamb C *et al*. The objectStore database system. Communications of the ACM, 1991, **34**(10): 50—63.

TRAPPING THE ACCESS TO PERSISTENT OBJECTS

Sun Jianling He Zhijun

(*Institute of Artificial Intelligence, Zhejiang University, Hangzhou* 310027)

Abstract A technique for trapping the access to persistent objects is presented, which is a key for object—oriented database systems to realize the consistency of access to transient objects and persistent objects, and then to realize the seamless integration of databases and programming languages, in the condition of keeping the object—level concurrency control, version management and constraint checking.

Key words Object—oriented database, programming language, integration.