面向对象数据库系统 ObjectStore 概述

摘要:*ObjectStore 是一个面向对象数据库管理系统(ODBMS),它具有一个相当完整的语言界面,这个语言界面可以提供常规的 DBMS 功能:永久存储、事务管理(并发性控制和恢复)、分布数据访问和相关查询(associative queries)。*

ObjectStore ODBMS 被设计成适用于这样的应用领域:在大的对象数据库上具有复杂的操作,并且对象具有复杂的结构。例如 CAD、CAM、CAE、CASE、出版(publishing)和 GIS。这些应用领域的开发者要求使用方便、强的表达能力、可重用代码库,以及与主机环境的紧密结合。ObjectStore 对永久数据(应用程序运行结束之后仍然存在的数据)和暂时数据提供了一个统一的编程界面。

ObjectStore for UNIX 支持 RISC SYSTEM/6000 工作站上的 IMB AIX3.1 和 AIX3.2。它是唯一一个与 AIX SMIT(System Management Interface Tool)相结合的商品化的 ODBMS。这使得把 ObjectStore 安装和配置到 AIX 环境变得简单易行。ObjectStore 也可运行在如下环境:Sun OS 下的 SPARCstations、Ultrix 下的 DECstations、HP/UX 下的 HP700 系列,以及 MS Windows3.X 的 80386 或更高的 PC 平台。

应用程序接口

ObjectStore 支持三种编程接口:C 语言库接口、C++语言库接口和扩展的 C++接口——提供一个与查询和关系(relationship)机制更紧密的语言集成。最后一个接口要通过 ObjectStore 特有的 C+编译器。两个库接口可以通过其它第三方式的 C 或 C++ 编译器访问,因而提供了最大的可移植性。ObjectStore 体系结构的所有特征和性能都可以从三种接口得到。

ObjectStore 与 C++语言集成的关键是:永久性不是一个对象的类型的一部分。任何 C++数据类型的对象都可以暂时的分配在堆上或永久地分配在数据库中。这包括内部类型,例如整型和字符串,也包括任意用户定义结构(可以包括指针,使用 C++虚函数,多重继承等等)。特别是,它们不需要从一特殊的"永久对象"(persistent object)基类中继承下来,大多数其它系统是这样做的。同一类型的不同对象在同一个程序中可以是永久的或暂时的。

```
#include <ostore/ostore.hh>
#include <record.H>
main()
{
// Declare a database and an"entrypoint"into the
// database of type"pointer to department."
os_database * db;
persistemt <db> department * engineering_department;
// Open the database
db = os database::open("/company/records");
// Start a transaction so that the database
// can be accessed.
os_transaction::begin();
// The next three statements create and manipulate
// a persistent object representing a
// person named Fred.
employee * emp = new(db)employee("Fred");
engineering_department->add_employee(emp);
emp->salary = 1000;
// Commit all changes to the database.
os_transaction::commit();
}
```

图 1　操纵永久数据

图 1 给出了一个简单的使用扩展 C++界面的 C++程序。这个程序打开一个已经存在的数据库,创建类 employee 的一个新永久对象,把这个新雇员加到已存在的部门中,并把他的薪水置为 1000。关键字 persistent 指明一个存储类型,说明这个变量存放在特定的数据库中,永久变量把一个名字同永久对象相联,提供导航或查询的起始点。new 操作符的参数说明生成的 employee 对象必须在 db 数据库中分配。

操纵数据的代码看起来像的一个普通 C++程序,即使对象是永久的。薪水域(salary field)的更新只用一

个条简单的存储(store)指令。ObjectStore 自动设置读写上锁,跟踪哪些东西已经修改了,这有助于保持数据库一致性。对永久数据的访问保证是事务一致的(要么全部更新,要么什么也不更新),并且在系统发生故障时可以恢复。

除了 C 和 C++语言提供的数据定义的操纵机制,ObjectStore 还支持在一个事务之内访问永久性数据、集合(collecation)的类库、双向关系(bidirectional relationship)机制、优化查询机制和支持协同工作的版本机制。它也提供模式设计工具、数据浏览工具、数据库管理,应用程序编译和查错工具。

VMMA(虚拟存储映射体系结构)

ObjectStore 的 VMMA(Virtual Memory Mapping Architecture)允许用户处理单一地址空间和单一类型系统。这种机制允许同主机环境的紧密结合,容易使用和重用现存库。由于使用相同机器代码来处理对暂时和永久数据的引用,VMMA 提高了 ObjectStore 的性能。其它 ODBMS 体系结构对永久对象的引用用软件方法处理,降低了性能。

数据库程序设计语言的一个基本操作是查找和使用由一源对象(source object)指向的目标对象(target object)。这个操作叫做〝取一个对象〞或逆引用(dereferencing)一个指针。ObjectStore 使得逆引用一个指向永久对象的指针和逆引用一个指向暂时对象的指针尽可能一样快,即一条 Load 指令的速度。这种特点提供了同宿主语言的透明结合。因此,逆引用一个指向永久目标(target)的指针必须同逆引用一个指向暂时目标的指针以同样方式编译(以一条 Load 指令),而不需要用额外指令来检查目标对象是否从数据库中检索出来了。

ObjectStore 的 VMMA 在商品化的 ODBMS 产品中显得独特。ObjectStore 利用了 CPU 的虚存硬件和标准操作系统的 API。VMMA 允许 ObjectStore 给虚存的任一页面设置成不可访问、只读或可读、写。当一个 ObjectStore 的应用程序逆引用一个指针,而这个指针所指向的目标对象又没有检索到客户机时,硬件检测到一个访问违法,操作系统把它当成内存错误(memory fault)返回给 ObjectStore。ObjectStore 从服务器中检索目标对象,并且把它放到客户缓冲区中。然后它呼叫操作系统,把页面的保护设置成只读,从内存错误处重新开始。

以后对同一个目标对象或在内存中其它对象的读都由一条指令完成,而不会引起错误(fault)。对目标对象的更新引起错误(fault),使得对象的访问模式和锁升级到读、写。在 ObjectStore 的指引下,操作系统使用常规系统调用来处理应用程序中所有虚存映射和地址空间操作。

ObjectStore 服务器为永久数据提供长期的存贮池,并且根据客户的请求存贮和检索对象。它使用类似于常规 DBMS 中使用的技术来处理并发性控制和恢复。它以每个页面提供两阶段上锁(读、写上锁)。恢复基于一个登录文件,使用提前写登录协议(write-ahead log protocol)。涉及一个以上服务器的事务使用两阶段提交协议来协调。服务器也提供磁带备份。

ObjectStore 在虚存中维持一个客户缓冲区(client cache),其中包括最近被使用的数据库对象。当应用程序发一个内存错误中断时,ObjectStore 决定被访问的对象是否在客户缓冲区中,只有当对象不在客户缓冲区中才访问服务器,当运行在不同机器上时,客房和服务器通过 LAN 通信,当运行在同一机器上时,使用一些较快的机制,如共享内存,局部套接字(local socket)。

集合机制(Collection Facility)

ObjectStore 以类库的形式提供了一种集合机制。集合是一个类似于传统程序设计语言中的数组和关系数据库中表格的抽象数据类型。然而与数组和表格不同的是,ObjectStore 的集合机制提供了一系列行为,包括有序集合(链表)及允许或不允许重复元素的集合。

ObjectStore 的集合机制对每一套抽象行为具有不同的内部表示。依照不同的集合大小和访问方式来选取内部表示可以改善性能,用户或者选择想要的内部表示,或者只是描述使用方式,两种情况下都能够使 ObjectStore 透明地选择效率最高的表示。这些性能调节机制使开发者从编写数据结构进步到描述访问方式。

关系机制(relationship)

ObjectStore 的关系机制有助于建立复杂对象的模型,例如部件层次关系、设计、文档和多媒体信息。可以把关系想象成一对互逆指针。如果第一个对象指向第一

个对象,第二个对象也有一个逆指针指向第一个对象,关系机制负责维持这些指针的一致性。例如,如果关系中的一方被删除了,则另一方指向它的指针被置为零。ObjectStore 支持一对一,一对多和多对多关系。

从语法上来说,关系机制可以象 C++语言中的数据成员来访问,但是更新一个关系的值会引起逆关系(inverse relationship)的值也更新。因此,两边就能保持相互一致。如果应用程序更新某一边,ObjectStore 的关系机制保证另一边也被更新。

查询(Queries)

查询是操作在一个或多个集合上的表达式,其结果是一个集合或指向一个对象的引用。ObjectStore 的查询机制同主语言紧密相联,它不象关系数据库。在关系数据里,查询是用一种特殊语言 SQL 来表达的。SQL 有它自己的变量和表达式,这些变量和表达式在语法上和语义上都不同于主语言中的变量和表达式。两种语言中变量的结合必须显式进行。

ObjectStore 处理查询表达式的方式不同于处理其它表达式。实现的策略——循环和检查谓词的效率对大的数据库来说很差是显而易见的。关系数据库用索引来提高效率,而 ObjectStore 的查证优化器检查一系列策略并选择最有效的一种。ObjectStore 也使用索引,但是它们比关系数据库中的索引更为复杂,可以给对象和集合来设置索引,而不单单是对象的域。

为关系型数据库开发的优化技术不怎么适合于 ODBMS。关系数据库模式是经过规范化的,其中没有嵌套集合(set)和指针。查询是通过连接(join)表达式(一种涉及多个表中的行的表达式)来进行的,因而优化器把它的绝大部分时间花在对带有多个连接表达式的查询的求值。在 ObjectStore 中,查询倾向于在小数目的顶级(不是嵌套的)集合上进行,通常只有一个。选择谓语(selection predicates)涉及遍历对象和嵌套集合的路径。这些路径表达了与关系型数据库查询中的连接表达式同样的联系。因为路径已经存在于数据库中,对象间引用和嵌套集合使得连接优化不成大问题。

版本机制

ObjectStore 提供一种机制使得一组用户以使用方式共享数据(有时被称为组件)。一个用户可取出(check out)一个对象或一组对象的版本做修改,把这些修改记入 (check in)主数据库以便使别的组员也能看见他们。在这期间,其它用户可以继续使用以前版本,而不会由在共享数据上的并发性冲突而妨碍其使用,也不要考虑这种编辑会话(editing sessions)会持续多长时间。这种在私有的,取出的版本上的护展编辑会语叫做长事务(long transactions)。

如果其它用户想做并发修改,它们能取出同一个对象的分枝版本 (alternative version)并在其上工作。不存在并发性冲突,即使用户在同一对象的不同版本上操作。以后,不同版本可以汇合(merge)在一起,解决由并发性修改带来的不一致性。汇合是困难的,开发者必须针对不同应用程序来实现。

通过设置私有工作空间(wordspace),用户可以控制使用哪个对象或一组对象的哪个版本。用户也可以使用工作空间来有选择性地共享他们正在进行的工作。工作空间可以从其它工作空间继承下来。例如,一组从事 CPU 设计的工程师可以设置一个工作空间,在其中创建新版本。只有当他们完成了设计,他们才能把完成的版本记入 (checkin)到公共工作空间(corporate workspace),使得制造组可以看到它们。在工程组的工作空间里,子组或单个组员可以有许多工作空间。正如整组通过把完成的版本记入公共工作空间来使得它们的工作可以被制造组使用,单个设计人或设计组可以通过把他们的中间版本记入它们的共享工作空间来使得他们正在进行的工作被别人使用。

一个对象的持久性和版本都不依赖于类型。任何类型的实例都可以版本化,相同的代码可以操作版本化或非版本化的实例。应用这个特征可以把针对没有版本概念的一段代码用于版本化数据。使用版本化数据的程序无须区分版本化,永久和暂时数据。

性能

控制对象放在数据库中什么地方可以提高性能。通过把被经常引用对象集中在客户缓冲区可以提高性能,因为访问对象时只有少数页面被传送了。ObjectStore 把数据库分成叫段(segment)的区域。每当应用程序创

(下转第 18 页)

(上接第 64 页)

建一个新的永久对象时,它要指明对象将创建在哪个段。应用程序可以随意创建多少个段。段中的数据从服务器传送到客户机中时,或者整段传送,或者一次只传送一个页,依赖于应用程序控制 per-segment 标志的设置。

1992 年 7 月份推出的 Release2.0 将是第一个支持对象级簇集(clustering)的 ODBMS。这种对象级簇集允许把单个对象一个挨一个地放在用户可控制的簇集中,并且可控制每个簇集中的对象数目。这种机制有两个优点:首先,通过把需要一起检索的对象聚集在一起,用户可以使访问一组对象时所需磁盘 I/O 操作最少;另一个方面,这种机制有助于管理和消除数据库"热点",而没有对象级上锁技术(objedt-level locking techniques)带来的性能下降。对象级簇集保留了 ObjectStore 的易于使用的程序开发模式,同时避免了给每个小时对象上锁带来的额外开销。

(李海飞译自<<*AIXpert*>>)