计算机科学 1997 Vol. 24 №. 2

对象库中的物理存贮重组技术

金蓓弘 冯玉琳

(中科院软件所基础部 北京 100080)

摘 要 Objectbase has been given more and more attentions in recent years, especially on its performance. This paper focus on physical storage mechanism in objectbase and surveys clustering strategies and garbage collection policies, which have significant effects on system performance.

关键字 Objectbase, Clustering, Garbage collection

近几年来，对象库在 CAD、CAM、CIMS、CASE、GIS 等应用领域中显示出来的生命力是有目共睹的，但对象库能否被大家完全接受很大程度上还取决于它的性能。对象库的物理存贮由于对象的创建、删除、修改而不断地变化，是影响对象库性能的一个重要因素。本文将介绍两种技术：聚簇技术和垃圾收集技术，它们对对象库的组织方式和整个系统的性能有很大影响。

聚簇策略

对象库的物理组织方式可由聚簇策略加以调整。对象的聚簇问题本质上是一个将对象按什么样的顺序放在磁盘页上的问题，它的目标是将相关对象尽可能放在一起，减少磁盘I/O 的次数。对对象进行聚簇也能提高主存缓冲区的使用效率。聚簇应用程序所需访问的对象，可使得一页中为应用程序所需的对象的百分比提高，从而减少页失配率，减少需调入内存进行缓冲的页。事实上，对象聚簇性的好坏影响到整个系统的实际负载大小、服务器的开销、并发控制和恢复性能。例如，100 个不同的对象，假设其平均大小为 200 字节。它们既可放在 100 页上，也可聚簇在一起，放在 5 页上(假设一页的大小为 4K 字节)。那么，在最极端的情况下，对同样的应用程序，后者所需的客户/服务器交互是前者的 1/20，后者所需的客户方的缓存空间是前者的 1/20，后者由于修改动作而引发的要做事务日志的页是前者的 1/20，后者要获得的页级锁是前者的 1/20，而且事务冲突的可能性也比前者要小得多。

在讨论具体的对象聚簇方法之前，要强调的两点是：

1)聚簇策略的性能与缓冲区的配置和管理方法有直接的关系。这里，给出一个例子说明仅有聚簇是不够的。假设有六个对象 A,B,C,D,E,F，应用程序的访问模式为$(AB)^{99}(CD)^{99}(EF)^{99}$，即先访问 A，再访问 B，再访问 A，重复 98 次后访问 C，依次类推。这些对象按某种聚簇算法排列后分放于两页中：[A, B,C…][D,E,F,…]。若这时主存缓冲区只能放一页，则会引起 198 次页失配。如果这时系统缓冲区大于一页，就会大大提高缓存命中率。

2)聚簇的方式方法与 OID 的表示形式有关。聚簇过程涉及将对象从初始页传送到另一页，这对代理键形式的 OID 是不成问题的，因为这种类型的 OID 与位置无关，而若 OID 是物理地址型的，即它们与物理位置有关的话，问题就没有那么简单，因为对象移动的同时需要调整所有指向它的引用，从而保证这些引用的正确性。为了做到这一点，就要求或是在对象处保留所有指向被引用处的指针，用它找到引用该对象的地方，或是采用扫描整个数据库的办法，来查找该对象的引用。不论哪一种方法，在性能上都不尽人意，因此，具有物理型 OID 的系统一般不允许动态重聚簇。

1. 聚簇的时间和范围

进行聚簇的时间可分为：1)在对象创建的时候，根据用户的提示。例如，用户要求"尽可能靠近对象 Obj1"，那么系统将把该对象与 Obj1 放在同一页上或相邻页上。这时对系统而言，用户的提示是仅有的有用信息，故要求用户要清楚地了解对象的访问行为。2)在系统运行一段时间后，进行脱机或联机重聚簇。这时是利用对象的继承和结构语义以及系统运行时收集到的统计数据，来决定如何聚簇。

对象库中对象被聚簇的范围可用参数加以限制，参数的取值类似 Cluster_within_buffer, 2.IO_limit, 10.IO_limit, Within_DB。这说明放置对象最终位置的页是限制在缓冲区中，还是可通过有限次的

磁盘查找，或是在整个数据库中。设置该参数的目的是为了控制聚簇算法的开销，例如，Cluster_within_buffer 就完全避免了物理 I/O。

2. 各类聚簇算法

2.1 基于类型的算法 该类算法的核心思想是将同一类型的实例聚簇在一起，用户指明要聚簇的对象类型的顺序，系统根据这些类型信息来决定如何放置对象。有时，用户还可在同一类型的对象上再加上若干限制，使类中的对象位置有章可循，如指明类中的对象是按某一方法值或某一属性值的升序或降序排列，或指明是按对象的创建时间早晚排列。施加后一种限制的原因是我们认为同一期间创建的对象很可能在应用程序中被同时访问。

2.2 基于对象图的算法 首先给出对象图的定义：

对象图中的结点表示的是对象，对象与对象之间的某种联系构成了联结两个对象结点的边。

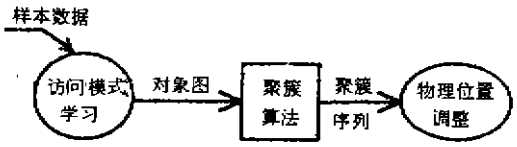
在聚簇算法中，我们用对象图来反映应用程序的访问模式，用图的方式表示访问模式具有简洁紧凑的特点。通常有三种类型的对象图：

1)对象结构图：该图中的边代表从一对象到另一对象的引用，表征的是对象的结构信息，从语义上区分有 is_a 和 is_part_of 两种边。该图不带任何统计信息。应用程序的访问模式可用用户定义的安置树来模拟。

2)统计对象图：在对象结构图的结点和边上加上权重。这些权重值是从一组样本数据中学习得到的，与样本数据中对象和引用出现的频率相同。

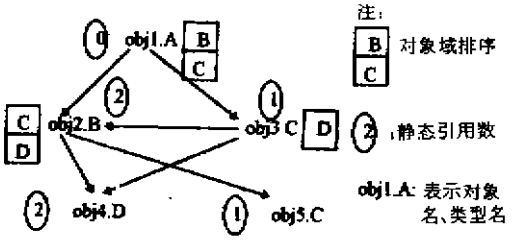
3)Markov 链图：这里的思路是将应用程序对对象库的访问模式看成是一个随机过程，我们首先根据样本数据估算出这个随机过程的参数，然后根据这个模拟访问模式的随机过程，决定如何将对象映射到页中，以减少对象访问时跨越页边界的可能性。具体的是用 Markov 链来模拟访问模式。Markov 过程是一族相关的随机变量，在此用它模型化对象请求，在时间 t 请求对象 x 的可能性完全地取决于最近一次的对象请求。由于对象请求的时间参数和对象转换的状态都是离散的，故该 Markov 过程是 Markov 链。对于 n 个可访问的对象，对象访问的(马式链)转移概率矩阵至多只有 n^2 个参数，这些参数可从样本数据中估算出来，然后将该 Markov 链转换成有向图的形式，系统中的可访问对象是图中的一个结点，若一对象在另一对象被访问之后可被访问，则用一有向边表示。结点和边的权重则直接从转移概率矩阵中得到。

基于图的聚簇算法的实质就是依据上述对象图做某种形式的图遍历，遍历的结果是对象的聚簇序列。系统将按照这个对象聚簇序列将对象放到页上。在非回溯式算法中，聚簇序列中对象的顺序就是图遍历中对象被访问的顺序，这时系统可在对象被访问到的时候就调整它的物理位置。下图是整个聚簇过程的示意图：



根据不同的对象图，有不同的聚簇算法：

1)采用最常用的图遍历算法。宽度优先遍历(BFT)和深度优先遍历(DFT)，该算法主要用于对象结构图。在 DFT 中，来自不同深度的对象放在同一/相邻页中，而同一父亲的两个后代可能隔开很远，因此，结点排序的顺序对 DFT 的遍历结果影响较大，直接影响聚簇的效果。根据在遍历时选取下一个结点的不同标准，BFT 和 DFT 各有两种变种：a)由对象类型定义中规定的对象域的顺序来决定下一个该遍历的对象。该信息一般由编译器负责记录。在下图中，按对象域排序的原则，DFT 的结果是 obj1，obj2，obj4，obj5，obj3。BFT 的结果是 obj1，obj2，obj3，obj5，obj4。b)根据系统静止时引用该对象的指针数(称为静态引用数)来决定。一般按静态引用数的升序排列比按降序排列要合理，因为，常规情况下，不被其他对象引用的对象很可能是应用程序要访问的对象。在下图中，按静态引用的升序原则，DFT 的结果是 obj1，obj3，obj4，obj2，obj5，BFT 的结果是 obj1，obj3，obj2，obj4，obj5。



2)采用安置树(Placement Tree)遍历,该方法用于遍历对象结构图。所谓安置树是这样一颗树:安置树的结点是类型名,安置树的边表示的是两个类型之间的引用关系。它事实上是系统模式图(即类关系图)的一个连通子树。安置树的组成完全由用户根据应用程序的访问模式指定,在用户指定了安置树后,系统便以对象图中具有安置树树根类型的对象为遍历的入口点开始遍历,遍历是沿着安置树的边(按BFT或DFT方式)依次访问对象图中具有安置树结点类型的对象。该遍历方式也可以诠释成根据安置树中的类型信息(属于模式层次的),在对象层次上即在对象图中找与安置树相匹配的对象树,最终将所有被匹配的对象聚簇在一起。注意,按这种方式形成的被匹配的对象可能已经被聚簇,原因是有对象共享的情况,故最终被聚簇的对象是还没有被聚簇的匹配对象。

我们还可以给安置树结点设置权重,一则可用它做为遍历安置树时选取下一个结点的标准,由它决定下一个要访问的对象的类型。二则可用它决定在遍历到的对象在一页中放不下时,哪一个对象宜于放在一页中。

该算法的复杂度与BFT、DFT相同,均为O(n)。

3)采用PRP(Probability Ranking Partitioning)方法,该方法可用于统计对象图和Markov链图。

该算法可概括如下:选择数据库中最经常使用的对象放入空的一页,然后考虑该对象与未聚簇对象之间的访问概率。可能性最大的对象被放入该页,重复进行该过程直至所有对象被重排序。还有一种PRP的改进型,其主要思想如下:选一个未聚簇的对象,在当前分区还有空间时,考虑可访问分区当前内容的所有对象,即从分区内的对象结点可达到的所有对象结点,注意,对对象统计图和Markov链图,这些对象是不一样的。根据这些对象访问该分区中所有对象的全概率,选择最有可能使用该分区的对象,将它放入该分区,重复该过程,直至分区满了。随后考虑下一个未聚簇的对象。由于该算法不需要回溯,故复杂度较低,为O(nlog n)。

4)采用图分区法。所谓图分区法是将对象图分区,划分成统一的页大小,使得分区中的对象能够一簇一簇地放到页中。它主要用于Markov链图。用标准的Kernighan-Lin图分区算法来对Markov链图进行划分,该划分的结果可使得任意两个属于不同分区的结点进行交换是不会产生更低开销的。该算法的性能较好,但它不是顺序地聚簇,故算法复杂度较高,为$O(n^{2.4})$。

各种聚簇算法都有其适应范围,如系统的缓存较大,又追求性能稳定,那么采用PRP算法较合适。如系统是从一冷缓存开始遍历一小部分数据库,这时拟采用安置树算法,但若应用程序涉及很多以不同的路径访问同一对象,由于安置树不能很好地刻画这类访问模式,这时用安置树的效果就不是很理想。若在短时间内系统的访问模式变化剧烈,那么聚簇与非聚簇在性能上的差别并不是很大,故选择聚簇算法要与实际应用的访问方式、系统追求的目标相适应。

存储回收策略

另一种引起对象库物理存贮变化的情形是由于垃圾收集。系统收集对象库中的垃圾,即可回收无用对象占用的存贮空间,又可借机对对象库空间进行重组。

什么是对象库中的垃圾?对这个问题有两种不同的观点:一是将系统中从永久根集合开始不能到达的对象看成垃圾。垃圾收集要做的就是将这些空间回收。在数据库系统中,永久根是下列二种情况的并集:(1)数据库对象图中的根对象,(2)包含具有指向数据库指针的任一活动事务的程序变量。另一种观点认为对象库中没有垃圾可言,因为任一永久对象都可看成是某一元类,例如元类Object的实例对象,所以不存在垃圾回收问题。不过,目前许多对象库都考虑了垃圾回收策略。

1.考虑因素

在客户/服务器体系结构下的垃圾回收策略要解决下列关键问题:

(1)保证回收所有的垃圾,并尽可能地提高对象引用的本地性,消除磁盘碎片。

(2)回收过程应是动态增量式的。由于回收的是磁盘上的数据,故要求算法能使得系统的I/O开销较小,运行的效率高。

(3)垃圾回收算法要能维护事务的语义,这包括两个问题:(a)垃圾回收器必须与并发的事务共存,并且不能影响这些事务的执行。(b)我们认为,不可达性是对象的一个稳定的性质,但事务回滚却与不可达性是相违背的,故要解决在存在事务回滚的情况下如何正确地识别并回收垃圾的问题。

(4)垃圾回收策略必须与数据库的恢复策略相兼容。

(5)恰当的垃圾回收频度,垃圾回收频度可由用户预先指定,或是由系统决定,后者是指由系统根据某一事件如系统内的自由空间低于某一阈值而触发垃圾回收动作。需要指出的是垃圾回收频度自动地与操作数据库的应用程序行为相适应才是最理想的情况。

(6)垃圾回收策略必须适应客户/服务器这种结构,减少由于垃圾回收而做的客户和服务器的I/O通讯,避免客户方缓存数据和服务器方缓存数据的不一致性,还要充分考虑客户和服务器双方的缓冲区管理方法对垃圾回收的影响。例如,客户方可以采用"Steal"缓冲区管理策略,即在事务执行时客户方的脏页(指被修改的数据页)可按任何一种顺序被回送到服务器方。这时若客户方的应用程序修改对象的跨页的指针引用,那么使用普通的垃圾回收策略就可能会产生悬挂的指针,从而引起系统崩溃。由此可见,垃圾回收策略与系统的并发控制、事务处理、缓冲区管理、恢复策略都有紧密联系。系统特殊的事务模型以及客户方Steal式缓冲区管理方式、客户-服务器的异步交互、避免日志的若干优化技术(如块拷贝)、流水型恢复机制都是对传统垃圾回收机制的一个挑战。它们的实现技术的不同也将影响垃圾回收策略的效率,例如同一垃圾回收策略在并发控制机制用常规2PL实现和非2PL实现情况下会呈现显著的不同。

2. 常见算法

在客户/服务器下的基本的垃圾回收算法有下列几种:

1)引用计数法:引用计数法分两个阶段,头一个阶段是连续计数阶段。服务器方保存对象的引用关系。最初的对象引用关系由服务器方遍历永久对象图得到。由客户方将所有影响引用计数的操作记录在事务提交时传送给服务器方。当磁盘上的自由空间量低于某一预定义的值或是为了使客户方应用程序的存贮分配量达到某一值。系统进入第二个阶段:周期性收集阶段,由服务器方确认对象的不可达性。然后回收这些对象的空间。回收过程可与客户方活动相并行。这是因为客户方的活动不会将垃圾再转换成有用的对象,只会产生更多的垃圾。由于在系统崩溃时,服务器方可通过遍历永久对象来重新生成对象引用关系,故该方法不影响常规的数据库恢复策略。该方法的致命弱点是无法收集具有环式引用链的对象垃圾,如相互引用的对象垃圾,而且存贮与维护对象引用关系的开销较大,也无法消除磁盘碎片。

2)标记加清除法(Mark And Sweep):该方法的基本思想是从系统的永久根集合开始遍历整个数据库,所有从永久根可达的对象被标记成活的。然后在清除阶段回收所有未标记的对象。Mark And Sweep方法原来用于程序语言,直接用于数据库上是不合适的。例如某些复杂的Mark方法涉及到链的反向,如果针对磁盘则I/O量大,故要做适当的改进。增量式的Mark And Sweep算法是这样工作的:Mark阶段,服务器方运行一收集器,以深度优先方式遍历对象库,用三色机制来记录遍历的过程,遍历的结果记录在位于内存的Color Map中,客户方可同时运行应用程序,由Write Barrier检测客户方对象的引用关系的变化,并送与服务器方,由服务器方再对这些对象做遍历。在Mark阶段的最后,由具体的并发实现技术决定服务器方是否要与各客户方确认一下。Sweep阶段先检查Color map上某页的情况,若该页上无垃圾则跳过该页,若该页上全是垃圾则释放该页,这两种情况都不直接读该页。否则,读入这页,做Sweep工作。除了常规做法外,还有一种方法是将Mark阶段和磁盘空间管理合并,在启动Mark阶段时,设定整个磁盘空间都是空闲的,做对象Mark的同时,重新构造磁盘的空闲列表。这种做法实际上是避免了单独的Sweep阶段。标志加清除法方法的另一变种是分区式标志清除法。

该方法在出现系统故障后的恢复策略:(1)若在Mark阶段出现系统故障,由于没有对数据页做任何修改,故无需恢复。(2)为解决在Sweep阶段出现系统故障,我们可以在Sweep一页时,在日志中记录Color Map中对这一页的垃圾收集记录,以便由系统重启后的自动恢复过程保证整个系统仍处于一致的状态。

3)拷贝法(要求对象标识符是逻辑型的):该方法将数据库空间划分成相等的两部分:FromSpace和ToSpace,服务器上的收集器不断地将活的对象从FromSpace拷贝到ToSpace,然后修改(Oid,磁盘地址)映射表中该对象的磁盘地址,使得将来对该对象所有的引用都直接指向ToSpace。对(Oid,磁盘地址)映射表的修改要在事务日志中做记录,以备在出现故障时做恢复用,所以该方法可与标准的数据库恢复机制相兼容。该方法也允许服务器方的收集器与客户方的应用程序并行执行。该方法的好处是在做垃圾回收时,可以运用聚簇策略重聚簇对象。其缺点是在正常执行时,只有一半的存贮空间可用。这在

计算机科学 1997 Vol. 24 №. 2

类比推理综述*)(下)

徐冬溶[1] 潘云鹤[2] 张 畅[2] 王 选[1]

(北京大学计算机研究所 北京 100871)[1] (浙江大学人工智能研究所 杭州 310027)[2]

4 类比能力的发育

隐喻与类比的关系十分密切,可以说是类比的一种特例。为了研究类比推理,深刻地理解类比、隐喻之间的关系以及这种能力的发生和发展是十分重要的。然而这项研究的成果并不太多,研究者共知的事实只是,即使很小的小孩子也常表现出类比、隐喻(Metphor)的语气。关于隐喻和类比能力发展的研究常常引用一些相当复杂的任务,要求被试儿童针对一给定上下文产生或选择一个合适的比喻,或解释测试者提供的比喻。主要工作集中在6岁至青春期这一区间。但是研究者却常常忽略了这样一个问题,除了隐喻能力,还有很多因素会影响测试,测试的设计应尽量去加以避免或消除。

(1)被试在以前生活中积累的经验,会左右其隐喻的使用。

(2)被试缺乏产生隐喻域或应用隐喻域中的概念性知识,并不表示其隐喻能力欠缺。

(3)测试者的要求有时暗示了被试,产生误异。

(4)为了便于事后的归类和分析而提供的标准化选择,掩盖了被试本身的隐喻能力的表现。

Verbrugge[Ver 1975]在分析隐喻和类比时将之分解为三个部分:主题和与主题相比较的载体(Vehicle),以及主题和载体的共同点一背景。他发现这种作为共同特性的背景内容在整个隐喻、类比的记忆中发挥着重要作用[Ver1973,75]。Gentner

---

数据库系统中简直是不能允许的。该算法的一种改进型称为分区式拷贝法,是将垃圾回收的范围限制在一小区域中,数据库的空间首先按逻辑或物理的标准划分成若干分区。要进行垃圾回收的这个区域一般被认定是在收集器的作用下能提供最大的自由空间,若分区中存在与其它分区的对象引用环,那么垃圾回收过程可能失败,这可通过扩大分区大小(将原来两个分区合并成一个)来解决,分区大小可扩大至整个数据库变成一个分区,保证垃圾回收工作的顺利进行。后二种算法也称为Tracing-Based垃圾回收法。

聚簇和垃圾回收是进行对象库存储重组的核心技术,也是目前对象库研究的热点之一。基于图分区的聚簇算法虽然聚簇效果最好,但由于其复杂性而未能在实际系统中采用,各类垃圾回收策略也面临着如何与应用程序行为相适应的挑战,因此,灵活、高效、实用的聚簇和垃圾回收技术将是一个有待进一步研究的目标。

参考文献

[1] Pamela Drew et al., The Performance and Utility of the Cactis Implementation Algorithms, Proc. of the 16th VLDB Conf. 1990

[2] James W. Stamos, Static Grouping of Small Objects to Enhance Performance of a Paged Virtual Memory. ACM Trans. on Computer Systems, 3(2)1984

[3] Manolis M. Tsangaris et al., A Stochastic Approach for Clustering in Object Bases, Proc. of ACM SIGMOD 1991

[4] Eillis E. Chang, Rendy H. Kats, Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-oriented DBMS

[5] Manolis M. Tsagaris et al., On the Performance of Object Clustering Techniques, Proc. of ACM SIGMOD 1992

[6] Margaret H. Butler, Storage Reclamation in Object oriented Database Systems, Proc. of ACM SIGMOD 1987

[7] Voon-Fee Yong et al., Storage Reclamation and Reorganization in Client-Server Persistent Object Stores, Proc. of IEEE Conf. on Data Eng. 1994

[8] Laurent Amsaleg et al., Efficient Incremental Garbage Collection for Client-Server Object Database Systems, Proc. of the 21th VLDB Conf. 1995

---

*)本文受到国家自然科学基金、国家"八五"攻关项目、国家"八六三"计划及攀登计划资助。