一种面向对象数据库继承层次索引配置方法①

刘彦宇[1] 章杰鑫[2] 陈晓辉[1]

(1.桂林理工大学 信息科学与工程学院 广西 桂林 541004)

2.广西经济管理干部学院 图书馆 广西 南宁 530007)

摘 要： 如何平衡面向对象数据库(OODB)索引的检索性能、维护开销、实现复杂度已经成为一个影响 OODB 性能的关键问题。提出一种 OODB 继承层次索引配置方法。通过开销模型评估各种索引配置的检索开销、存储开销、更新开销，根据数据库真实使用情况选择最优索引配置。通过实验，验证这种方法的准确性，并且同其他常见索引策略作对比。结果表明该方法可以为 OODB 提供性能最优的索引方案。

关键词： 面向对象数据库;继承层次索引;索引配置;开销模型

Approach for Index Configuration of Inheritance Hierarchy in Object-Oriented Database

LIU Yan-Yu[1], ZHANG Jie-Xin[2], CHEN Xiao-Hui[1]

(1.College of Information Science and Engineering, Guilin University of Technology, Guilin 541004, China;
2.Library, Guangxi Economic Management Cadre Colleges, Nanning 530007, China)

Abstract: How to balance the retrieval performance, maintenance cost and implementation complexity becomes one of the key elements in the research of OODB indexing. This paper proposes an approach for the indexing configuration of inheritance hierarchy. It estimates retrieval cost, storage cost and updates cost for all kinds of index configurations through cost model, and chooses optimal index configuration based on real database usage. The accuracy of this method is tested through experiments, and the performance of this method is compared with other indexing scheme. The results show that the proposed method can provide optimal indexing for OODB.

Keywords: OODB; inheritance hierarchy index; index configuration; cost model

OODB 中，一个类可以特殊化成一系列递归子类，形成一个继承层次。因此，查询目标既可以限定在单个类上，也可以扩展到以该类为根的继承层次上。结果导致 OODB 索引不仅要支持单个类上的对象检索，还要支持这个继承层次上所有对象的检索[1]。近年来，提出了很多支持继承层次的索引技术。①类层次索引(CH-Tree)[1]，这种方法为继承层次上所有类维护一棵 B+-Tree。缺点是，当查询目标针对继承层次中低层的类时，要扫描大量多余数据。②B+-Tree 链接结构：H 树，HcC 树，CG 树。由于结构实现复杂，又不能确保性能的可靠增强，这些方法都没能很好的应用到 OODBMS 中。③类分组(CD)方法，一种基于 CH-Tree 的扩展策略。像 CH-Tree 一样，仍然在根类上建立包含继承层次上所有类的一个索引，除此之外，通过 CD 算法把一个继承层次划分成许多类分组，在类分组上建立索引。查询时，通过合并这些分组索引来检索一个类的目标对象，这样可以避免像 CH-Tree 一样扫描多余数据。④基于数据库统计数据的方法[2,3]。⑤加速范围查询的二维索引结构，-tree，MT-index，2D-CHI 等。但是到目前为止仍然没有一种策略可以和满外延(full extent 以下简称 FE)索引策略[4]的检索性能相当。一个类的 FE 是该类及其全部子类的对象集合。FE 索引

①收稿时间:2009-11-18;收到修改稿时间:2009-12-26

策略为每个类的 FE 建立索引，它是继承层次上的最优检索方案。目前，一些商用 OODB 都在用这种索引策略，像 DB4O 等。但是该策略增加了维护开销，像添加或删除一个叶子类的对象，需要更新继承层次上全部祖先类索引。

如何平衡 OODB 索引的检索性能、维护开销、实现复杂度已经成为一个影响 OODB 性能的关键问题。同时，关于选择合适索引集来最小化检索及维护开销的索引选择问题(ISP)已经提出了很多解决方案，也提出一些针对 OODB 进行索引选择评估[5]的方案。但是到目前为止都很少有针对继承层次的索引配置方法。本文给出一种解决方案。

1　基于数据库统计数据的初级索引

先通过数据库统计数据建立继承层次的初级索引(Primary index 以下简称 PI)，用来提高继承层次检索性能，同时也作为索引配置的基础，PI 可以逐步退化成 CH-Tree。

1.1 初级索引基本原则

同 CD 方法一样，这里对继承层次分组建立小索引。每个小索引应用在一个类及其下级类(像子类)上。依靠该类及其子类的存取概率、实例数来建立。初级索引 i (简称 PI(i))是建立在以类 Ci 为根的继承层次上的 CH-tree 索引。

PI(i)中平均查询对象率：

$$QR(i)=P_i-\sum_{\substack{C_j\in 类C_i的子类\\ C_j\notin 类C_k及其子类}} P_j\times\frac{S_i-S_j}{S_i}$$

QR(i)是建立在类 C_i 上的 PI(i)的权值。P_i (或 P_j)是类 C_i (或 C_j)的存取概率。类 C_k 是以类 C_i 为根的继承层次上已经建立了 PI(k)的类。S_i (或 S_j)是以类 C_i (或 C_j)为根的继承层次上的所有实例数，不包含 S_k。

如果 QR(i)=P_i，说明 PI(i)中平均查询对象数为 S_i，使用 PI(i)检索对象时不用扫描多余的数据，然而，这种情况只可能出现在叶子类上。因此，我们需要考虑当查询目标针对类 C_i 及其子类时，扫描了多少多余的数据，可以通过 QR(i)来体现。将所有类的 QR(i)排序，在拥有最大权值的类上建立 PI(i)，如果该类的权值为正，并且它的祖先没建立过索引，则建立 PI(i)会提高检索性能。当拥有最大权值的类为根类或最大权值非正时停止建立。最后在根类上建立 PI(root)。

1.2 初级索引算法

/*输入：继承层次类：C1，C2，C3…Cn (C1 是根类)，每个类的存取概率、实例数。输出：创建 PI*/

while(类 Ci 的祖先没有建立 PI and i≤n)

计算 QR(i); end while

排序 QR(i)；选择 QR 最大的类;

while(当前类不是根类 and QR(i) > 0)

建立 PI(i)；调整剩余 QR；重新排序; end while

建立 PI(root);

1.3 初级索引举例

这里用一个例子来阐述 PI 的建立。如图 1 所示，给出一个 10 个类的继承层次，标有每个类的存取概率和实例数。

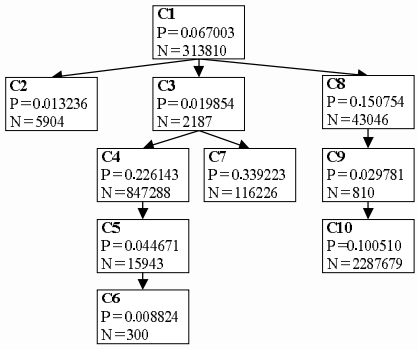

图 1　以 C1 为根的继承层次索引结构

计算所有类的 QR(i)，并排序，如表 1 所示。

表 1　第一次排序后的 QR

类	C_7	C_4	C_8	C_{10}	C_5
QR	0.3392	0.1735	0.1483	0.1005	0.036
类	C_9	C_2	C_6	C_1	C_3
QR	0.0297	0.0132	0.0088	-0.617	-0.3592

第一次排序后，在类 C_7 上建立 PI(7)。然后，调整剩余类的 QR(例：C_1,C_3)，并重新排序，如表 2 所示。

表 2　建立 PI(7)后的 QR 排序

类	C_7*	C_4	C_8	C_{10}	C_5
QR	0.3392	0.1735	0.1483	0.1005	0.036
类	C_9	C_2	C_6	C_1	C_3
QR	0.0297	0.0132	0.0088	-0.2814	-0.0334

第二次排序后，在类 C_4 上建立 PI(4)。继续调整，并重新排序，如表 3 所示。

表 3　排序建立 PI(4)后的 QR 排序

类	C_7*	C_4*	C_8	C_{10}	C_9
QR	0.3392	0.1735	0.1483	0.1005	0.0297
类	C_3	C_2	C_1		
QR	0.0199	0.0132	-0.0023		

第三次排序后，在类 C_8 上建立 PI(8)。继续调整，并重新排序，如表 4 所示。

表 4　建立 PI(8)后的 QR 排序

类	C_7*	C_4*	C_8*	C_1	C_3
QR	0.3392	0.1735	0.1483	0.0343	0.0199
类	C_2				
QR	0.0132				

第四次排序后，C_1 的 QR 最大，并且为根类，因此，建立 PI(1)，终止算法。最终，建立了 PI(7)，PI(4)，PI(8)，PI(1)。

2　继承层次索引配置

PI 在检索性能上仍然无法同 FE 索引策略相比，但维护开销要比 FE 索引策略低很多。如果我们仍要提高检索性能，就需要对不是 PI(j)根类的剩余类建立 FE 索引。FE(C_i)是为类 C_i 的 FE 建立的 B+-Tree 索引。如果对全部剩余类建立 FE(C_i)，索引配置将进化成 FE 索引策略。

2.1　索引配置算法

/*输入：继承层次类：C_1,C_2,...C_n(C_1 是根类)，每个类的存取概率，单值查询率，平均查询范围，数据库参数，索引参数

输出：(索引配置，RC 如式(4),SC 如式(6),AUC 如式(9))

创建 PI;

while(Ci ∈ 可创建 FE 索引的全部类)

计算 PI 相比 FE 索引策略针对类 C_i 的检索性能损失率;

$$RPLR_{C_i}=((\frac{H_j}{H_{C_i}}-1)\times 单值查询率+(\frac{LP_j}{LP_{C_i}}-1)\times(1-单值查询率))\times P_{C_i}\times 100$$

end while

//以 RPLRCi 为阀值配置 FE 索引

while(Cj ∈ 可创建 FE 索引的全部类)

while(RPLRCj ≥ RPLRCi) 创建 FE(Cj); end while

计算并输出以 RPLRCi 为阀值得到的 RC,SC,AUC;

清除当前创建的所有 FE 索引; end while

//以 QR(i)为阀值配置 PI

while(PI 不空)

计算并输出以最大 QR(i)为阀值得到的 RC,SC,AUC;

清除 PI(i); end while

2.2　索引配置举例

这里对图 1 继承层次进行配置。单值查询率 95%，平均查询范围 10%,总的不同关键值数 300，开销模型参数如表 6。表 5 给出可用来配置 PI 索引的根类及 FE 索引的类、阀值。

表 5　图 1 继承层次的可配置类、阀值

类	C_7	C_4	C_8	C_1		
QR	0.3392	0.1735	0.1483	0.0343		
类	C_2	C_5	C_6	C_3	C_9	C_{10}
RPLR	3.8715	6.0864	13.611	0	0	0

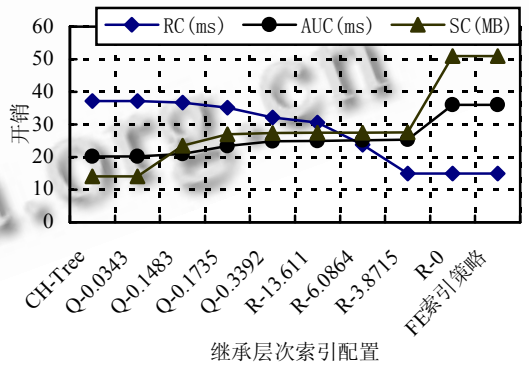

图 2　继承层次各种索引配置开销

如图 2 所示，当以 R-0(阀值类型 RPLR，阀值 0)为阀值时，我们的索引配置进化成 FE 索引策略；当以 Q-0.3392(阀值类型 QR，阀值 0.3392)为阀值时，退化成 PI；当以 Q-0.0343 为阀值时，退化成 CH-Tree；当阀值为 R-3.8715 时索引配置最优，RC 接近 FE 索引策略，SC,AUC 较 FE 索引策略都有大幅降低。最优索引配置建立七个索引文件。

①以 C_7,C_4,C_8,C_1 为根类分别建立 PI(7)，PI(4)，

PI(8)，PI(1)。

②在 C_2,C_5,C_6 的 FE 上分别建立 FE(C2)，FE(C5)，FE(C6)。

3 开销模型

我们通过开销模型来评估一个索引配置的检索、存储、更新开销。首先给出模型用到的一些参数，然后由它们得到开销模型，最后通过实验来验证开销模型。

3.1 参数

3.1.1 数据库、索引参数

①数据库参数

N_{Ci}：类 C_i 的实例数。DIS_{Ci}：类 C_i 中不同关键值数。Nj/C_i：PI(j)/FE(C_i)中实例数。$D_{j/Ci}$：PI(j)/FE(C_i)中不同关键值数。$K_{j/Ci}$：PI(j)/FE(C_i)中一个关键值包含的平均元素个数，$K_{j/Ci}= N_{j/Ci} / D_{j/Ci}$。AST：平均寻道时间。RPT：读取一个索引页时间。P_{Ci}：类 C_i 的存取概率。FN：存取的索引文件数。NC_j：PI(j)中一个索引记录包含类数，$NC_j = \sum_{C_i \in \text{PI(j)中包含的所有类}} DIS_{C_i} / D_j$ 。

② 索引参数

P：索引页的大小。f：内部节点的平均扇出数。kl：索引属性值的平均长度。

3.1.2 PI(j)/FE(Ci)索引参数

①$XL_{j/Ci}$ — PI(j)/FE(C_i)中叶节点记录的平均长度(PI(j)的叶节点记录结构如图 3 所示)。

record header	key value	#classes	(CID,offset,#OIDs)...(CID,offset,#OIDs)	OID_1...OID_N

图 3 根据 B+-Tree 修改后的 CH-Tree 的叶节点记录结构

XL_j = header_length+kl+ (sizeof (CLASSID) +sizeof (offset) +sizeof (number_of_OIDs)) × NC_j +sizeof (OID) × K_j

XL_{Ci} = header_length+kl+sizeof (OID) × K_{Ci}

②$LP_{j/Ci}$ — PI(j)/FE(C_i)中包含的叶节点记录页的个数。

③$OP_{j/Ci}$ — PI(j)/FE(C_i)中包含溢出页的个数。

如果 XL≤P　$LP = \lceil (D \times XL)/P \rceil$

如果 XL > P　$LP = D$，$LP + OP = D \times \lceil XL/P \rceil$

④$H_{j/Ci}$ — PI(j)/FE(C_i)的高度(包含叶节点层),H = 多项式 $LP + \lceil LP/f \rceil + \lceil \lceil LP/f \rceil / f \rceil + \ldots + 1$ 中项的个数。

3.2 检索开销模型

PI(j)/FE(C_i)的检索开销如式(3)，其中单值查询开销如式(1)，范围查询开销如式(2)。继承层次总检索开销如式(4)。

$$RC^{\text{Single}}_{j/C_i} = \begin{cases} H+1 & (XL \le P) \\ H + \lceil XL/P \rceil & (XL > P) \end{cases} \quad (1)$$

$$RC^{\text{Range}}_{j/C_i} = \begin{cases} H + \lceil \text{queryRange} \times LP \rceil & (XL \le P) \\ H + \lceil \text{queryRange} \times (LP+OP) \rceil & (XL > P) \end{cases} \quad (2)$$

$$Cost_{j/C_i} = RC^{\text{Single}}_{j/C_i} \times \text{单值查询率} + RC^{\text{Range}}_{j/C_i} \times (1 - \text{单值查询率}) \quad (3)$$

$$RC = \sum_{C_i \in \text{全部类}} Cost_{j/C_i} \times P_{C_i} \times RPT \quad (4)$$

3.3 存储开销模型

PI(j)/FE(C_i)的存储开销如式(5)，总存储开销如式(6)。

$$SC_{j/C_i} = \begin{cases} LP + (\lceil LP/f \rceil + \lceil \lceil LP/f \rceil / f \rceil + \cdots + 1) & (XL \le P) \\ LP + OP + (\lceil LP/f \rceil + \lceil \lceil LP/f \rceil / f \rceil + \cdots + 1) & (XL > P) \end{cases} \quad (5)$$

$$SC = \sum_{\text{索引配置中全部索引}} SC_{j/C_i} \quad (6)$$

3.4 更新开销模型

当添加(或删除)一个对象时,需要更新索引。为了简化分析，我们不考虑非叶节点的更新。PI(j)/FE(C_i)的更新开销如式(7)，类 C_i 的更新开销如式(8)，平均更新开销如式(9)。

$$UC^{\text{Insert/Delete}}_{j/C_i} = \begin{cases} H+1+1 & (XL \le P) \\ H + \lceil XL/P \rceil + 1 & (XL > P) \end{cases} \quad (7)$$

$$UC_{C_i} = \sum_{\text{包含类}C_i\text{的全部索引文件}} UC^{\text{Insert/Delete}} \times RPT + FN \times AST \quad (8)$$

$$AUC = \sum_{C_i \in \text{全部类}} UC_{C_i} \times \text{类}C_i\text{的更新率} \quad (9)$$

4 实验

下面给出在继承层次上的 RC 及 AUC 实验结果。用数据库统计数据来反映 OODB 真实的使用情况。我们比较常用继承层次索引方法：CH-Tree，CD 方法，FE 索引策略。实验证明文中给出的评估模型结果接近真实的实验结果，这种根据数据库使用情况的索引配置方法可以为OODB提供性能最优的继承层次索引方案。

4.1 实验环境

这里采用磁盘存取时间作为评估参数。所有试验在三星 MP0804H 磁盘上进行。磁盘的外部传输速度

100MB/S，平均寻道时间 12ms，读取一个 4KB 的索引页用时 0.03906ms。随机产生类数量为 15 的继承层次，单值查询率，平均查询范围，每个类的实例数，存取概率，关键值分布随机分配。实验参数如表 6 所示。

表 6 实验参数值

P	*f*	*kl*	header_length
4096	255	4	16
offset	OID	number_of_OIDs	CLASSID
2	4	2	4

4.2 实验结果

如图 3，最优索引配置的检索性能好于 CH-Tree 及 CD 方法，略低于 FE 索引策略。因为 CH-Tree 扫描大量多余数据，检索性能最差。CD 方法由于读取多个索引文件，需要大量寻道时间。更新方面，如图 4，由于 FE 索引策略要更新继承层次上全部祖先类索引，更新开销最高。在检索性能相差不大的情况下，最优索引配置的更新开销远低于前者。

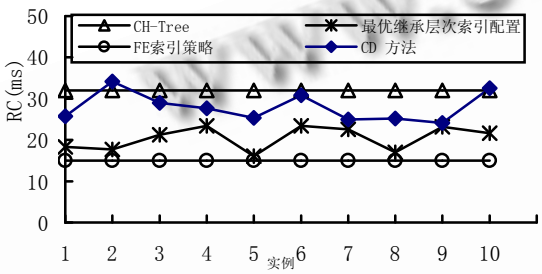

图 3 15 个类的继承层次检索开销

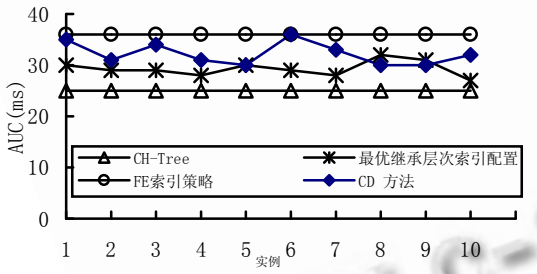

图 4 15 个类的继承层次更新开销

5 结语

该文给出一种 OODB 继承层次的索引配置方法，它可以用于 OODB 的查询优化及物理数据库设计。尽管文中的评估模型经过实验验证，接近真实结果，但精确度不高，而且评估采用探索方法，评估开销很高，未来重点解决以上问题。

参考文献

1 kim W, Kim KC, Dale A. Object-oriented Concepts Database and Applications. Boston: Addison-Wesley, 1989: 371－394.

2 Huang YF, Chen JM. The study of indexing techniques on object oriented database. Information Sciences, 2000,130(1):109－131.

3 Ahn JH, Song HJ, Kim HJ. Index set: A practical indexing scheme for object database systems. Data&Knowledge Engineering, 2000, 33(1):199－217.

4 kim W. Introduction to Object-Oriented Database. Cambridge: MIT Press, 1990:89－125.

5 Cho WS, Hong KH, Loh WK. Estimating nested selectivity in object-oriented and object-relational databases. Information and software Technology, 2007,49(1):806－816.