文章编号: 1001 - 9081 (2006) 06Z - 0253 - 02

引入角度信息改进的 NB-Tree多维索引方法

徐 焕[1],林坤辉[1],周昌乐[2]
(1. 厦门大学 软件学院, 福建 厦门 361005; 2. 厦门大学 信息科学与技术学院,福建 厦门 361005)
(jploxh@sina.com)

摘 要: NB-Tree的缺点是只存储了对象的欧氏距离,忽略了具有相同欧氏距离的对象的位置信息,在范围检索中,不可避免会有欧氏距离相近而多维矢量并不相关的对象被作为检索对象读入内存,进行二次过滤,增大了不必要的 I/O操作和距离计算。针对 NB-Tree的不足,引入多维矢量的空间位置信息:与矢量 $[1, 1, \cdots, 1, 1_n]$ 的偏移角,提出了一种新的索引结构: New-NB-Tree,通过较少的计算,进一步加强过滤。

关键词:多维索引;欧氏距离;偏移角
中图分类号: TP311.134.3 **文献标识码**: A

0 引言

现在,高维索引技术和算法的已经进行了很多的研究,取得了很大的进步,但也存在一些问题尚待解决,"维数危机"依然是困扰多维索引性能的一个重要问题,很多索引结构在维数较低 (≤10)表现出良好的性能,但是如果维数多达 100维,那么传统的索引结构的性能降急剧下降,甚至不如顺序扫描。另外目前的很多索引结构都是适用于维数一定的数据,但是应用领域的很多数据并不是固定维的,所以,设计一种性能优良,简单有效并且和维数无关的索引结构成为研究的重点。文献[3]提出了一种新的索引结构 NB-tree,这种访问方法就避开了复杂的结构,利用 B^+-Tree来存储对象的欧氏距离,该算法有较好的适应性,且易于在现有的 DBMS上实现,实验证明了索引结构的有效性。然而 NB-tree的缺点是只存储了对象的欧氏距离,忽略了具有相同欧氏距离的对象的位置信息。不可避免会有欧氏距离相近而多维矢量并不相关的对象被作为检索对象读入内存,进行二次过滤,增大了不必要的 I/O操作和距离计算。通过仔细研究原来的算法,我们引入多维矢量的空间位置信息——偏移角,提出了一种新的索引结构: New-NB-Tree,通过加强过滤能力,进一步减少访问对象的数目,进而减少 I/O操作。实验表明, New-NB-Tree能提高检索效率。

1 NB-Tree的设计思想

首先计算对象特征向量 (N维)的欧氏距离,将 N维向量映射为 1维值,然后通过 B^+-Tree来组织和存储这些距离。B^+-Tree的最大优点是可以快速的进行范围查询且叶子上的数值是顺序连接的。NB-Tree支持点查询,范围查询和 KNN最邻近查询。NB-Tree的检索过程如下:

点查询:首先计算查询对象 Q的欧氏距离 R, 利用 NB-Tree初步确定欧氏距离为 R的查询结果,然后计算查询结果中的对象和 Q的距离,若有结果为 0,则说明 Q在 NB-Tree中,返回结果。

范围查询:给出查询对象 Q和查询半径 r,首先计算查询对象的欧氏距离 R, 利用 NB-Tree初步确定欧氏距离在 $[\max(0, R - r), (R + r)]$范围内的对象,然后对这些对象按着定义的相似度公式进行精确的计算,返回结果集。

NB-Tree的检索过程如图 1所示。

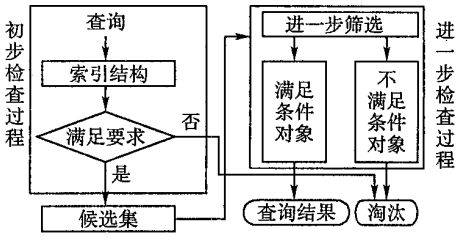

图 1 NB-Tree的检索过程

NB-Tree叶子节点的节点结构如图 2所示。

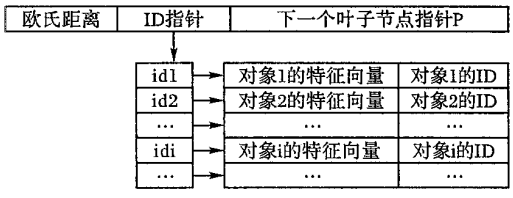

图 2 NB-Tree的叶子节点结构

优点:该算法简单有效,很容易实现;避免了维数危机,随着维数的上升,性能变化缓慢; B^+-Tree的优良性能,使得该算法也有很好的性能表现。

不足:由上面的检索过程图可以了解到, NB-Tree首次检索到的候选结果集里有很多脏数据,有很多是欧氏距离相近但对象的特征矢量并不相关的,也要被读入内存进行二次筛选,增加了不必要的 I/O代价和距离计算。而 I/O操作是影响索引性能的很重要的指标。

2 对 NB-Tree算法的改进

2.1 New-NB-Tree的设计思想

针对 NB-Tree的以上特点,我们在 NB-Tree的叶子节点上加入和维数无关的偏移角信息,设计了一种新的索引结构 New-NB-Tree。通过少量的计算,进一步加强过滤,最大程度的减少候选数据里的脏数据。通过改进的索引结构,保留了

收稿日期: 2005 - 10 - 20;**修订日期**: 2006 - 01 - 10 **基金项目**:厦门大学 985二期信息创新平台资助项目
作者简介:徐焕 (1975 -),女,河南禹州人,硕士研究生,主要研究方向:网络多媒体、数据库应用; 林坤辉 (1961 -),男,福建惠安人,副教授,硕士,主要研究方向:网络多媒体、数据库应用; 周昌乐 (1959 -),男,江苏太仓人,教授,博士生导师,博士,主要研究方向:人工智能.

NB-Tree的优点,有效改进了原来算法的不足,提高了检索效率。设计思想如下:

在 N 维空间中,选取向量 $[1, 1, \cdots 1, 1_n]$ (n是矢量维数)作为基向量,计算每个对象的特征向量与该向量的夹角,并把该夹角信息写入索引的叶子节点中。该夹角信息说明了对象的特征向量在空间中的位置。

改进后的节点结构如图 3所示。

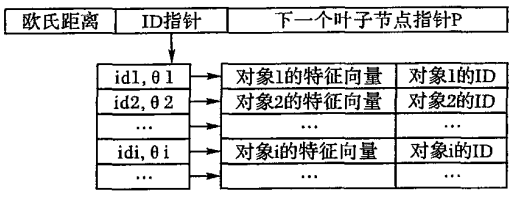

图 3 改进后的节点结构

2.2 两种算法检索范围的比较

以二维空间范围查询为例,查询对象 q,查询半径为 r,图 4(a)的阴影部分为 NB-Tree的检索结果。作以原点为起点,以 q为圆心,以 r为半径的圆的两条切线,则得到图 4(b)阴影部分是 New-NB-Tree的检索结果。图 4说明 New-NB-Tree极大的缩小了检索范围。

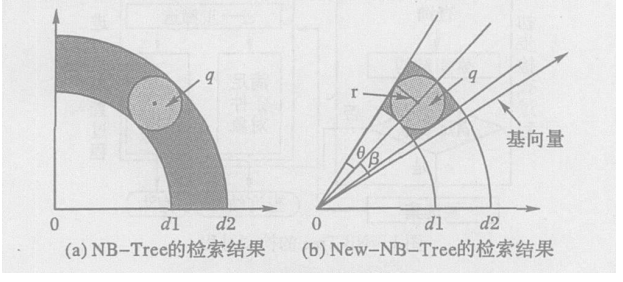

图 4 两种方法的检索范围

2.3 New-NB-Tree的实现原理

定义 1 在 n维空间中,选取向量 $[1, 1, \cdots, 1, 1_n]$作为基向量,那么 n维向量到该基向量的夹角称为该向量的偏移角。

由图 2所示,查询对象为 q,查询半径为 r,设所做的切线和查询对象 q之间的夹角为 θ,查询对象和基向量的夹角 β,首先需要计算 θ以及 β的值,则查询范围内的对象,其偏移角应满足区间 $[\beta - \theta, \beta + \theta]$。$\theta$和 β的计算方法如下:

$$\sin\theta = r / |q| \approx \theta \quad (1)$$

成立的条件是 $r \leq |q|$, r是查询半径, $|q|$是向量 q的欧氏距离。当 $r > |q|$时,夹角信息不起任何作用。

向量 $q = [q_1, q_2, \cdots, q_i, \cdots, q_n]$与向量 $p = [p_1, p_2, p_3, \cdots, p_n]$的夹角为 α,则:

$$\cos\alpha = \frac{p \cdot q}{|p|\,|q|} = \frac{\sum p_i q_i}{|p|\,|q|}$$,其中 $|p|$和 $|q|$分别是向量 p和 q的欧氏距离。

所以向量 $q = [q_1, q_2, \cdots, q_i, \cdots, q_n]$与向量 $[1, 1, \cdots 1, 1_n]$的夹角为 β,则:

$$\cos\beta = \frac{\sum q_i}{\sqrt{N}\,|q|} \quad (2)$$

其中 N为特征向量的维数, $|q|$是向量 q的欧氏距离。因为 β的最大值为 π,而 $\cos\beta$在 $[0, \pi]$的区间内是单调的,则:

$$\beta = \arccos\frac{\sum q_i}{\sqrt{N}\,|q|}$$

考虑以下一种特殊情况,当查询对象的偏移角 β小于等于 θ,则基向量穿过查询区域,如图 5所示。

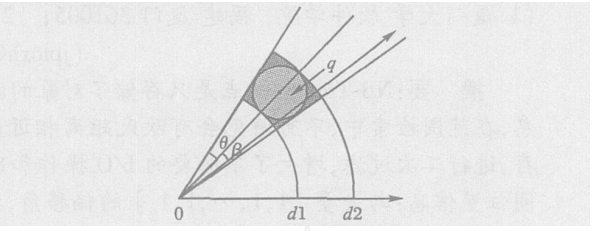

图 5 基向量穿过查询区域

此时查询对象的夹角和基向量的夹角最小值为零,检索结果的偏移角应在区间 $[0, \beta + \theta]$内。

综上所述,查询对象的偏移角应满足 $[\max(0, \beta - 1), \beta + \theta]$。

2.4 New-NB-Tree检索过程

改进后的索引结构,由于在叶子节点里引入了偏移角信息,所以相应的改进了检索算法,进一步加强了过滤,New-NB-Tree的范围查询的检索过程如下:

1) 查询对象 q和查询半径 r,计算 q的欧氏距离 $|q|$,计算切线和查询对象 q之间的夹角为 θ以及查询对象和基向量的夹角 β;

2) 遍历索引树,返回欧氏距离在 $[\max(0, |q| - r), |q| + r]$的所有对象。称为候选集;

3) 读取候选集里对象的偏移角,验证其是否在范围 $[\max(0, \beta - \theta), \beta + \theta]$;是则返回,不是则舍掉;

4) 对于 3) 返回的对象进行精确匹配,根据相似度公式,计算每个对象与查询对象 q的精确距离 L,返回所有 $L \leq r$的对象,提交结果集。

3 实验结果及性能分析

3.1 测试环境

本文中所做的所有测试使用的软硬件环境如下:

1)硬件环境: P4联想兼容机, 3.0GHz双 CPU, 1G内存, 160GB硬盘。

2)操作系统平台: Microsoft Windows XP。

3)编程环境: Microsoft VC++6.0编译器。

3.2 测试方案

选用 http://www.stbag.com/~berchtol/papers/founew.nonniert.gz上提供的特征数据集。该数据集共有 10万个 32维矢量,曾用作 X-tree的测试数据集,它的特点是特征向量是图像特征的付立叶系数表示,因此可以将特征向量的后若干维简单的去掉,仍然能够得到另一个有效的特征向量。该特征向量集的最大维数是 32,实验分为四组,选取不同的查询半径,在该数据集上随机选出 10个点,对每个查询半径分别检索 10次。由于所测试的索引结构都是基于外存的索引结构,因此内存和外存之间的 I/O操作是影响查询性能的最主要的因素。而每次查询执行之后,查找路径上的节点都被读进了内存中,这样下次查询时,就减少了外存到内存的读操作,这样得到的查询响应时间就是所谓的热结果,但是这种热

(下转第 263页)

式,实验中选择的是 lite模式 (关于 Minigui的更多信息,请查阅其手册)。设置环境变量 PATH如下:

PATH = /usr/godson-linux/bin: $PATH

首先编译 Minigui的函数库 libminigui,进入 libminigui-1.3.0目录进行配置:

./configure --host = i686-pc-linux-gnu --enable-lite

配置完成后输入命令 make install,系统将开始编译和生成 minigui的动态和静态库文件,并将生成的库文件安装到 /usr/local/lib目录下,相应的头文件将会安装到 /usr/local/include/minigui目录下,然后进入 minigui-res-1.3.0目录安装资源文件,至此,Minigui图形开发环境就建立起来了,利用该环境,编译 mde-1.3.0中的演示程序,编译完成后,该演示程序在龙芯"gs32i"系统平台上成功运行起来。

6 结语

在嵌入式软件系统设计中,交叉开发环境的设计至关重要,本文通过对嵌入式 Linux交叉开发平台技术的探讨和研究,利用开源软件构建了一个嵌入式 Linux的交叉开发环境,并成功地在龙芯"gs32i"系统上实现了一个嵌入式 Linux运行平台,在此基础上,可以构建更丰富的应用,目前已经完成 web服务器、SSH服务程序、Minigui库等。

当然这个交叉开发平台还有很多不足,其编程、编译、链接库、调试工具等相对独立,没有一个统一的操作窗口;目前还未在该系统上实现图形窗口,只能在虚拟终端上进行操作,给应用开发人员的使用带来一些不便。这些问题将会在未来的研究中逐步解决,使交叉开发环境更加完善,并在应用领域得到更大的扩展。

参考文献:

[1] WIN-BIN SEE, PAO-ANN HSIUNG, TRONG-YEN LEE, *et al*. Software platform for embedded software development [A]. Real-Time and Embedded Computing Systems and Applications, 9th International Conference, RTCSA 2003 [C]. Berlin, Germany: Springer-Verlag 2004. 545-57.

[2] LEHRBAUM J, WEINBERG W. Cross-development of embedded Linux programs on windows-based host computers [J]. Elektronik, 2003, 52 (16): 45-9.

[3] SEUNGWOO SON, CHAEDEOK LIM, HEUNG-NAM KIM. Debugging protocol for remote cross development environment [A]. Proceedings of the Seventh International Conference on Real-Time Systems and Applications (RTCSA'00) [C]. 2000.

[4] 陈滔,李志刚. 刘执远,等. CC-Linux集成开发环境的设计与实现 [J]. 计算机工程与应用, 2001, 37 (23).

[5] 刘执远. 嵌入式 Linux集成开发环境 [M]. 西安:西北工业大学. 2001.

[6] 何家胜. 龙芯开发板的软件构建 [Z]. 2004.

[7] Busybox: The Swiss army knife of embedded linux [EB/OL]. http://busybox.net/about.html, 2005-10.

(上接第 254页)

结果反应不出基于外存的索引结构的性能特点,因此我们只使用冷结果作为评价算法性能的依据:每次查询进行之后,都有相应的清理内存的动作。

3.3 结果分析

表 1 实验结果分析

查询半径 0.04		查询半径 0.06		查询半径 0.08		查询半径 0.1	
旧算法	新算法	旧算法	新算法	旧算法	新算法	旧算法	新算法
2216	1636	2560	1960	2890	2404	4563	4097
1654	1136	1962	1392	2234	1820	3610	2931
1980	1423	2234	1748	2569	2064	3621	3190
2631	1767	3112	2226	3461	2798	5698	4846
966	665	1268	989	1657	1488	2645	2352
1589	1064	2311	1720	2886	2530	2641	2234
2136	1662	2594	2003	3152	2546	3845	3234
1589	934	1898	1320	2123	1654	2964	2346
2114	1463	2480	2084	2698	2386	3156	2691
2365	1538	3120	2332	3561	2733	3679	3160

在磁盘进行一次 I/O的时间里,一台典型的机器能执行百万条的指令[5],因此 I/O操作所用的时间是算法所用时间的近似值,是影响索引算法效率的主要因素。在实验中以过滤后的对象数量作为 I/O次数比较的量度。这里采用的检索效率提高程度的计算公式为:

$$\left(\sum_{i=1}^{N}\frac{oldnum_i - newnum}{oldnum_i}\right)/N$$

其中,$oldnum_i$表示第 i次旧算法过滤后的返回的数据量,$newnum_i$表示第 i次新算法过滤后的返回的数据量,N表示检索的次数。按着这种计算方法得到 4种情况的检索效率的提高率分别为:30.8%, 21%, 16.2%, 14.2%. 从实验结果可以看出,当查询半径较小时,索引结构的效率提高较大。

4 结语

针对 NB-tree只存储对象的欧氏距离而忽略位置信息的缺点,提出了 New-NB-tree索引结构。New-NB-tree主要是修改了 NB-tree叶子节点的结构,增加了独立于维数的角度信息,通过较小的存储空间和代价不大的计算,在完成原有的过滤功能外,进行了第二级过滤,减少了 I/O次数,提高了算法的效率。实验数据证明了算法的有效性。

参考文献:

[1] GAEDE V, GUNTHER O. Multidimensional Access Methods [J]. ACM Computing Surveys, 1998, 30 (2), 170-231.

[2] WEBER R, SCHEK H-J, BLOT S. A Quantitative Analysis and Performance study for Similarity-search Methods in High-dimensional Spaces [A]. In 24th Int Conference on Very Large Databases (VLDB) [C]. New York, 1998. 194-295.

[3] FONSECA MJ, JORGE J. Indexing High-Dimensional Data for Content-Based Retrieval in Large Databases [A]. Proceedings of the 8th International Conference on Database Systems for Advanced Applications [C]. Kyoto, Japan, 2003. 267-274.

[4] CHA GH, ZHU XM, PETKOVIE D. An Efficient Indexing Method for Nearest Neighbor Searches in High-Dimensional Image Databases [J]. IEEE, 2002, 4 (1): 76-87.

[5] FILHO RFS, TRAINA A, TRAINA C JR, *et al*. Similarity Search without Tears: the OMNI-Family of All-Purpose Access Methods [A]. 17th International Conference on Data Engineering [C]. Heidelberg, Germany, 2001. 623-630.

[6] LI C, CHANG E, GARCIA-MOLINA H, *et al* Clustering for Approximate Similarity Search in High-dimensional Spaces [J]. IEEE Transactions on Knowledge and Data Engineering, 2002, 14 (4): 792-808.