数据仓库系统中层次式 Cube 存储结构*

高 宏+, 李建中, 李金宝

(哈尔滨工业大学 计算机科学与技术学院,黑龙江 哈尔滨 150001)
(黑龙江大学 计算机科学与技术学院,黑龙江 哈尔滨 150086)

Hierarchical Cube Storage Structure for Data Warehouses

GAO Hong+, LI Jian-Zhong, LI Jin-Bao

(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)
(School of Computer Science and Technology, Heilongjiang University, Harbin 150086, China)
+ Corresponding author: Phn: 86-451-6415827, Fax: 86-451-6415827, E-mail: gaohong@mail.banner.com.cn
Received 2002-11-25; Accepted 2003-03-04

Gao H, Li JZ, Li JB. Hierarchical cube storage structure for data warehouses. *Journal of Software*, 2003,14(7):1258~1266.
http://www.jos.org.cn/1000-9825/14/1258.htm

Abstract: Range query is a very important operation to support On-Line Analytical Processing (OLAP) in data warehouses. Although several cube storage structures for range sum queries and dynamic updates have been introduced recently. However, the complexities of both space and time are too higher to realistic. To solve this problem, a hierarchical data cube (HDC) and corresponding algorithms are provided in this paper. Both of the range query and update costs of HDC are $O(\log^d n)$, and the overall cost is $O((\log n)^{2d})$ (under the C_qC_u model) or $O(K(\log n)^d)$ (under the $C_qn_q+C_un_u$ model). The analytical and experimental results show that the costs of HDC's range queries, dynamic updates, storage space and the overall performance of HDC are superior to other cubage storage structures.
Key words: data warehouse; on-line analytical processing; cube; range query

摘 要: 区域查询是数据仓库上支持联机分析处理(on-line analytical processing,简称 OLAP)的重要操作.近几年,人们提出了一些支持区域查询和数据更新的 Cube 存储结构.然而这些存储结构的空间复杂性和时间复杂性都很高,难以在实际中使用.为此,提出了一种层次式 Cube 存储结构 HDC(hierarchical data cube)及其上的相关算法.HDC 上区域查询的代价和数据更新代价均为 $O(\log^d n)$,综合性能为 $O((\log n)^{2d})$(使用 C_qC_u 模型)或 $O(K(\log n)^d)$(使用 $C_qn_q+C_un_u$ 模型).理论分析与实验表明,HDC 的区域查询代价、数据更新代价、空间代价以及综合性能都优于目前所有的 Cube 存储结构.

---

* Supported by the National Natural Science Foundation of China under Grant No.60273082 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2001AA415410 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.G1999032704 (国家重点基础研究发展规划(973)); the Natural Science Foundation of Heilongjiang Province of China under Grant No.F0208 (黑龙江省自然科学基金)

第一作者简介: 高宏(1966－),女,黑龙江哈尔滨人,博士,副教授,主要研究领域为数据库,数据仓库.

关键词: 数据仓库;联机分析处理;Cube;区域查询
中图法分类号: TP311 **文献标识码:** A

区域查询是数据仓库上进行OLAP(on-line analytical processing)分析的重要操作.它在Cube[1]的指定子空间范围内,对度量属性执行聚集操作,如sum.设 $S=[l_1,h_1]\times[l_2,h_2]\times\ldots\times[l_d,h_d]$ 是 d-维Cube C 的一个 d-维区域,则 S 上的区域查询可以形式化地表示为 $f(S)=f(\{C(x_1,x_2,\ldots,x_d)|\forall(x_1,x_2,\ldots,x_d)\in S\})$, f 是一个聚集函数.由于Cube存储结构应该同时支持高效率的数据查询和数据更新,评价Cube存储结构的性能不能单纯地考察区域查询代价或数据更新代价,需要综合考虑两者的性能.文献[2]给出了一个基于乘积的Cube存储结构评价模型 $C_q\times C_u$, C_q 是数据查询的代价, C_u 是数据更新的代价.文献[3]给出了一个比较准确的Cube存储结构的评价模型 $C_qn_q+C_un_u$,其中 n_q 是数据查询次数, n_u 是数据更新次数.

近几年,很多研究者在Cube存储结构方面开展了研究工作,努力提高Cube存储结构的综合性能(即减少 $C_q\times C_u$ 或 $C_qn_q+C_un_u$),取得了一些成果[2~8].文献[4]提出了一种称为Prefix Sum(简称PS)的存储方法,其查询代价为 $O(1)$,更新代价为 $O(n^d)$,综合性能为 $O(n^d)$(使用评价模型 $C_q\times C_u$)或 $O(Kn^d)$(使用评价模型 $C_qn_q+C_un_u$).其他研究工作都是在文献[4]的基础上,引入一些辅助结构,尽量降低更新代价.然而,这些工作仍存在一定的局限性.原因在于,尽管采用了辅助结构,使得更新代价从某种程度上得到减少,但却以牺牲过多的查询代价为前提,其综合性能虽然高于文献[4],但是仍然较低.随着数据仓库中海量数据(高于 10^{12} 字节)的大量涌现,在海量数据仓库上有效地进行区域查询与数据更新成为亟待解决的问题.目前已有的各种方法不能很好地处理海量数据上的区域查询与数据更新问题.因此,本文提出了一种新的基于层次划分的Cube组织结构HDC(hierarchical data cube).HDC上区域查询的代价为 $O(\log^d n)$,数据更新代价为 $O(\log^d n)$,综合性能为 $O((\log^d n)^2)$(使用 C_qC_u 模型)或 $O(K(\log^d n))$(使用 $C_qn_q+C_un_u$ 模型).理论分析与实验表明,HDC的区域查询代价、数据更新代价和综合性能都优于目前所有的Cube存储结构.

1 区域查询模型与相关技术

假定 C 为一个 d-维的Cube, D_i 为第 i 维的值域.我们用一个大小为 $\prod_{i=1}^{d} n_i$ 的 d-维数组来存储 C.在下面的讨论中,我们将把 C 视为一个 d-维数组.为简单起见且不失一般性,我们假定 $|D_1|=|D_2|=\ldots=|D_d|=n$.

文献[4]利用Prefix Sum技术提出了Prefix Cube存储结构(简称PC).PC是与 d-维Cube C 大小相同的 d-维数组, $PC(x_1,\ldots,x_d)=f(\{C(y_1,\ldots,y_d)|\forall 1\le i\le d, 0\le y_i\le x_i\})$, f 是聚集函数.图1给出了一个2-维Cube C及其对应的PC结构, f=sum.由图1可知, $PC(1,2)=C(0,0)+C(0,1)+C(0,2)+C(1,0)+C(1,1)+C(1,2)=21$,是 C 中以 $C(0,0)$ 和 $C(1,2)$ 为顶点的矩形中所有点的和.利用PC,可以通过 2^d 个单元的访问完成任意区域查询.例如,我们可以如下计算以 $C(1,1)$ 和 $C(4,4)$ 为顶点的矩形区域Area_E的聚集之和:sum($Area_E$)= $PC(4,4)-PC(4,1)-PC(1,4)+PC(1,1)$.

Index	0	1	2	3	4	5	6	7
0	3	5	1	2	2	4	6	3
1	7	3	2	6	8	7	1	2
2	2	4	2	3	3	3	4	5
3	3	2	1	5	3	5	2	8
4	4	2	1	3	3	4	7	1
5	2	3	3	6	1	8	5	2
6	4	5	2	7	1	9	3	3
7	2	4	2	2	3	1	9	1

(a) Cube C represented as an array A
(a) 用一个数组 A 表示cube C

Index	0	1	2	3	4	5	6	7
0	3	8	9	11	13	17	23	26
1	10	18	21	29	39	50	57	62
2	12	24	29	40	53	67	78	88
3	15	29	35	51	67	86	99	117
4	19	35	42	61	80	103	123	142
5	21	40	50	75	95	126	151	172
6	25	49	61	93	114	154	182	206
7	27	55	69	103	127	168	205	230

(b) Prefix sum cube PC
(b) 前缀和cube PC

Fig.1 A 2-dimensional cube C and its corresponding PC
图1 一个2-维cube C及其对应的PC

图2给出了计算过程的示意图.可以看出,PC技术把Cube上的区域查询简化为访问PC元素的问题.虽然PC方法提供了复杂性为 $O(1)$ 的区域查询,但最坏情况下,数据更新复杂性为 $O(n^d)$.

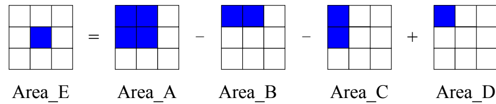

Fig.2 A range query on 2-dim PC
图 2 二维 PC 上的区域查询方式

虽然 PC 方法提供了复杂性为 $O(1)$的区域查询,但却带来了严重的数据更新复杂性,数据更新复杂性为 $O(n^d)$.文献[2]改进了 PC 方法,提出了 Relative Prefix Sum(RPS)存储策略,将更新复杂性降至 $O(n^{\sqrt{d}})$.文献[3]提出了 Double Relative Prefix Sum 算法,使查询与更新复杂性分别为 $O(n^{1/3})$和 $O(n^{d/3})$.文献[5]改进了文献[3]的存储结构,使其额外存储代价减为 n^d,同时保持了原有的查询与更新代价.文献[6]提出了两种树形 Cube 存储结构:HRC 和 HBC.从查询与更新两方面综合考虑,HBC 的性能优于 RPC.然而 HBC 与 HRC 的结构过于复杂,难以实现.文献[7]提出了 Dynamic Data Cube(DDC).从文献[7]的证明过程可以看到,当每维采用二分划分时,DDC 的查询代价为 $O((2^d-1)\log^d n)$,更新代价为 $O(\log^d n)$.文献[7]把(2^d-1)视为常数,得到查询的代价为 $O(\log^d n)$.实际上,d 是 Cube 的维数.对不同 Cube,d 是不同的.因此,(2^d-1)并不是一个常数,DDC 的查询代价应为 $O((2^d-1)\log^d n)$.尽管如此,与以往的方法相比,DDC 方法仍具有最小综合代价.但是,该方法的问题是必须假定 Cube 的每一维具有相同的基数 n,否则难以保证树的平衡性.然而,在实际应用中,这一点是很难保证的.另外,当 Cube 的维数增加时,整个树型存储结构不仅变得十分复杂,使得计算代价变得越来越高,而且需要大量额外的存储空间.

文献[8]在 PC 结构基础上增加了一种类似于 R 树的辅助存储结构,称为Δ-tree.该方法的问题是:(1) 当Δ-tree 中更新的数据积累到一定程度时,系统需要批量更新整个 PC 的代价.图 3 给出了具有不同规模数据量的 5-维 PC 的批量更新代价.由图中可以看到,当数据量达到 512G 以后,批量更新所需时间急剧增长,需要几天甚至十几天的时间.(2) 在实际应用中,更新频繁的数据往往也是决策者所关心的热点数据,因此区域查询常常需要同时访问 PC 与Δ-tree 树.众所周知,使用 R-Tree(即Δ-tree 树)结构处理区域查询的时间复杂性非常大.因而,当Δ-tree 树很大时,区域查询的代价远高于其他存储结构.文献[8]已经意识到这两个问题,所以建议使用该方法进行区域查询的近似处理.遗憾的是,当数据量很大而且数据更新频率较高时,这种方法为了控制查询时间代价上升的速度,必须增加 PC 更新的频率,导致该方法的更新代价与文献[4]给出的方法相同,即 n^d,而查询代价却远高于文献[4]给出的方法,进而其综合代价也远高于文献[4]给出的方法.

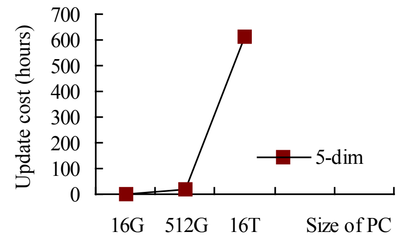

Fig.3 Comparison of update cost for different size of PC
图 3 不同大小 PC 的更新时间对比

请注意,文献[2,3,5~7]和本文提出的方法都是为了在更新频率较高的大数据仓库条件下改善文献[4]给出的方法.由于文献[8]给出的方法在这种条件下性能低于文献[4]给出的方法,不适用于这种条件,因此本文以后不再考虑这种方法.

2 层次式 Cube 存储结构 HDC 及相关算法

2.1 HDC存储结构及其构造过程

定义 1(**划分点**). 设 $A=\times_{i=1}^{d}[0,n-1]$是一个大小为 n^d的 d-维空间,l,b_i为非负整数.我们称 $P_i^l=\{(j*n/b_i^l)-1|1\le j<b_i^l, b_i^l\le n, \text{ and } j \bmod b_i\ne 0\}$为维 i 上的第 l 层划分点集合.$\forall x_i\in P_i^l$,称 x_i 为维 i 上第 l 层的一个划分点.

在下面的讨论中,不失一般性,我们假定 $b_1=b_2=\ldots=b_d=b=2$.设 $C\subseteq A=\times_{i=1}^{d}[0,n-1]$是一个 d-维 Cube.利用定义 1 中划分点对 A 进行层次式划分,相应地在 C 的各维上产生了 $\log n+1$ 层划分点集合,即 $P_i^0,P_i^1,\ldots,P_i^{\log n}$ $(1\le i\le d)$.$P_i^0\cup P_i^1\cup\ldots\cup P_i^{\log n}$ 是第 i 维的值域,$|P_i^0\cup P_i^1\cup\ldots\cup P_i^{\log n}|=n$.

图 4 描述了一个大小为 8×8 的 2-维 Cube C 的划分过程,及每次划分后在各个维上得到的划分点集合.由图可见,C 第 1 层划分后在每维上得到划分点集合:$P_1^1=P_2^1=\{3\}$.第 2 层划分后,每维上得到划分点集合:$P_1^2=P_2^2=\{1,5\}$.最后,经过第 3 层划分,每维上得到划分点集合为 $P_1^3=P_2^3=\{0,2,4,6\}$.$P_1^0=P_2^0=\{7\}$是第 0 层上的

划分点集合.

定义 2(Cube 存储结构 HDC). 设 Cube $C=\times_{i=1}^{d}[0,n-1]$ 是一个 d-维 Cube,$P_i^0,P_i^1,\ldots,P_i^{\log n}(1\le i\le d)$是 C 在第 i 维上的划分点集合.C 所对应的 HDC 是一个与 C 具有同样大小的 d-维数组.HDC 中任一单元 $c(h_1,h_2,\ldots,h_d)=f(\{C(x_1,x_2,\ldots,x_d)|\forall(x_1,x_2,\ldots,x_d)\in S\})$,$f$ 是聚集函数,$S=[l_1,h_1]\times[l_2,h_2]\times\ldots\times[l_d,h_d]\subseteq C$.$l_1,l_2,\ldots,l_d$ 的值如下确定:

对$\forall h_i$,必存在一个 $k(0\le k\le\log n)$,使得 $h_i\in P_i^k$.检查 $P_i^0\cup P_i^1\cup\ldots\cup P_i^{k-1}$ 中是否存在 p_i,使其满足:

(1) $p_i<h_i$,并且

(2) $\forall p_i'\in P_i^0\cup P_i^1\cup\ldots\cup P_i^{k-1}$,有 $p_i'\le p_i$,

若存在这样的 p_i,则 $l_i=p_i+1$;否则 $l_i=0$.

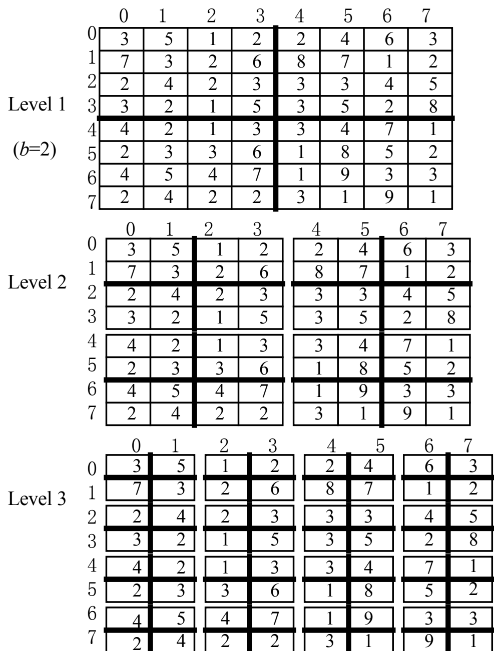

Fig.4 The hierarchical partition procedure on a 8×8 cube, and the sets of corresponding partition points for each dimension

图 4 8×8 的 cube 划分过程及每一层划分后在各维上得到的划分点集合

图 5 给出了按图 4 描述的划分过程产生的 HDC,其中 f=sum. 下面,对于任意 d-维数组 ARR,我们用 $ARR[x_1{:}y_1;x_2{:}y_2;\ldots,x_d{:}y_d]$表示数组 ARR 中的区域$[x_1,y_1]\times[x_2,y_2]\times\ldots\times[x_d,y_d]$.对于阴影部分的点 $HDC(7,7)$,$h_1=7\in P_1^0$,$k=0$.在 $P_1^0\cup\ldots\cup P_1^{k-1}$ 中不存在比 7 小的划分点,由 HDC 构造规则可得 $l_1=0$,$h_2=7\in P_2^0$,$k=0$,于是 $l_2=0$,$HDC(7,7)$记录了图 4 中 $C[0{:}7;0{:}7]$内所有元素之和 230.对于 $HDC(6,7)$,$h_1=6\in P_1^3$,$k=3$.由 $p_1=5\in P_1^0\cup P_1^1\cup P_1^2$,且$\forall p_1'\in P_1^0\cup P_1^1\cup P_1^2$,有 $p_i'\le 5$,于是 $l_1=p_1+1=5+1=6$;$h_2=7\in P_2^0$,$k=0$.$P_2^0\cup\ldots\cup P_2^{k-1}$ 中不存在比 7 小的划分点,$l_2=0$.于是 $HDC(6,7)$记录了图 4 中 $C[6{:}6;0{:}7]$内元素之和 34.

2.2 HDC上的区域查询

	0	1	2	3	4	5	6	7
0	3	8	1	11	2	6	6	26
1	10	18	3	29	10	21	7	62
2	2	6	2	11	3	6	4	26
3	15	29	6	51	16	35	13	117
4	4	6	1	10	3	7	7	25
5	6	11	4	24	4	16	12	55
6	4	9	2	18	1	10	3	34
7	27	55	14	103	24	65	37	230

Fig.5 The HDC according to Fig.4

图 5 按图 4 划分后对应的 HDC

如图 2 所示,Cube 上任意区域的聚集查询都可以转化为计算若干个由起始单元到 Cube 的某一单元所覆盖的区域的聚集.我们把由 $C(0,0,\ldots,0)$到 C 中任一单元所覆盖区域上的查询称为前缀区域查询.利用 HDC 可以有效地完成任一前缀区域查询.在下面的讨论中我们假定聚集函数为 Sum.

图 6 给出了计算图 2(a)中区域 $Q=A[0{:}6;0{:}4]$内所有点之和的过程.区域 Q 包含了 6 个子区域:R,S,T,U,V 和 W.区域 Q 的聚集和可以由这 6 个子区域的聚集和得到,即 $\mathrm{Sum}(Q)=\mathrm{Sum}(R)+\mathrm{Sum}(S)+\mathrm{Sum}(T)+\mathrm{Sum}(U)+\mathrm{Sum}(V)+\mathrm{Sum}(W)$.由该 2-维数组对应的 HDC(如图 5 所示)可知,子区域 $R[0{:}3;0{:}3]$的聚集和为 $HDC(3,3)$,子区域 $S[4{:}5;0{:}3]$的聚集和为 $HDC(5,3)$,子区域 $T[0{:}3;4{:}4]$的聚集和为 $HDC(3,4)$,子区域 $U[4{:}5;4{:}4]$的聚集和为 $HDC(5,4)$,子区域 $V[6{:}6;0{:}3]$的聚集和为 $HDC(6,3)$,子区域 $W[6{:}6;4{:}4]$的聚集和为 $HDC(6,4)$.于是,

$$\mathrm{Sum}(Q)=HDC(3,3)+HDC(5,3)+HDC(3,4)+HDC(5,4)+HDC(6,4)=114.$$

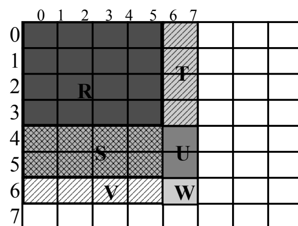

Fig.6 Calculate the sum of all cells in [0:6;0:4]

图 6 计算[0:6;0:4]内各点的聚集和

问题的关键是如何来确定这 6 个子区域.首先分别在每一维的第 1 层划分点集合中查找位于区间[0:6]和区间[0:4]内的最大划分点的值,在第 1 维的 P_1^1 中找到 3,在第 2 维的 P_2^1 中找到 3,得到了最大子区域

$R[0:3;0:3]$;其次,分别在每一维的第 2 层划分点集合中查找位于区间[4:6]和区间[4:4]内的最大划分点的值,在第 1 维的 P_1^2 中找到 5,而在第 2 维的 P_2^2 中未找到相应的划分点,得到了次大子区域 $S[4:5;0:3]$;最后,在每一维的第 3 层划分点集合中查找位于区间[6:6]和区间[4:4]内的最大划分点的值,在第 1 维的 P_1^3 中找到 6,在第 2 维的 P_2^3 中找到 4,得到其他子区域 T,U,V 和 W.实际上,我们是在区域 Q 内的相应每一维上寻找划分点集合 $P_1'=\{3,5,6\}$ 和 $P_2'=\{3,4\}$.可以看出,P_1'将区间[0:6]分割为 3 个子区间:[0:3],[4:5]和[6:6],而 P_2'将区间[0:4]分割为两个子区间:[0:3]和[4:4].于是,这些子区间将 Q 围成了$|P_1'|\times|P_2'|=6$ 个子区域:[0:3;0:3],[0:3;4:4],[4:5;0:3],[4:5;4:4],[6:6;0:3]和[6:6;4:4],即得到了 R,S,T,U,V 和 W.

下边是 HDC 上的区域查询处理算法的形式化描述:

HDC_range_query(Q)

算法输入:Prefix range query $Q[0:h_1;0:h_2;\ldots 0:h_d]$;

算法输出:聚集结果 SUM=Sum(Q).

(1) SUM=0;

/* 第 1 步:把 Q 拆分为一组子区域 $Q_1,Q_2,\ldots,Q_t$.首先,在 Q 的各个维上确定划分点集合 $T_1,T_2,\ldots,T_d$ */

(2) FOR i=1 TO d DO

(3) l_i=0; end=false; j=0; $T_i=\varnothing$;

(4) WHILE (end=false) DO /* 在 P_i^j 中查找包含在$[l_i:h_i]$内的最大划分点值 p,若未找到,$p=-1$ */

(5) p=max_partition_point(P_i^j,l_i,h_i);

(6) IF ($p\neq-1$) THEN

(7) $T_i=T_i\cup\{p\}$; /* T_i 是有序集合 */

(8) IF ($p\neq h_i$) THEN $l_i=p+1$;

(9) ELSE end=true;

(10) $j=j+1$; /* END WHILE

/* 其次,根据第 i 维的划分点集合 $T_i=\{p_{i1},p_{i2},\ldots,p_{it_i}\}(1\le i\le d,\ p_{it_i}=h_i)$,在第 i 维区间$[0:h_i]$上得到 t_i 个分段$[0:p_{i1}]$, $[p_{i1}+1:p_{i2}],\ldots,[p_{it_i-1}+1:h_i]$ */

(11) $l_1=0;\ l_2=0;\ldots;\ l_d=0$;

(12) FOR i=1 TO d DO

(13) $U_i=\varnothing$;

(14) FOR j=1 TO t_i DO

(15) $U_i=U_i\cup\{$由$[l_i:p_{ij}]$构成的分段 $u_j\}$;

(16) $l_i=p_{ij}+1$;

/* 最后,由 $U_1,U_2,\ldots,U_d$ 中的各分段,确定子区域集合$\{Q_1,Q_2,\ldots,Q_t\}$ */

(17) SQ=空集合;

(18) FOR $\forall(1\le j_1\le t_1,1\le j_2\le t_2,\ldots,1\le j_d\le t_d)$ DO

(19) $SQ=SQ\cup\{Q[u_{1j_1};u_{2j_2};\ldots;u_{dj_d}]\}$;

/* 第 2 步:使用 HDC 计算每一个子区域的聚集值,进而计算 Sum(Q) */

(20) FOR 每个 $Q_r[l_{r1}:h_{r1};l_{r2}:h_{r2};\ldots;l_{rd}:h_{rd}]\in SQ$ DO

(21) Sum(Q_r)=HDC($h_{r1},h_{r2},\ldots,h_{rd}$)

(22) $SUM=SUM$+Sum(Q_r); /* 计算 Sum(Q) */

定理 1. 设 $Q[0:h_1;0:h_2;\ldots 0:h_d]$是算法 HDC_range_query 的输入,$Q_1,Q_2,\ldots,Q_m$ 是算法 HDC_range_query 第 1 步得到的子区域集合,则:(1) $Q=Q_1\cup Q_2\cup\ldots\cup Q_m$;(2) $Q_i\cap Q_j$=空集合,$i\neq j$.

证明:略.

定理 2. 设 $Q_1,Q_2,\ldots,Q_m$ 是算法 HDC_range_query 第 1 步得到的子区域集合,则对于任一 Q_r,HDC 中必存在一个单元 HDC($x_1,x_2,\ldots,x_d$),使得 sum(Q_r)=HDC($x_1,x_2,\ldots,x_d$).

证明:略.

定理 1 和定理 2 说明算法 HDC_range_query 正确地完成了 $Q[0{:}h_1;0{:}h_2;\ldots 0{:}h_d]$的聚集计算.

2.3 HDC上的数据更新

当 Cube C 上某单元 c 的值发生改变时,需更新 HDC 中相关数据.HDC 的更新过程是沿着每一维划分的层次路径,逐层修改 HDC 中与 c 有关的信息.首先在 HDC 中搜索与 c 相对应的单元 e,并根据 c 的新值来修改 e 值,同时计算 c 的新、旧值之间的差Δ.然后,利用Δ修改 HDC 中每个满足下列条件单元(设为 c')的值:(1) c'是 C 中某区域 r 内所有数据的聚集值;(2) r 包含 c.

例如,我们欲修改图 4 中单元 $C(2,2)$的值.由图 7 可以看到,在修改与 $C(2,2)$对应的被标记为“?”的 $HDC(2,2)$的同时,还要修改 HDC 中标记为“*”的所有单元.原因是这些单元所描述的聚集区域覆盖了 $C(2,2)$.例如,HDC(3,3)是图 4 的 $C[0{:}3;0{:}3]$内所有元素之和,$C(2,2)\in C[0{:}3;0{:}3]$,故当 $C(2,2)$发生改变时,需要修改 $HDC(3,3)$.如何确定与 $C(2,2)$相关的单元呢?由图 4 可知,图 7 对应的 C 在每一维上具有 4 个划分层次.首先,由第 0 层开始,分别搜索 P_1^0 和 P_2^0 中包含在区间[2:7]内的所有划分点,得到划分点“7”;然后,在 P_1^1 和 P_2^1 中搜索包含在区间[2:6]内所有划分点,得到“3”;继续在 P_1^2 和 P_2^2 中搜索包含在区间[2:2]内所有划分点;最后,在 P_1^3 和 P_2^3 中搜索包含在区间[2:2]内所有划分点,得到“2”.这样,我们在每维上都得到了一组划分点的有序集合,即 $T_1=\{7,3,2\}$,$T_2=\{7,3,2\}$.以 T_1 和 T_2 中任意点为坐标的那些单元即为与 c 相关的、需要修改的单元.HDC 上的数据更新算法如下:

Fig.7　Update the cell ‘?’
图 7　修改“?”标记的单元

HDC_update(Q)

算法输入:某一个单元 c 的位置$(x_1,x_2,\ldots,x_d)$及更新值 new_c;

算法输出:更新后的 HDC.

(1) 确定 HDC 中与 c 相对应的单元 $e(x_1,x_2,\ldots,x_d)$;

(2) 计算新、旧 c 数据的差Δc;

(3) FOR　i=1　TO　d　DO

(4)　　$h_i=n-1$;　end=false;　$j=0$;　$T_i=\varnothing$;

(5)　　WHILE　(end=false)　DO /*在 P_i^j 中查找包含在$[x_i{:}h_i]$内的划分点 p,若未找到, $p=-1$*/

(6)　　　　$p=partition_point(P_i^j,x_i,h_i)$;

(7)　　　　IF ($p\neq-1$) THEN

(8)　　　　　　$T_i=T_i\cup\{p\}$;

(9)　　　　　IF ($x_i\neq p$)　THEN　$h_i=p-1$;

(10)　　　　　　ELSE　end=true;

(11)　　　　$j=j+1$;

(12) FOR　$\forall(y_1\in T_1,y_2\in T_2,\ldots,y_d\in T_d)$ DO

(13)　　$HDC(y_1,y_2,\ldots,y_d)=\Delta c+HDC(y_1,y_2,\ldots,y_d)$;

定理 3. 设 C 为一个 d-维 Cube,$c(y_1,y_2,\ldots,y_d)\in C$ 是算法 HDC_update 的输入(即欲修改数据),$S=\{HDC(z_1,z_2,\ldots,z_d)\}$是 HDC_update 第(14)步产生的 HDC 的子集合,则:(1) $\forall HDC(z_1,z_2,\ldots,z_d)\in S$,$HDC(z_1,z_2,\ldots,z_d)$聚集区域包含 $c(y_1,y_2,\ldots,y_d)$,(2) S 是 HDC 中满足条件(1)的所有单元的集合.

证明:略.

定理 3 说明,算法 HDC_update 不但正确地修改了与 $c(y_1,y_2,\ldots,y_d)$对应的 HDC 单元,也正确地修改了 HDC 中所有与 $c(y_1,y_2,\ldots,y_d)$相关的单元.于是,算法 HDC_update 是正确的.

3 性能分析与实验结果

3.1 性能分析

给定任意大小为 n^d 的 d-维 Cube C,n 是每维值域的大小,根据 HDC 的定义,C 对应的 HDC 需要与 C 同样大小的存储空间,即 n^d.从 HDC_range_query 的定义可知,算法的第 1 步把 Q 拆分为 HDC 可计算子区域集合 $\{Q_1,Q_2,\ldots,Q_t\}$,无须 I/O 操作.第 2 步只包括 1 个 d 重循环,最坏情况下,每重循环执行 log$n$ 次,d 重循环所需磁盘 I/O 为 $\log^d n$.于是,最坏情况下,HDC 区域查询的复杂性为 $O(\log^d n)$.类似地,最坏情况下,数据更新代价亦为 $O(\log^d n)$.表 1 给出了我们的 HDC 存储结构与文献[2~7]提出的 Cube 存储结构的性能比较.从表 1 可以看到,我们的 Cube 储存结构 HDC 具有最优的综合性能.

Table 1 Performance comparison between different methods

表 1 各种方法性能对比

Method	Query	Uptate	$c_q \cdot c_u$ model	$c_q \cdot n_q + c_u \cdot n_u$ model	Space
Naive	$O(n^d)$	$O(1)$	$O(n^d)$	$O(K\cdot n^d)$	n^d
PS	$O(1)$	$O(n^d)$	$O(n^d)$	$O(K\cdot n^d)$	$2n^d$
RPS	$O(1)$	$O(n^{d/2})$	$O(n^{d/2})$	$O(K\cdot n^{d/2})$	$2\cdot n^d + n^d\left(1-\left(\frac{\sqrt{n}-1}{\sqrt{n}}\right)^d\right)$
DRPS	$O(n^{1/3})$	$O(n^{d/3})$	$O(n^{(1+d)/3})$	$O(K\cdot n^{d/3})$	$2\cdot n^d + d\cdot n^{d-1/3}$
BRPS	$O(n^{1/3})$	$O(n^{d/3})$	$O(n^{(1+d)/3})$	$O(K\cdot n^{d/3})$	$2\cdot n^d$
DDC	$O((2^d-1)\cdot\log^d n)$	$O(\log^d n)$	$O((2^d-1)\cdot\log^{2d} n)$	$O(K\cdot 2^d\cdot\log^d n)$	$\sum_{h=1}^{\log n}\left(\frac{n}{2^i}\right)^d\cdot\left(\left(\frac{n}{2^h}\right)^d-\left(\frac{n}{2^h}-1\right)^d\right)>3n^d$
HDC	$O(\log^d n)$	$O(\log^d n)$	$O(\log^{2d} n)$	$O(K\cdot\log^d n)$	$2n^d$

3.2 实验结果

DDC 是目前对 PS 改进最好的 Cube 存储结构,其他方法的性能都低于 DDC.我们仅从区域查询代价和综合代价两方面比较 HDC 和 DCC 的优劣.影响算法性能的因素主要有两个:一个是 Cube 的数据量;另一个是 Cube 的维数.实验从这两方面,对 HDC 与 DDC 进行了对比.实验中 HDC 与 DDC 均采用二分划分,即 k=2.

3.2.1 区域查询性能对比

第 1 组实验测试当维数不变时,Cube 数据量对查询性能的影响.维数 d 分别为 3,4,5,测试数据的大小分别为 s1=64×64×64, s2=128×128×128, s3=512×512×512, s4=64×64×64×64, s5=128×128×128×128, s6=512×512×512×512,s7=64×64×64×64×64,s8=128×128×128×128×128 和 s9=512×512×512×512×512.图 8~图 10 分别给出了不同维数的测试结果,纵坐标表示区域查询算法需要访问的单元个数.从这些图可以看出,当维数较低时(如 d=3),Cube 数据量的增长对查询性能产生的影响很小;当维数增高时,随着 Cube 数据量增大,DDC 查询过程中访问的单元数急剧增多,使其查询代价急剧增大,如图 10 中 d=5 对应的曲线.

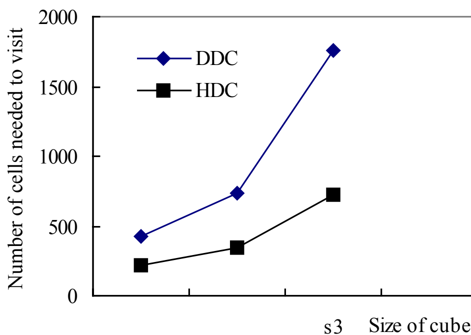

Fig.8 Effect of cube size on query (d=3)

图 8 Cube 大小对查询的影响(d=3)

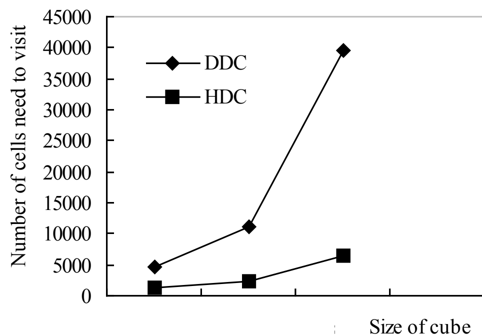

Fig.9 Effect of cube size on query (d=4)

图 9 Cube 大小对查询的影响(d=4)

第 2 组实验的目的是测试维数变化对查询性能的影响.我们采用了高、中、低 3 种不同规模的测试数据.T

级以上的数据称为高等规模数据.几十至几百 G 以上的数据称为中等规模数据.几个 G 或几百 M 的数据称为低等规模数据.实验中 3 种规模数据分别取为 2^{40},2^{36},2^{32}.实验结果如图 11~图 13 所示.从图中可以看出,对于相同规模的数据,变化维数只引起 HDC 查询性能的缓慢变化.而对于 DDC 来说,当数据量规模较低时,变化维数对查询性能的影响不是很大,如图 11 所示.然而,随着数据量规模的增大,维数的变化对查询性能的影响逐渐增大,如图 13 所示,当维数由 4 变化到 5 时,DDC 查询所需的单元个数陡然增加.

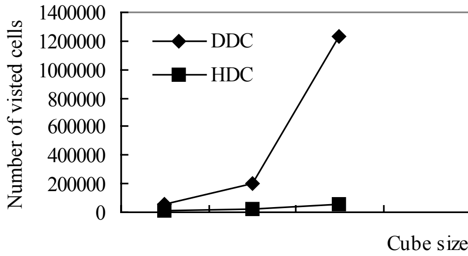

Fig.10 Effect of cube size on query (d=5)

图 10 Cube 大小对查询的影响(d=5)

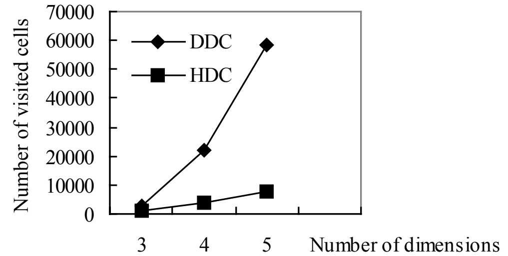

Fig.11 Effect of number of dimension on querying small-scaled cubes

图 11 在小规模数据上,维数对查询的影响

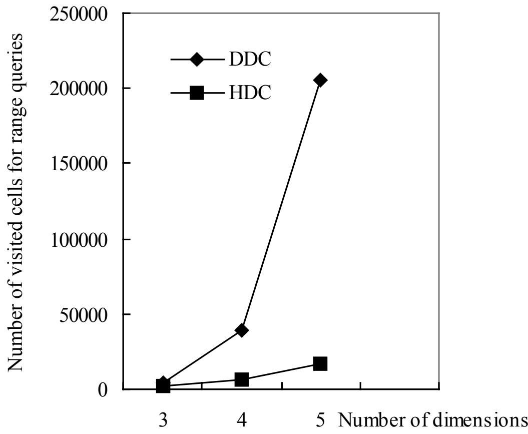

Fig.12 Effect of number of dimensions on querying medium-scaled cubes

图 12 在中等规模数据上,维数对查询的影响

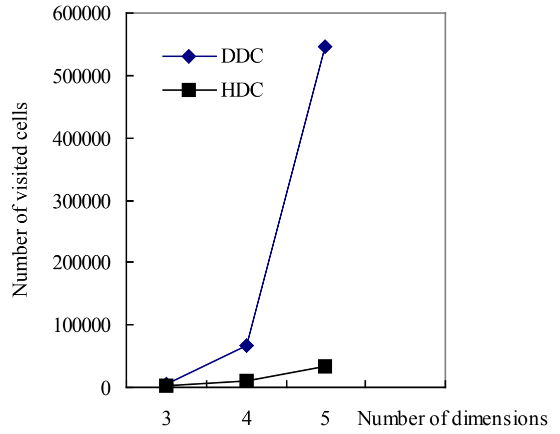

Fig.13 Effect of number of dimension on querying large-scaled cubes

图 13 在大规模数据上,维数对查询的影响

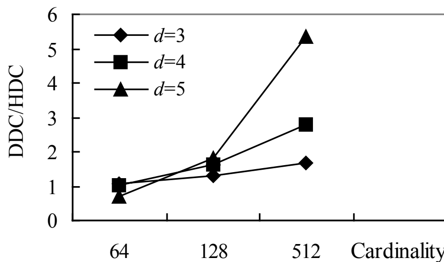

Fig.14 Performance comparison under $C_q \cdot C_u$ model

图 14 基于 $C_q \cdot C_u$ 模型的性能对比

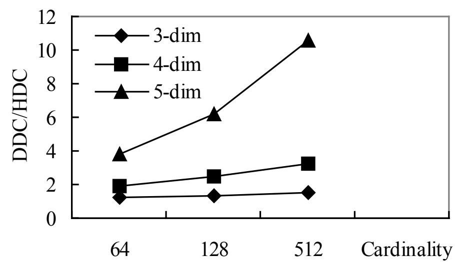

Fig.15 Performance comparison under $N_q \cdot C_q + N_u \cdot C_u$ model

图 15 基于 $N_q \cdot C_q + N_u \cdot C_u$ 模型的性能对比

3.2.2 综合评价模型对比

我们在与第 3.2.1 节第 1 组实验中相同的一组数据集合上进行了实验,考察了 HDC 与 DDC 在两种不同评价模型下的综合性能.从图 14 可以看到,在 $C_q \cdot C_u$ 模型下,当数据集合的基数比较小,即数据量的规模比较低的时候,DDC 与 HDC 的综合性能差别不是很大,HDC 的性能甚至低于 DDC.但是,当维的基数增大(≥5),数据量的规

模比较高时,DDC 与 HDC 的综合性能出现很大的差别,HDC 的综合性能明显高于 DDC.从图 15 可以看到,在 $C_q{\cdot}n_q+C_q{\cdot}n_u$ 模型下(假定 $n_q=n_u=10$),当数据集合的维数比较低时(d=3 或 4),DDC 与 HDC 的综合性能差别不是很大.但是,当维数增高(≥5)时,DDC 与 HDC 的综合性能出现巨大的差别,尤其是当维度较高,同时维的基数也较大时,HDC 的综合性能明显优于 DDC.

4 结 语

本文提出了层次式 Cube 存储结构 HDC.它的区域查询代价、数据更新代价、空间复杂性以及综合代价都低于已有的 Cube 存储结构.我们将进一步优化 HDC 存储结构,使它能够更加有效地支持数据仓库上的区域查询与批量追加更新.

References:

[1] Gray J, Bosworth A, Layman A, Pirahesh H. Data cube: A relational aggregation operator generating group-by, cross-tab and sub-total. In: Marek R, ed. Proceedings of the 12th ICDE. IEEE Press, 1996. 152~159.

[2] Geffner S, Agrawal D, Abbadi A, Smith T. Relative prefix sums: an efficient approach for querying dynamic OLAP data cubes. In: Alberto O, ed. Proceedings of the 15th International Conference on Data Engineering. IEEE Press, 1999. 328~335.

[3] Liang W, Wang H, Orlowska ME. Range queries in dynamic OLAP data cubes. Data and Knowledge Engineering, 2000,34(1): 21~38.

[4] Ho CT, Agrawal R, Megiddo R, Srikant R. Range queries in OLAP data cubes. In: Joan P, ed. Proceedings of the International ACM SIGMOD Conference. ACM Press, 1997. 73~88.

[5] Li HG, Ling TW, Lee SY, Loh ZX. Range sum queries in dynamic OLAP data Cubes. In: Lu HJ, Stefano S, eds. Proceedings of the 3th International Symposium on Cooperative Database Systems for Advanced Applications (CODAS 2001). IEEE Computer Society Press, 2001. 74~81.

[6] Chan CY, Ioannidis YE. Hierarchical cubes for range-sum queries. In: Bassiouni A, ed. Proceedings of the 25th VLDB Conference. IEEE Press, 1999. 675~686.

[7] Geffner S, Agrawal D, Abbadi AE. The dynamic data cube. In: Zaniolo C, Lockemann PC, Scholl MH, Grust T, eds. Proceedings of the EDBT. LNCS 1777, Heidelberg: Springer-Verlag, 2000. 55~77.

[8] Chun S J, Chung C W, Lee J H, Lee S L. Dynamic update Cube for range-sum queries In: Peter MG, ed. Proceedings of the 27th VLDB Conference. IEEE Press, 2001. 521~530.