CBFrag-Cubing:一种基于压缩位图的高维数据立方创建算法*)

刘运涛 鲍玉斌 吴 丹 冷芳玲 孙焕良 于 戈
(东北大学信息科学与工程学院 沈阳 110004)

摘 要 数据立方的计算是数据仓库和OLAP研究的一个重要方向,同时又是数据仓库中代价很大的操作。针对在生物信息、统计分析、文本处理等领域中存在的基数较小的高维数据集,X. L. Li 等人提出了 Frag-Cubing 算法。为了提高 Frag-Cubing 算法的效率,本文提出了基于分片思想的算法 CBFrag-Cubing。该算法使用了位图索引结构,优化了数据立方的存储,减少了数据立方的计算时间。实验表明,与 Frag-Cubing 算法相比,该算法在存储空间上至少节省 25%,在计算时间上节省 30%。

关键词 数据立方计算,OLAP,Frag-Cubing,压缩位图

CBFrag-Cubing: An Algorithm of High-Dimensional Data Cubing Based on Compressed Bitmap

LIU Yun-Tao BAO Yu-Bin WU Dan LENG Fang-Ling SUN Huan-Liang YU Ge
(School of Information Science and Engineering, Northeastern University, Shenyang 110004)

Abstract The computation of data cube is one of the most essential but expensive issues in data warehouse and OLAP. Frag-Cubing algorithm is recently proposed to perform data cubing on such data sets, which exist in applications like bioinformatics, statistics, and text processing that characterized by high dimensionality and low cardinality. In order to enhance efficiency of Frag-Cubing algorithm, a partition-based CBFrag-Cubing algorithm is proposed. It uses Compressed bitmap index to optimize the storage of cube, and reduces the computing time. Experimental results show that, compared with Frag-Cubing algorithm, the CBFrag-Cubing algorithm saves at least 25% on storage space and about 30% on computing time.

Keywords Data cube computation, OLAP, Frag-Cubing, Compressed bitmap

1 引言

数据立方(Cube)计算是数据仓库和 OLAP 领域一个重要问题[1]。为了提高多维查询和分析的响应速度,通过实例化数据立方实现预先计算和存储多维聚合值是一种常用的方法。关于 Cube 实例化已有大量工作。例如,基于 ROLAP 的多维聚合运算[2],多维数组聚合[3],H-Cubing[4] 和 Star-Cubing[5]。由于数据立方的完全实例化在时间和空间上开销太大,因此研究人员提出用部分实例化和压缩的方法生成数据立方,例如 Iceberg Cube[4],Condensed[6],Dwarf[7] 和 Quotient Cube[8] 等。然而,这些方法都只适用于低维或中等维度数据集,对于生物信息处理、调查统计分析和文本处理等领域的数据集[9](其维数在 100 维以上,甚至高达 1000 维,元组数在 10^6 左右,而每个维的不同值的个数较少)却不适用。比如一个 100 维,每维有 10 个不同值的数据集,采用完全实例化的方法创建 Cube 可能就包含 11^{100} 个聚合 cell。Iceberg Cube[4] 根据一个设定的阈值剪去了大量的高维 cell,大大减少了存储空间和计算时间,然而,对于高维数据集它生成的 cell 数目依然很大。比如一个 60 万条记录,60 维的 base-cuboid,采用阈值 5,生成的 Cube 大约包含 2^{60} 个 cell。Quotient[8] 通过共享元组来压缩数据立方体的体积,同等条件下,采用该算法生成的 Cube 体积是 Iceberg Cube 的两倍以上[9],因此它也不适用于高维数据集。Dwarf[7] 的空间复杂度是 $O(T^{1+1/(\log dC)})$[10],其中 d 是维数,C 是每个维不同值的个数,T 是元组数。对于上述高维数据集,当 d 很大且 C 较小时,$\log_d C$ 就很小,因而空间复杂度仍然较高。

为了解决高维数据集 Cube 创建的问题,Xiaolei Li 等人提出了 Frag-Cubing 算法[9]。该算法采用垂直分片的方式把高维数据集分成互不相交的低维子集,称之为片段(Fragment),每个片段实例化得到局部数据立方,各片段之间通过元组标识(tid)进行联系。这种方法使高维数据集的数据立方创建时间和存储空间降到了线性。但采用倒排索引的方式索引数据立方,标准地求交运算实例化片断,造成运算速度较慢;采用值链结构存储数据立方,也存在冗余。文[11]采用位图的方法索引数据立方,与其它的索引方式相比,大大提高了运算速度,节省了存储空间。不过它是基于整个数据集的,对于高维数据集的数据立方,其计算时间和存储空间依然是指数级的,因此它也不适用于高维数据集数据立方的计算和存储。

基于分片和位图索引的思想,本文提出了一种高维数据立方的创建算法 CBFrag-Cubing,对垂直分片后的片段采用位图方式索引,同时对位图进行了压缩。由于位"与"运算高效率,使数据立方计算速度较大提高,压缩的位图也提高了存储效率。

*)本课题得到国家自然科学基金(编号:60173051),国家 863 高技术计划 CIMS 主题(编号:2003AA414210),教育部优秀青年教师科研教育奖励计划资助。刘运涛 硕士研究生,主要研究领域为数据仓库和 OLAP;鲍玉斌 副教授,主要研究领域为数据仓库、数据挖掘;于 戈 教授,博士生导师,主要研究领域为数据库理论与技术。

本文以下部分的组织结构如下：第2节介绍Frag-Cubing算法及其存在的问题，第3节具体介绍CBFrag-Cubing (Compressed Bitmap Frag-Cubing)算法的压缩位图索引结构并给出算法的详细描述，第4节是实验结果和分析，最后总结全文，并给出进一步的工作。

2 Frag-Cubing算法

通过对OLAP分析的统计分析发现，进行高维数据集的OLAP分析时，通常是只在少数的几个维上进行钻取和旋转操作，而其它的多数维要么取定值，要么取ALL[9]。因此，Frag-Cubing算法把高维数据集垂直分片成不相交的低维片断，每个片断进行完全实例化，而不是实例化整个数据集。用户查询时，如果查询所需的聚合操作的维不在一个片断中，则基于实例化的各个片断进行动态组合计算。实践表明，这种方法能够满足OLAP的响应时间要求。同时，它的数据立方的计算时间和存储空间与维数呈线性关系，因此是一个非常适用于每个维取值个数不多的高维数据集的数据立方创建的算法。示例数据集见表1，它有5个维，聚合操作假设为count()。对于其它的聚合操作在本节后面讨论。

表1 示例数据集

tid	A	B	C	D	E
1	a1	b1	c1	d1	e1
2	a1	b2	c1	d2	e1
3	a1	b2	c1	d1	e2
4	a2	b1	c1	d1	e2
5	a2	b1	c1	d1	e3

按照Frag-Cubing算法，首先把数据集垂直分片成(A,B,C)和(D,E)两个片断。在实际应用中，每个片断包含的维数和每个维分配到哪个片断中，可以由数据的语义和查询模式决定。实验表明：当每个片断的维数小于等于4时，存储空间和计算时间随着维数的增加都是呈线性增长的[9]。

下一步，需要构建这个数据集对应的倒排索引(见表2)。每个维的每个属性值对应表中的一行，例如属性值a2出现在元组4和5中，那么a2对应的元组tid链就包含4和5两项。

接下来，计算每个片断的数据立方。以片断(A，B，C)为例，它需要计算A,B,C,AB,AC,BC,ABC这7个cuboid。通过对表2中cuboid格采用自底向上、深度优先的方法把各个属性值对应的元组tid链相交得到。比如，计算cuboid AB的cell(a1 b2 *)，是把a1和b2的倒排索引的元组tid链求交运算，得到它对应的倒排索引{2,3}。表3是cuboid AB。

表2 维A,B,C,D,E的倒排索引

属性值	Tid List	size
a1	1 2 3	3
a2	4 5	2
b1	1 4 5	3
b2	2 3	2
c1	1 2 3 4 5	5
d1	1 3 4 5	4
d2	2	1
e1	1 2	2
e2	3 4	2
e3	5	1

表3 Cuboid AB

Cell	Intersection	Tid List	Size
a1 b1	1 2 3∩1 4 5	1	1
a1 b2	1 2 3∩2 3	2 3	2
a2 b1	4 5∩1 4 5	4 5	2
a2 b2	4 5∩2 3	Φ	0

同理，把cuboid AB对应的表3和表2中c1所在行逐个相交，可以得到cuboid ABC。对于表3中的(a2,b2)为空的情况，直接丢弃即可。同样的过程可以应用到片断(D,E)，从而得到它的局部数据立方。

在这个算法中，对不同类型的聚合操作采用了不同的计算方法。如元组数这样的聚合值，可以直接得到，因为它正好等于倒排索引的长度。对于avg或sum等运算，需要在最初遍历原始数据集时抽取出(tid,measure)这两列，根据查询得到的相关的元组标识，再进行单独的聚合运算，得到最终的查询结果。

Frag-Cubing算法把高维数据集进行垂直分片，每个片断完全实例化。查询时，如果相关的维不在一个片断内，则进行在线动态组合，即可得到查询结果。在保证查询响应时间要求的同时，数据立方的计算时间和空间复杂度也降到了线性，适用于本文所研究的这种高维数据集的Cube的创建。

在该算法中，倒排索引对应的元组标识链的相交运算和存储占有很大比重，而倒排索引采用整型数组的形式表示，对这种每个维的不同值的个数非常少的情况，存在冗余。比如只有男、女两个不同值的性别维，若是采用倒排索引的表示方法，每个元组标识需要32比特，而采用位图索引表示只需要1个比特。因此，对于在生物信息、调查统计和文本处理中广泛存在的大数据集，采用倒排索引结构会占用更多的存储空间。而且，倒排索引之间的求交运算也比位图结构的bit-AND运算慢很多。而位图索引表示的元组标识链比较短，运算时一次可以装载更多数据，也可以加快运算的速度，因此对于每个维不同值个数较少的高维数据集，采用位图索引的结构应该是一种更好的选择。

3 CBFrag-Cubing算法

本节将介绍压缩位图索引的结构、在算法中的应用及其对计算时间和存储空间的改善；最后给出算法的整体结构和描述。

3.1 CBFrag-Cubing的压缩位图结构

数据立方中可以使用多种索引，如B树索引、Hash表等。但是考虑到高维数据集每个维不同值个数非常少的具体特点，采用位图索引的结构，存储更加有效，而且它将标准的比较、连接和聚集都变成了位算术运算，大大减少了运行时间，从而极大地提升了系统性能[11]。因此，在CBFrag-Cubing算法中采用了压缩位图索引结构。

表4是和表2对应的A，B，C，D，E五个维上cell的位图索引。正如上述，在倒排索引中存储一个元组标识需要32比特，而在位图结构中只需要1个比特，除去一些位为零造成的冗余外，对于每个维的不同值的个数较少的高维数据集，压缩率依然是非常可观的。

假定在CBFrag-Cubing算法中也把表1的数据集分成(A,B,C)和(D,E)两个片断，下面以片断(A,B,C)的实例化为例来说明该算法的局部实例化操作。把表4中A,B两个

维的属性值对应的位图索引执行如表 5 所示的 bit-AND 运算，即可得到 Cuboid AB。类似的 Cuboid ABC 可以用 Cuboid AB 和 表 4 中的 c1 的对应的行进行 bit-AND 运算得到。从上表可以看出，该算法把倒排索引之间的相交运算转化为 bit-AND 运算，较大地提高了运算速度。

表 4　维 A,B,C,D,E 的位图索引

属性值	Tid Bitmap Index	Size
a1	11100000	3
a2	00011000	2
b1	10011000	3
b2	01100000	2
c1	11111000	5
d1	10111000	4
d2	01000000	1
e1	11000000	2
e2	00110000	2
e3	00001000	1

表 5　Cuboid AB 对应的位图索引结构

Cell	bit-AND	Tid List	Size
a1 b1	11100000&10011000	10000000	1
a1 b2	11100000&01100000	01100000	2
a2 b1	00011000&10011000	00011000	2
a2 b2	00011000&01100000	00000000	0

经过对实际数据集的数据分布研究分析和实验得知，这种位图索引结构也存在较大的冗余。假设一个 100 万条记录的真实数据集，可能比较多地存在图 1 所示的情况。cell a 和 cell b(属性值和它对应的位图索引以及相应的聚合值称为 cell)在位图的开头或结尾存在着大量连续为零的位；又如图中的 cell abc，其位图的前端和结尾也同时存在大量连续为零的位。这种情况在多维组合上的 cell 的位图中更为常见(如表 5)，因此很有必要消除这种冗余。

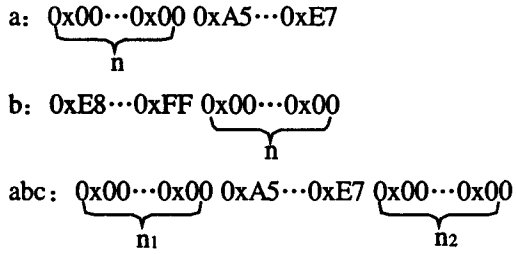

图 1　Cell 的位图索引结构

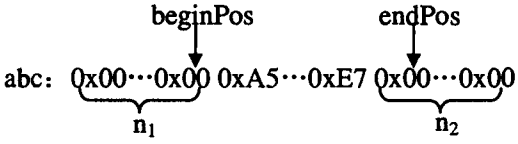

图 2　Cell 压缩位图索引结构

本文提出的 CBFrag-Cubing 算法，通过引入两个指针记录非 0 位出现的开始位置(即“始有效指针”)和结束位置(即“尾有效指针”)来压缩位图的存储(如图 2 所示)。这样，在实际存储中，只存储这两个位置指针以及它们之间的位图段，而不需要存储整行的位图，大大减少了无意义的前后缀全零字段冗余，节省了存储空间。

该压缩方法另一个优点是较多地减少了在这个算法占有很大比重的 bit-AND 运算以及内存的消耗。比如有两个 cell 对应的位图序列 Index1 和 Index2，Index1 的两个有效指针是 beginPos1 和 endPos1，Index2 的两个有效指针是 beginPos2 和 endPos2。这样的两个位图序列做 bit-AND 运算时，只需在 max(beginPos1，beginPos2)到 min(endPos1，endPos2)区间的各个位进行 bit-AND 运算，而不需要对整个位图序列做 bit-AND 运算。同时在存储运算结果时，只需要使用(endPos-beginPos ＋ 1)/8 个字节即可，而不需要使用(元组总数/8)个字节的内存单元，因此这个方法也减少了内存的消耗。

3.2　CBFrag-Cubing 算法及描述

CBFrag-Cubing 算法可以描述如下：

算法：CBFrag-Cubing 数据立方构建算法

输入：n 维事实表$(A_1,\cdots,A_n)$；

输出：1)一组片断$\{P_1,\cdots,P_k\}$和它们对应的 cube$\{S_1,\cdots,S_k\}$，P_k 表示一些维的集合。

2)如果聚合操作不是 count()，则输出 ID_measure 数组。

```
Begin
1. 分配 (A1,…,An)n 个维到 k 个分片 P1,…,Pk 中。
2. for 原始集的每个元组 t do {
3. 把每个〈tid,. measure〉插入到 ID_measure 数组中；
4. 对每个维 Ai 的每个元素 ai 构建它的位图索引〈ai,BitmapIndex〉
5. }
6. for 每个片段 Pi do {
7. 使用 bit-AND 计算局部 cube Si,
8. 计算相应的聚合值；
9. 把 Si 及其聚合值存储到磁盘上；
10. }
End.
```

CBFrag-Cubing 算法首先把数据集垂直分片(步骤 1)，扫描原始数据集，抽取〈tid，measure〉到 ID_measure 数组中(步骤 3)，同时建立各个属性值的位图索引(步骤 4)，然后通过位图之间的 bit-AND 运算得到每个片断的数据立方(步骤 7)，接下来采用和 Frag-Cubing 算法相同的策略计算相应片断的聚合值(步骤 8)，最后以压缩位图的形式存储到磁盘上(步骤 9)。

对于第 3 行，如果聚合操作是 count()，则不需要构建 ID_measure 数组，因为位图索引中“1”的个数就是元组数。对其他聚合操作，如 avg()，就需要保留 ID_measure 数组，并进行聚合计算。

由于位图索引本身的局限性[12]，这种优化算法，对每个维不同值的个数较多的情况，性能会下降。因为随着每个维不同值个数的增加，位图中为“0”的比特的比重必然增多，而且属性值在元组序列中的分布也可能更加随机，前后缀全零字段冗余消除方法的效果也会减弱。

4　实验结果与分析

本节在不同大小、不同维数和不同基数的数据集上对文中提出的算法进行了性能测试和比较。实验环境：Windows 2000 (professional) 系统，Intel Pentium 4 CPU 2.4GHz，512MB RAM。测试使用 KDD-CUP-99 数据集[13]，记录条数为 200000，同时为了和原算法保持一致，对它进行合理改造，以满足每个维不同值的个数较少的要求。以下各实验中每个片段的维数采用了与原算法中相同的数目 3。在图中，C 表示基数(Cardinality)，FC 表示 Frag-Cubing 算法，CB 表示 CB-Frag-Cubing 算法。

(下转封四)

(上接第 93 页)

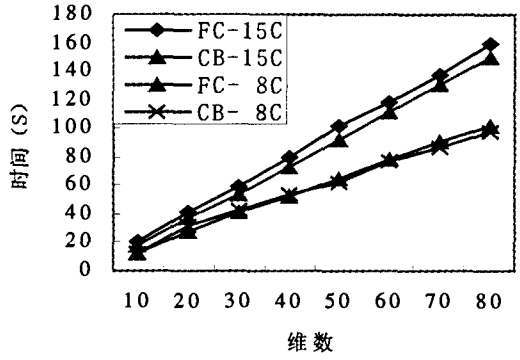

图 3 不同维数下创建时间的比较

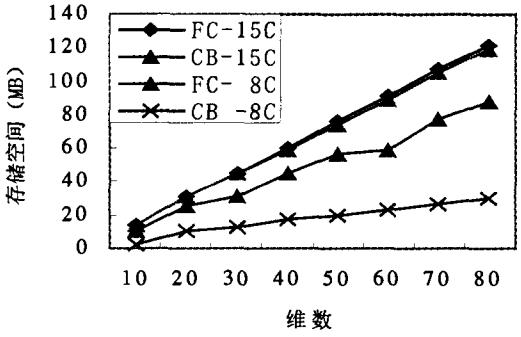

图 4 不同维数下存储空间的比较

图 3 是基数分别为 8 和 15 的情况下，Frag-Cubing 算法和 CBFrag- Cubing 算法的 Cube 计算时间随维数从 10 到 80 时所需时间的曲线。可以看出，新算法比原算法在时间上的提高是非常明显的，大约能节省 30%左右。新算法的数据立方创建时间随着维数的增加，也是接近线性增加的，并且每个维不同值个数对时间复杂度的影响也不明显。图 4 是基数分别为 8 和 15 的情况下，Frag-Cubing 算法和 CBFrag-Cubing 算法的数据立方存储所需的空间随维数从 10 到 80 时变化的曲线。两种不同的情况下，CBFrag-Cubing 算法都比 Frag-Cubing 算法得到了更高的压缩比，尤其是当每个维的不同值的个数为 8 时，压缩率达 80%。然而，也正如第 3 节所述，该算法由于受到位图索引本身的影响，当每个维的不同值的个数由 8 增加到 15 时，空间的压缩率降低较大。

综上所述，本文提出的 CBFrag-Cubing 算法非常适用于生物信息处理、调查统计分析和文本处理等领域广泛存在的维数很高而每个维不同值的个数较少的数据集的数据立方的创建，它比 Frag-Cubing 算法压缩率更高，运算速度更快。这对于高维 OLAP 的存储来说，有着非常重要的意义。

结论及未来的工作 本文提出了一种基于分片和压缩位图的高维 OLAP 数据立方计算和存储的算法 CBFrag-Cubing，它采用的压缩位图结构有以下优点：(1)基于位图的按位“与”运算(bit-AND)比基于倒排索引的合并求交运算快；(2)引入的“始有效指针”和“尾有效指针”较多地减少了 bit-AND 运算量及内存的消耗；(3)较大节省了存储数据立方所用的磁盘空间。从实验结果中可以看出，改进算法的数据立方计算时间可节省 30%，存储空间可节省 25%以上，比原算法更适用于每个维不同值个数较少的高维数据集。

当每个维的不同值的个数较多时，CBFrag-Cubing 性能会下降。下一步的工作将用编码位图[11]的方式来解决该问题。同时，作为一个 OLAP 系统，它的查询和增量更新与计算存储具有同等重要的作用，基于压缩位图的在线查询和增量更新是另一项要进行的研究工作。

参 考 文 献

1 Chaudhuri S, Dayal U. An Overview of Data Warehousing and OLAP Technology. In: SIGMOD, 1997, 26(1)
2 Agarwal S, Agrawal R, Deshpande P M, et al. On the computation of multidimensional aggregates. In : VLDB, 1996. 506～521
3 Zhao Y, Deshpande P M, Naughton J F. An array-based algorithm for simultaneous multidimensional aggregates. In: SIGMOD, 1997. 159～170
4 Han J, Pei J, Dong G, et al. Efficient computation of iceberg cubes with complex measures. In: SIGMOD, 2001
5 Xin D, Han J, Li X, et al. Starcubing: Computing iceberg cubes by top-down and bottom-up integration. In: VLDB, 2003. 476～487
6 Wang W, Lu H, Feng J, et al. Condensed cube: An effective approach to reducing data cube size. In: ICDE, 2002. 464～475
7 Sismanis Y, Roussopoulos N, Deligianannakis A, et al. Dwarf: Shrinking the petacube. In: SIGMOD, 2002
8 Lakshmanan L V S, Pei J, Han J. Quotient cube: How to summarize the semantics of a data cube. In: VLDB, 2002
9 Li X L, Han J W, Gonzalez H. High-Dimensional OLAP: A Minimal Cubing Approach. In: VLDB, 2004
10 Sismanis Y, Roussopoulos N. The dwarf data cube eliminates the high dimensionality curse: [TR-CS4552]. 2003
11 Wu M C, Buchmann A P. Encoded bitmap indexing for data warehouses. In: ICDE, 1998. 220～230
12 Chan C Y, Ioannidis Y E. Bitmap index design and evaluation. In: SIGMOD, 1998. 355～366
13 KDD CUP 1999 Data http: // kdd. ics. uci. edu/databases/- kddcup99/kddcup99. html

计算机科学

(1974 年 1 月创刊)
第 32 卷第 11 期 (月刊)
2005 年 11 月 25 日出版
ISSN 1002－137X
CN50—1075/TP
定价：25.00 元 国外定价：5 美元
邮发代号：78—68
发行范围：国内外公开

主管单位：国 家 科 学 技 术 部
主办单位：国家科技部西南信息中心
编辑出版：《计 算 机 科 学》杂 志 社
重庆市渝中区胜利路 132 号 邮政编码：400013
电话：(023) 63500828 E-mail：jsjkx@swic.ac.cn
网址：www. jsjkx. com
社 长：牟炳林
主 编：彭 丹
副 主 编：朱宗元
主编助理：徐书令
印 刷 者：重 庆 科 情 印 务 有 限 公 司
总发行处：重 庆 市 邮 政 局
订 购 处：全 国 各 地 邮 政 局
国外总发行：中国国际图书贸易总公司 (北京 399 信箱)
国外代号：6210—MO