用于数据仓储的一种改进的多维存储结构*

冯建华, 蒋旭东, 周立柱

(清华大学 计算机科学与技术系,北京 100084)
E-mail: fengjh@tsinghua.edu.cn
http://www.cs.tsinghua.edu.cn

摘要: 对于数据仓库中数据的物理存储组织,目前主要有关系和多维数组两种方式.这两种方式各有自己的优缺点,从提高联机分析处理(online analytical processing,简称 OLAP)查询处理性能的角度出发,多维数组方式相对较优,目的主要是解决数据仓库的多维存储结构问题.针对当前多维数组存储组织方式存在的一些问题,提出了 Cube(立方体)逻辑存储和物理存储的概念,首先将原多维数据空间划分为逻辑子空间,逻辑块再划分为多个物理块.在物理存储时充分考虑了多维数组的大容量和高稀疏度的问题,并采用新的多维数组的分布和压缩方法.这些概念和方法有效地解决了维内部层次结构的聚集操作和 Cube 操作的效率问题,显著提高了涉及维内部层次的聚集查询的响应速度,同时还解决了增量维护的效率问题.

关 键 词: 数据仓库;多维数组;聚集查询;区域查询;联机分析处理(OLAP)
中图法分类号: TP311 文献标识码: A

与传统的关系数据库相比,数据仓库中的数据及其建立在数据仓库之上的应用(如 OLAP)都具有自己的特点[1],因此,为了提高数据仓库系统的性能,可以采用新的技术用于数据的物理存储组织.对于数据仓库中数据的物理存储组织,目前主要有关系和多维数组两种方式[2],这两种方式各有自己的优缺点.

当采用关系方式存储数据仓库中的数据时,通常都是使用星型模式或者雪花模式来组织数据[2],也就是将维信息和事实信息分别存储在维表和事实表中,然后通过事实表的外码将事实表和维表关联起来.关系存储方式的主要优点是基于关系模式存储数据,从而可以继续使用已经比较成熟的关系数据库的理论和实现技术;而缺点是由于关系存储主要面向 OLTP(联机事务处理)应用进行优化,即使采用新的索引技术[3]以及面向星型结构进行优化的查询处理算法[4],OLAP(online analytical processing)查询处理的性能仍然相对较差.

因此,为了方便、高效地进行多维数据分析,就需要采用多维数组存储方式来进行多维数据的物理存储组织,也就是将事实数据直接存储为多维数组.对于多维数据集合 $R(D_1,D_2,\ldots,D_n,F_1,F_2,\ldots,F_m)$,其中 $D_1,\ldots,D_n$ 分别为相应 n 个维的取值集合,并且这 n 个维的笛卡尔积将构成一个 n 维空间,$F_1,\ldots,F_m$ 表示 m 个事实属性,我们可以使用 n 维数组来存储这 m 个事实属性.当然,实际存储到磁盘上时还需要首先将 n 维数组线性化为一维数组.多维数据直接存储到多维数组中,能够在多维数组的基础上方便地进行各种 OLAP 操作,OLAP 查询处理的响应速度很快,但相应地也会带来一些问题.如当数组比较稀疏时,存储空间的浪费就很大,需要对多维数组进行压缩存储.文献[5]提出了不存储完整数组,而只存储有非空(也就是事实属性值不全为 0)数组记录的数据压缩存储方法,对于每条数组记录,存成$\langle S,F_1,F_2,\ldots,F_m\rangle$的形式,$S$ 为该数组记录多维数组线性化后(多维数组的线性化参见文献[5,6])的线性偏移,$F_1,\ldots,F_m$ 表示 m 个事实属性.实际上我们还可以扩展对 S 的定义,在更一般化的情况下,S

* 收稿日期: 2001-03-07; 修改日期: 2001-09-19
基金项目: 国家重点基础研究发展规划 973 资助项目(G1998030414)

作者简介: 冯建华(1967－),男,山西运城人,副教授,主要研究领域为数据库,数据仓库,WWW 环境下的信息处理;蒋旭东(1972－),男,江苏徐州人,博士,工程师,主要研究领域为数据仓库,OLAP,查询处理;周立柱(1947－),男,江苏连云港人,教授,博士生导师,主要研究领域为数据库,数字图书馆,海量信息处理.

表示该多维数组记录在多维空间的位置信息,也就是说,S 由多维数组记录在各维的位置信息合成,并且能由 S 反算出各维位置信息.这样,在新的 S 值定义的前提下,我们从减少存储空间以及便于进行查询处理的角度出发,还可以提出新的数组压缩方法,具体参见第 1.2 节.

虽然已经有文献给出了部分对多维数组有效地进行存储组织的方法[5,7,8],但这些方法都还没有完全解决多维数组的高效存储过程中存在的一些问题.首先,数组过于稀疏会导致大量的存储空间的浪费,而使用数据压缩技术又会给 OLAP 查询处理带来额外开销,因此需要选择有效的适合多维数组的数据压缩方式.其次,多维数组初始建立后,当有增量数据需要进入时,对于现在的多维数组存储方式一般都需要重构多维数组,并且随着数组的膨胀,重构的开销也越来越大.另外,现在的多维数组结构都没有考虑如何有效地存储维内部的层次信息(关系存储方式通过使用雪花模式来处理维内部的层次问题),而事实上许多 OLAP 操作都是针对维内部的层次进行向上综合或者向下细化的,现在的多维存储结构对这些 OLAP 操作处理的效率就很差.为了解决上面提到的这些问题,我们提出一种改进的多维存储结构,其基本思想是以多维数组结构为基础,同时结合关系存储方式的优点来存储组织数据仓库的多维数据.我们在设计多维存储结构时充分考虑了实际多维数组的大容量和高稀疏度等问题,对各种 OLAP 查询(区域查询,聚集查询等)都进行了优化,特别是通过在多维存储结构中存储各维内部的层次信息,显著提高了涉及维内部层次的聚集查询的响应速度.

1 改进的多维存储结构

1.1 多维数组的划分

由于各种 OLAP 操作通常都需要处理整个多维数组的数据,因此为提高效率,通常都是将整个数组读入内存后再进行各种计算处理,否则将导致频繁的磁盘重复读写.实际应用中,当多维数据集中的记录数目很大时,即使采用压缩数组方式进行存储,整个数组需要占用的存储空间仍然很大,经常会超出系统可用内存空间的大小.因此需要首先对容量过大的多维数据集进行划分,然后分别存储较小多维数据集对应的多维数组,从而保证较小多维数组占用的存储空间小于系统可用内存空间.文献[7]就采用了首先对多维数据集进行划分,然后分块存储对应多维数组的方法以有效地存储组织多维数组.对多维数据集合 $R(D_1,D_2,\ldots,D_n,F_1,F_2,\ldots,F_m)$进行划分时,首先分别对 n 个维 $D_1,D_2,\ldots,D_n$ 按其值域进行划分,如可以将第 i 维的取值空间 D_i 划分成 P_i 个互不相交的子空间,那么这 n 个维不同的子空间交叉组合将把原来的多维数据集合划分成Πp_i 个互不相交的子多维数据集合;然后按照多维数组方式存储这Πp_i 个子多维数据集合.

我们注意到划分完成后,即使保证每个多维子空间大小基本相同,但由于数据偏斜的存在,这Πp_i 个子多维数据集合对应的多维数组的大小(记录数目)也不一定相等,特别是当实际数据偏斜比较严重时,各个数组的大小其差别会很大.因此,为了有效地存储组织划分后形成的多维数组,提高磁盘 I/O 操作的效率(除此之外,还有其他的好处,如支持多维数组的增量更新等),我们考虑对文献[7]给出的划分方法进行扩展,将原来的划分过程称为逻辑划分阶段,然后再增加额外的物理划分阶段.也就是说,在对原多维数据集合进行划分得到Πp_i 个互不相交的子多维数据集合后,我们不将子数据集合对应的多维数组(称为逻辑块)作为一个整体直接存储到磁盘上,而是将逻辑块再次划分成若干个具有固定大小的物理块.这样一个逻辑块在物理存储上将对应若干个物理块(除最后一个物理块外,前面的物理块都将存满数组记录).磁盘存取的最小单位将是物理块,而不是大小不固定的逻辑块.

实际存储时,我们需要存储逻辑块到物理块的对应表,相当于一个索引,另外还需要存储物理块的额外信息,如该物理块存储的数组记录数目、该物理块在数据文件中的物理地址、最大和最小 S 值(此处指一般化的 S 定义,只需要 S 中含有各维的位置信息)等.多维数组先经过逻辑划分和物理划分,然后再存储到磁盘上,能够显

著提高磁盘的存取效率,特别是对于某些形式的查询,只需要少量的磁盘 I/O 操作.

1.2 多维数组的压缩存储

对于实际的多维数组,通常都非常稀疏,因此为了减少存储空间的浪费,需要对数组进行压缩存储.目前使用较多的是文献[5]采用的压缩存储方法,该方法只存储非空值(也就是事实属性值不全为 0)数组记录,对于每条数组记录,都存成$\langle S,F_1,F_2,\ldots,F_m\rangle$的形式,$S$ 表示该数组记录在多维数组线性化后的线性偏移(此处指一维线性化后的数组坐标),$F_1,\ldots,F_m$ 表示 m 个事实属性.

假设多维数据空间为$(D_1,D_2,\ldots,D_n,F_1,F_2,\ldots,F_m)$,第 i 个维的取值数目为 D_i 个,那么数组记录$\langle d_1,d_2,\ldots,d_n,f_1,f_2,\ldots,f_m\rangle$将存成$\langle S,f_1,f_2,\ldots,f_m\rangle$的形式.其中 $S=(\ldots(d_1*D_2+d_2)*D_3+\ldots)*D_{n-1}+d_n$,也就是相当于将 n 维坐标$\langle d_1,d_2,\ldots,d_n\rangle$使用线性化函数线性化为一维坐标.

在 OLAP 查询处理过程中,在得到数组记录后,还需要根据反线性化函数,从 S 反算出 n 维坐标$\langle d_1,d_2,\ldots,d_n\rangle$,其中 $d_n=S/D_{n-1}$,$d_i=(S-d_{i+1}*D_i)/D_{i-1}$,$i=n-1,n-2,\ldots,1$.

从上面的线性化和反线性化计算过程我们可以知道,在线性化时,对于一条数组记录,需要 $n-1$ 次乘法和加法运算,反线性化时需要 n 次除法,以及 $n-1$ 次乘法和减法运算,CPU 计算的开销还是比较大的,特别是对于反线性化计算.

我们知道,S 在数组记录中的作用就是存储各维的坐标信息,因此也可以考虑使用其他方式合成 S 值,使其包含各维坐标信息,并且 CPU 计算的开销比较小.文献[8]就是将各维坐标使用位拼接的方式组合得到 S,相比存储多维数组线性偏移的方式,计算开销要小.假设多维数据空间为$(D_1,D_2,\ldots,D_n,F_1,F_2,\ldots,F_m)$,并且第 i 个维的取值数目对应的二进制位数为 $DBit_i$.那么生成的数组记录$\langle S,f_1,f_2,\ldots,f_m\rangle$中的 $S=(\ldots(d_1<<DBit_2|d_2)<<DBit_3|\ldots)<<DBit_{n-1}|d_n$.从 S 反算各维坐标时,除了需要第 i 维的二进制位数为 $DBit_i$ 外,还需要第 i 维的位掩码 $Dmask_i$,它可以由 $2\uparrow DBit_i-1$ 计算得到;$d_n=S\ \&\ DMask_n$,$d_i=(\ldots(S>>DBit_n)\ldots)>>Dbit_{i+1}\&Dmask_i$.从上面的计算 S 以及由 S 反算各维坐标的方法可以知道,对于一条数组记录,计算 S 时需要 $n-1$ 次按位左移运算和按位或运算,由 S 反算各维坐标时需要 n 次按位与运算和 $n-1$ 次按位右移运算.显然,使用这种方式与存储数组记录在多维数组中的线性偏移的方式相比,CPU 计算上的开销要小得多.

在上面介绍的两种数组压缩存储方式中,S 中仅包含或者说只能从 S 直接计算得到数组记录在各个维的坐标信息,据此可以方便地在各个维之上进行分组聚集计算.但是对于实际的多位数据,维内部通常还具有层次,如时间维就具有年、月、日 3 个层次,并且分组聚集查询通常都会涉及维内部的层次,而不仅仅是在各维上进行聚集.例如,假设 D 对应时间维,由(A,B,C,D,F)聚集计算得到(A,B,C,D_{Year},F),就是仅在时间维的内部层次上进行聚集计算.这种情况下,前面介绍的两种数组压缩方法效率就不是很高了.

因此,为了有效地处理涉及维内部层次的聚集查询,我们考虑在多维数组的压缩存储时,额外存储维内部的层次信息.也就是对于数组记录$\langle d_1,d_2,\ldots,d_n,f_1,f_2,\ldots,f_m\rangle$,仍存成$\langle S,f_1,f_2,\ldots,f_m\rangle$的形式,但此时 S 中还额外地包含维内部的层次信息.在这里我们仍使用与前面压缩方式相同的位拼接方法,具体计算方法如下.

首先我们假设第 i 维有 k 个层次,分别为 $DL_{i1},DL_{i2},\ldots,DL_{ik}$,其中 DL_{i1} 为最高层次,并且各层次的最大取值数目也分别用 $DL_{i1},DL_{i2},\ldots,DL_{ik}$ 来表示,对应能够表示出这些取值数目的最小二进制位数分别为 $DLBit_{i1},DLBit_{i2},\ldots,DLBit_{ik}$.对于最高层次,最大取值数目就是指该层有多少个不同取值;对于其他较低层次,最大取值数目是指该层中对应于相同的较高层次其不同取值数目中的最大者.例如对于时间维,共有年、月、日 3 个不同层次,显然对于最高层次“年”来说,最大取值数目就是多维数组中年的不同取值数目;对于层次“月”来说,由于一年最多有 12 个月,那么该层最大取值数目就是 12;同样对于层次“日”来说,其最大取值数目是 31.

然后对于数组记录$\langle d_1,d_2,\ldots,d_n,f_1,f_2,\ldots,f_m\rangle$,我们将其在第 i 维的坐标记为 d_i,而在层次 $DL_{i1},DL_{i2},\ldots,DL_{ik}$ 的对应坐标分别记为 $dl_{i1},dl_{i2},\ldots,dl_{ik}$.那么

第 i 维的总位数 $DLBit_i=DLBit_{i1}+DLBit_{i2}+\ldots+DLBit_{ik}$;

第 i 维的位掩码 $DLMask_i=2\uparrow DLBit_i-1$,$\uparrow$ 表示指数幂运算;

$$dl_i=(\ldots(dl_{i1}<<DLBit_{i2}|dl_{i2})<<DBit_{i3}|\ldots)<<DBit_{ik-1}|dl_{ik},$$

最后我们可以使用位拼接的方式由 dl_i 和 $DLBit_i$ 计算得到 S

$$S=(\ldots(dl_1<<DLBit_2|dl_2)<<DLBit_3|\ldots)<<DLBit_{n-1}|dl_n.$$

这样最终计算出来的 S 将包含维内部的层次信息,根据 S 反算 dl_i 的过程与前面介绍的类似,全部计算过程只涉及位运算,具有较高的效率.

使用位拼接方式,将各维内部的层次信息组合而得到的 S 值,有可能会使 S 超出 32 位整数的取值范围.为解决此问题,可以将这种压缩方法和第 1.1 节介绍的数组分布存储技术结合起来,也就是说,表示各维以及各维内部层次信息的位数目的选择全部针对划分后的子多维数据空间,并且在特殊情况下,如某维的某层次取值只有一个时,可以通过额外记录这个值,而不用再在 S 值中为其分配存储位,此时我们称得到的 S 为局部 S 值.通过这种方式能够显著减少存储 S 值所需的位数目,使其在一般情况下都会小于 32 位,这样对 S 进行处理时都是整数位运算,而不会增加额外的开销.但是需要注意的是,如果采用这种方法,必要时还需要进行局部 S 值到全局 S 值的变换.

1.3 多维数组的初始建立

前面两节我们介绍了用于数据仓储的改进的多维存储结构中的两项关键技术:数组的划分以及数组压缩存储方法.下面我们给出初始创建多维存储结构的具体步骤,并进一步讨论数组创建过程中的其他技术问题.首先我们假设在创建多维数组之前,已经知道数组的相关信息,包括数组各个维的全部信息、事实属性的数目、数组记录的大致数目等.然后创建数组时主要按照下面的几个步骤进行:

(1) 根据已知的数组信息,对多维数组进行逻辑划分,并存储各逻辑块的划分信息,包括每个逻辑块的各维取值范围以及根据数组压缩方法计算出的该逻辑块所有记录的 S 的最大和最小可能取值,此外还需要将数组各个维的信息按照类似雪花模式的结构存储到磁盘上;

(2) 顺序读所有数组记录,根据逻辑划分条件,对数组进行划分,然后根据选定的数组压缩方式(参见第 1.2 节),计算得到每条数组记录的 S 值(各维位置信息值),形成压缩数组记录$\langle S,F_1,F_2,\ldots,F_m\rangle$,并根据需要将各划分块暂存到磁盘上;

(3) 顺序读入每个划分块的所有记录,按照 S 值对记录进行排序,排序后的记录再进行物理划分,并存储到磁盘上,同时记录逻辑块和物理块的对应关系以及相关信息,如每个物理块的最大、最小 S 值.

在创建数组的过程中一个比较关键的问题是,如何对多维数组进行逻辑划分.我们在进行逻辑划分时的处理步骤是这样的:首先根据数组总的记录数目 n,压缩数组记录的大小 r 以及系统可用内存空间 m,并依据同时在内存中处理一个逻辑块的所有数组记录的约束条件,利用记录公式 $L=n*r/m/2$(多除以 2 是为了保证即使在数据偏斜的情况下,多数逻辑块也满足约束条件),来确定逻辑划分块的数目 L;然后根据逻辑块的数目 L,(为简单起见)在假设各个维的划分块数相等的前提下,利用 LD 的 n 次幂等于 L 来确定各个维的划分区间数 LD;最后对各个维的取值区间进行划分,为便于处理涉及维内部层次的聚集查询,我们在进行划分时,先判断直接对维的最高层次取值区间进行划分是否就能完成划分(也就是判断划分区间数 LD 是否小于该层取值数目),若不能,再去判断较低层次的取值空间.另外,在划分时,对任一逻辑划分块,假设其在某个维的划分层次为 L_i,我们还要求该逻辑块的所有记录的 S 值对应在层次 $L_j(j<i$,也就是 L_j 的层次比 L_i 高)的取值要相同.上面的必要条件能够减少各逻辑块对应多维子空间的交叉程度,从而在 OLAP 查询处理时通过减少读取逻辑块的数目来提高效率.

2 性能分析与评价

2.1 区域查询和点查询

数据仓库中的多维数据按照多维结构进行存储后,将能够有效地支持各种查询.首先我们来讨论如何基于多维存储结构有效地处理各种区域查询和点查询.

对于给定区域条件的区域查询,查询处理时首先基于区域条件、各维信息以及多维数组的划分方式,确定满足区域条件的逻辑块集合;然后按照顺序,对于满足条件的逻辑块集合中的每个逻辑块,根据逻辑块和物理块对应信息以及每个物理块中记录的最大、最小 S 值,判断是否需要存取相应物理块;最后对于需要存取的物理

块,将其读入内存,根据每条记录的 S 值计算出对应各维的取值,并判断该记录是否满足区域条件.

对于点查询,查询处理时首先根据点查询条件和多维数组的划分条件,确定该点所属的逻辑块;然后根据该点的各维取值,计算出 S 值,再根据逻辑块和物理块对应信息,以及每个物理块中记录的最大、最小 S 值,确定该点所处的物理块;最后读入相应的物理块,在内存中二分查找得到满足点查询条件的记录的度量数据,完成查询.

根据上面的区域查询和点查询的处理过程我们可以知道,查询处理时只存储满足条件的逻辑块或者物理块,能够显著减少磁盘存储的开销,特别是对于点查询,在各维信息已事先读入内存的前提下,可以只需要存取一个物理块数据,也就是只需要 1 次磁盘 I/O 操作,磁盘开销降到了最低.

2.2 聚集查询

对于多维数据分析或者 OLAP 查询处理,聚集查询是其中非常重要的一类查询.下面我们来讨论如何基于多维存储结构来有效地处理各种聚集查询.

文献[6]给出了在压缩数组上的高效聚集算法,该算法直接在压缩数据上进行聚集计算,具有较高的效率,并且支持多种数组压缩方法,具有较好的通用性,对于本文提出的数组压缩方法,也可以直接使用文献[6]给出的聚集算法.但是由于我们设计的多维结构除采用数组压缩技术外,还使用了分布存储技术,这与文献[6]针对的数组存储结构有所不同.此外,文献[6]中也没有考虑(实际应用中经常出现的)涉及维内部层次的聚集查询,因此在这里我们需要进一步讨论如何在新的多维存储结构的基础上高效地实现各种聚集查询.

在第 1.2 节讨论数组的压缩存储时,我们已经给出了存储维内部层次的压缩存储方法,也就是说,压缩数组记录中的 S 值将包含维内部的层次信息.根据这一点,我们现在给出涉及维内部层次的聚集查询的处理步骤.

(1) 首先根据聚集计算要求得到的各维的层次,计算出从 S 中取出要聚集到的维层次信息所需的位掩码 *BitMask*.假设原多维数据集合为(D_1,D_2,D_3,F),维 D_1 有层次 L_{11},L_{12},L_{13}(L_{11} 层次高于 L_{12}),维 D_2 有层次 L_{21},L_{22},维 D_3 有层次 L_{31},L_{32},则压缩记录的 S 值形式为 $L_{11}L_{12}L_{13}L_{21}L_{22}L_{31}L_{32}$,采用位拼接的方式形成.当要求聚集得到($L_{12}$,$L_{21}$,$L_{31}$,$F$)时,我们可以首先确定需要取出的层次应为 L_{11},L_{12},L_{21},L_{31},然后将需要取出的对应层次信息的位掩码通过按位或运算得到 *BitMask*.一般来说,当 D_i 维聚集到的层次为 L_{ij} 时,则该维需要取出所有形如 L_{ik}($k<=j$,k 的层次不低于 j 的层次)的层次信息,例如聚集到的层次为 L_{12},则需要取出的层次除 L_{12} 外,还有 L_{11};

(2) 获得各逻辑块的最小 S 值,将 S 值和 *BitMask* 进行按位与运算,得到值 *SL*,然后按照各逻辑块的 *SL* 值对所有逻辑块进行排序.此处假设分组聚集的维序和多维数组生成时的维序相同,否则还需要对 *SL* 进行额外的变换(变换是指调整各维信息在 *SL* 值中的位次序),以使得维序一致.

(3) 按照排序后的逻辑块顺序,读相应逻辑块对应的所有物理块,然后将每条压缩记录的 S 值和 *BitMask* 值进行按位与运算得到值 *SL*,最后基于值 *SL* 对所有数组记录进行分组聚集计算.

在上面的步骤(3)中,我们只是简单地说按照计算出来的 *SL* 值对所有数组记录进行聚集计算,但实际上聚集计算过程还是比较复杂的.通用的聚集计算过程应该是:首先顺序对于每个逻辑块,按照计算出来的 *SL* 值在内存中对逻辑块的所有记录进行排序(在多维数组进行划分时已确保在多数情况下内存中可以存储一个逻辑块的所有记录);然后按照已经排序后的 *SL* 值进行分组聚集计算,并将该逻辑块的分组聚集计算结果写入磁盘;顺序处理完所有逻辑块后,再将所有逻辑块分组聚集计算的结果归并得到最终的聚集计算结果.由于共享排序的存在,在排序和归并过程中,都可以使用文献[6]中的优化策略来提高查询处理的性能,但是对于涉及维内部层次的聚集计算,无法直接使用文献[6]中的优化方法,我们将另文讨论.这里我们只讨论最简单的一种情况,也就是聚集计算的结果分组可以全部存储于内存中,即结果分组的数目较少,此时上述步骤 3 中的聚集处理算法如下:

```
GroupBuf 为存储结果分组的缓冲区,结果分组按照 SL 值有序
While 顺序取排序后的每个逻辑块:
        LastSL=NULL;/*上一条记录的 SL 值*/
    While 顺序读逻辑块对应的每个物理块,每条压缩记录〈S,F〉
        SL=S & BitMask;
        if SL!=LastSL
            根据 SL 值查找 GroupBuf 得到相应分组的分组序号 GroupNO;
```

```
                若未找到,需要在缓冲区中分配新的结果分组,保证结果分组序列有序,并得到相应的分组序号 GroupNO;
                LastGroupNO=GroupNO;
                LastSL=SL;
            Else
                GroupNO=LastGroupNO;
            End if
            根据 GroupNO 得到当前记录对应的分组,进行聚集计算;
        End while
    End while
    While 顺序对于 GroupBuf 中的每个结果分组
            根据 SL 查找相应的维信息,得到对应各维取值,并输出结果;
    End while
```

算法描述中的 *LastSL* 和 *LastGroupNO* 记录上一条数组记录对应的 *SL* 值和结果分组序号,这样,当连续有若干条记录属于同一结果分组时(实际计算中这种情况出现的机率相当高),可以避免在 *GroupBuf* 中查找结果分组的开销,从而提高性能.我们还注意到,聚集计算过程中并不需要先按照 *SL* 值对属于同一逻辑块的所有记录进行排序,并且也不需要对不同逻辑块分组聚集的结果进行归并,从而使聚集计算的总开销较小.但是在采用这种聚集计算方法之前,需要首先使用我们在文献[4]中给出的方法,根据各维信息估计结果分组的数目,判断结果分组是否可以全部存储于内存中.

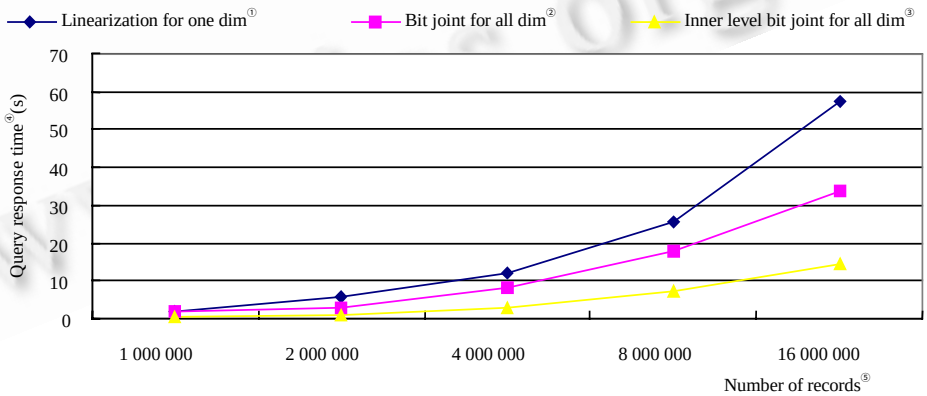
下面给出基于几种不同的数组压缩方法(见表 1 和图 1)进行聚集计算的实验结果.实验所用的环境为 Sun Ultra30,128M 内存;数组的记录数目从 1 000 000 变化到 16 000 000;表中的压缩方法 1,2,3 分别对应于第 1.2 节给出的 3 种数组压缩方法;聚集查询的结果分组数目较少,可以全部存储于内存中,并且是涉及维内部层次的聚集查询.对于压缩方法 1,2,由于压缩数组的 *S* 值中没有直接含有各维的层次信息,因此聚集计算时需要首先由 *S* 值计算出各维的坐标信息,然后通过内存中的维结构信息得到维内部的层次信息,并进行相应的分组聚集计算.

Table 1 Comparison of compression methods for multi-dimensional array

表 **1** 多维数组压缩方法的比较

Number of records①	First method②(s)	Second method③(s)	Third method④(s)
1 000 000	2.0	1.7	0.7
2 000 000	5.6	3.1	1.2
4 000 000	12.1	8.3	2.8
8 000 000	25.6	17.7	7.4
16 000 000	57.4	33.6	14.6

①数组记录数,②压缩方法 1,③压缩方法 2,④压缩方法 3.



①一维线性化,②各维位拼接,③各维层次位拼接,④聚集查询响应时间,⑤记录数目.

Fig.1 Comparison of aggregation query based-on three compression methods

图 1 3 种不同压缩方法聚集查询结果比较

从上面的实验结果数据可以看出,压缩方法 2 和压缩方法 1 都没有在 *S* 值中存储维内部的层次信息,但是压缩方法 2 的位拼接方式性能更优;采用压缩方法 3,在压缩记录中额外存储维内部的层次信息,与未存储层次信息的方法 2 相比,聚集查询处理的性能有极大的改善.

3 总 结

本文针对当前多维数组存储组织方式存在的一些问题,提出了一种改进的多维存储结构.在设计过程中,我们充分考虑了实际多维数组的大容量和高稀疏度等问题,提出了新的多维数组分布存储和压缩存储技术,使得基于该多维存储结构能够优化处理各种 OLAP 查询(区域查询,聚集查询等),特别是通过在多维存储结构中存储各维内部的层次信息,显著提高了涉及维内部层次的聚集查询的响应速度.在今后的研究工作中,我们还将进一步研究基于新的多维存储结构,如何实现完整的 OLAP 查询处理算法、Cube 计算方法、多维数组的增量更新算法,充分发挥新存储结构的优势,获得高性能的数据仓库和 OLAP 系统.

References:

[1] Chaudhuri, S., Dayal, U. An overview of data warehousing and OLAP technology. ACM SIGMOD Record, 1997,26(1):65~74.
[2] Colliat, G. OLAP, relational, and multidimensional database systems. SIGMOD Record, 1996,25(3):64~69.
[3] O'Neil, P., Quass, D. Improved query performance with variant indexes. ACM SIGMOD Record, 1997,26(2):38~49.
[4] Jiang, Xu-dong, Zhou, Li-zhu. A new aggregation algorithm for OLAP query evaluation. Journal of Software, 2002,13(1):65~70 (Chinese).
[5] Zhao, Yi-hong, Deshpande, P.M., Naughton, J.F. An array-based algorithm for simultaneous multidimensional. ACM SIGMOD Record, 1997,26(3):159~170.
[6] Li, Jian-zhong, Riotem, D., Srivastava, J. Aggregation algorithms for very large compressed data warehouses. In: Malcolm, P.A., ed. Proceedings of the 25th International Conference on Very Large Data Bases. 1999. 651~662.
[7] Sarawagi, S., Stonebraker, M. Efficient organization of large multidimensional arrays. In: Proceedings of the 10th International Conference on Data Engineering. 1994. 328~336.
[8] Goil, S. High performance on-line analytical processing and data mining on parallel computers [Ph.D. Thesis]. Northwestern University, 1999.

附中文参考文献:

[4] 蒋旭东,周立柱.联机分析查询处理中的一种聚集算法.软件学报,2002,13(1):65~70.

An Improved Multi-Dimensional Storage Structure for Data Warehousing*

FENG Jian-hua, JIANG Xu-dong, ZHOU Li-zhu

(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084, *China*)

E-mail: fengjh@tsinghua.edu.cn

http://www.cs.tsinghua.edu.cn

Abstract: As for physical data organization in data warehouse, there are mainly two kinds of methods, relational and multi-dimensional. These two methods have their own advantages and disadvantages, but as to improve the performance of OLAP (online analytical processing) query processing, the method of multi-dimensional array is superior. To solve the current problems in the method of multi-dimensional array, an improved multi-dimensional storage structure for data warehouse is proposed, and the concepts of logical storage and physical storage for data cube are given. According to this proposal, the original multi-dimensional data space is divided into many logical blocks, and a logical block is divided into many physical blocks. This multi-dimensional storage structure takes the characteristics of the large amount and highly sparse multi-dimensional array into consideration fully, and a new distributing and compressing method for the multi-dimensional array is adopted. These methods availably solve some efficiency problems of the aggregation query along with the inner level of the dimension and cube query, and dramatically improve the response time of the aggregation query. In particular, these methods also bring additional b9enefit for incremental maintenance of the multi-dimensional array.

Key words: data warehouse; multi-dimensional array; aggregation query; range query; OLAP (online analytical processing)

* Received March 7, 2001; accepted September 19, 2001
Supported by the National Grand Fundamental Research 973 Program of China under Grant No.G1998030414