ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.5, May 2006, pp.1213−1221
DOI: 10.1360/jos171213

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

静态物化视图的动态 Cache 优化算法*

张柏礼[1+], 孙志挥[1], 周晓云[1], 杨宜东[1], 朱玉全[2]

[1](东南大学 计算机科学与工程系,江苏 南京 210096)

[2](江苏大学 计算机科学与通信工程学院,江苏 镇江 212013)

A Dynamic Cache Optimized Algorithm of Static Materialized Views

ZHANG Bai-Li[1+], SUN Zhi-Hui[1], ZHOU Xiao-Yun[1], YANG Yi-Dong[1], ZHU Yu-Quan[2]

[1](Department of Computer Science and Engineering, Southeast University, Nanjing 210096, China)

[2](School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang 212013, China)

+ Corresponding author: Phn: +86-25-83111398, E-mail: bailey_zhang@163.com

Zhang BL, Sun ZH, Zhou XY, Yang YD, Zhu YQ Zhu YQ. A dynamic cache optimized algorithm of static materialized views. *Journal of Software*, 2006,17(5):1213−1221. http://www.jos.org.cn/1000-9825/17/1213.htm

Abstract: Because the static materialized views lack of better response performance for dynamic query, an optimized algorithm DCO (dynamic cache optimization) is proposed, which generates a dynamic materialized views set to cooperate with the existing static materialized view set by cache. With the assistance of the additional materialized views, the dynamic adaptability and response capability to query increase greatly. Meanwhile, a novel method of allocating the space is presented to provide the alternative for realizing the dynamic cache, and then the free space of system can be used efficiently to store more materialized views for improving the response capability. Experimental results indicate the efficiency and feasibility of DCO, and also show that DCO can overcome the SPSE (space-performance saturation effect) of materialized views in some degree.

Key words: data warehouse; materialized view; dynamic cache

摘 要: 针对静态物化视图集动态适应能力的不足,提出一种动态 cache 优化算法 DCO(dynamic cache optimization).它在保持静态算法获取最优物化集能力的基础上,将 cache 机制直观、快速的动态特性结合进来,以提高数据仓库的动态自适应性能.在 cache 机制具体实现中提出了一种新颖的空间申请方法,可以充分利用系统剩余空间提高查询响应性能.实验结果在表明算法有效、可行的同时,也显示出该算法可以在一定程度上克服静态物化集存在的空间-性能饱和效应(space-performance saturation effect,简称 SPSE),使通过增加物化空间进一步提高数据仓库对查询的响应速度成为可能.

关键词: 数据仓库;物化视图;动态 cache

中图法分类号: TP311 文献标识码: A

物化视图是指在数据仓库中将一部分查询视图预先进行计算并加以物理存储,这样可以有效地加快数据

* 本文为 2005 年中国计算机大会推荐优秀论文.Supported by the National Natural Science Foundation of China under Grant Nos.60572112, 70371015 (国家自然科学基金)

Received 2005-06-15; Accepted 2005-12-16

仓库对查询的响应速度.它已成为提高系统多维分析性能的重要手段.然而,一定约束条件下物化视图集的选择属于 NP-hard 问题[1],一直是数据仓库界的热点.为此,研究人员提出了一些解决方案,其中:Harinarayan 等人提出的 BPUS(benefit per unit space)算法是一种基于立方格的贪心算法[2];Gupta 则较为系统地阐述了物化视图选择问题,并提出了一系列启发式算法[1];文献[3]将物化视图选择问题转化为空间转移问题进行研究;文献[4,5]则将遗传算法获取最优解的能力用于物化集的选择,并在降低算法的复杂度方面进行了研究.

以上算法都基于查询分布预知的假设,本质上都可归结为静态算法.由于数据仓库的时变性,特别是决策支持应用中存在较大成分的即席访问,使得系统的查询模式难以预测,导致静态算法所选择的物化视图集逐步失去时效性.这意味着管理员需要及时发现查询模式的变化,选择适当的时机重新执行视图选择算法.对于一个具有复杂用户需求的大中型数据仓库而言,此工作是相当复杂而耗时的.一些研究人员通过对静态算法进行改造使其能够基本适应在线运行[6,7],从而可以定期地对物化集自动进行调整,在一定程度上减少了管理员的工作量.但是,这种方法需要基于一定的统计数据后才能进行.这表明在一定的统计周期内,对于查询分布的变化或是即席查询,物化视图集无法进行针对性的调整.为此,文献[8]在对静态算法改进的基础上提出即时调整的 FPUS(frequency per unit space)算法,具有一定的创新性.然而,该算法在每次查询后都要进行全体物化效益的比较,运行开销较大,特别是对于查询密度很高的情况不能适应.此外,部分视图可能出现频繁的“抖动”,使物化集缺乏稳定性,也将使很多经过优化的查询方案和优化路径不能重复利用,反而在一定程度上增加了查询开销,从而使该算法失去真正的实用价值.

为此,本文基于动态缓存(dynamic cache)技术,提出了 DCO(dynamic cache optimization)算法,包括基于空间约束的 DCO-S 算法和基于空间代价与维护代价双重约束的 DCO-SM 算法.需要明确的是:DCO 算法属于一种补充算法,是在静态算法获取物化集的基础上,利用动态 cache 机制实现额外一部分视图的动态物化,以适应查询分布的变化和即席查询的需要,使整个物化视图集具有较好的动态调整能力.同时,由于采用了不同的选择机理,动态物化视图集可在一定程度上克服文献[9]中所提到的随物化空间增加性能不再变化的现象,即本文所谓静态物化集存在的空间-性能饱和效应(space-performance saturation effect,简称 SPSE),从而在一个更大的区间内保证:增加物化空间可以使物化效益继续有较大的增长.另外,在 cache 的具体物理实现时,DCO 算法可以采用比较独特的空间申请方法,即借鉴操作系统中回收箱实现的机制,不占用数据库的有限数据区,而利用系统的剩余硬盘空间作为 cache 空间.并且,这种外挂式的技术使 DCO 算法不局限于某一种或某一类数据仓库的具体实现,具有很强的可移植性和跨平台性.

1 DCO 算法

1.1 问题描述

为了方便叙述,本文将利用静态算法选择的物化视图集称为静态物化集,用 M_S 表示;而采用 DCO 算法确定的物化视图集称为动态物化集,用 M_D 表示.下面首先给出基于物化视图集的查询代价定义.

定义 **1**. 若用户查询集 Q 中每个查询 q_i 的发生频度为 f_{q_i},以 $C_{q_i}(M)$ 表示由物化视图集 M 获取 q_i 的查询代价,则整个用户查询集 Q 的查询代价为 $C(Q,M)=\sum_{q_i\in Q} f_{q_i} * C_{q_i}(M)$.

① 若物化视图集 M 仅包括 M_S,即 $M=M_S$,则 $C(Q,M)=C(Q,M_S)=\sum_{q_i\in Q} f_{q_i} * C_{q_i}(M_S)$;

② 若物化视图集 M 包括 M_S 和 M_D,即 $M=M_S\cup M_D$,则

$$C(Q,M)=C(Q,M_S\cup M_D)=\sum_{q_i\in M_D} f_{q_i} * C_{q_i}(M_D)+\sum_{q_i\in Q-M_D} f_{q_i} * C_{q_i}(M_S).$$

由于动态物化集 M_D 的存在,一部分原来由 M_S 获得结果的查询 q_i 可以转而通过与 M_D 中的物化视图直接匹配而得到更快的响应,整个用户查询集 Q 的查询代价将产生以下变化:

$$\begin{aligned}\Delta C&=C(Q,M_S)-C(Q,M_S\cup M_D)\\&=\sum_{q_i\in M_D} f_{q_i} * C_{q_i}(M_S)+\sum_{q_i\in Q-M_D} f_{q_i} * C_{q_i}(M_S)-\sum_{q_i\in M_D} f_{q_i} * C_{q_i}(M_D)-\sum_{q_i\in Q-M_D} f_{q_i} * C_{q_i}(M_S)\end{aligned}$$

$$= \sum_{q_i \in M_D} f_{q_i} * (C_{q_i}(M_S) - C_{q_i}(M_D)).$$

定义 **2**. 若以查询 q_i 由物化集 M_S 得到查询结果的代价为 $C_{q_i}(M_S)$,而当 q_i 通过 cache 优化而获得动态物化后,其查询结果的代价变为 $C_{q_i}(M_D)$,则 q_i 由于 cache 优化机制而得到物化产生的效益为 $B(q_i)= C_{q_i}(M_S) - C_{q_i}(M_D)$.若用户查询集 Q 中每个查询 q_i 的发生频度为 f_{q_i},则整个查询集因物化集 M_D 获得的效益为

$$B(Q,M_S,M_D)= \sum_{q_i \in M_D} f_{q_i} * (C_{q_i}(M_S) - C_{q_i}(M_D)) = \sum_{q_i \in M_D} f_{q_i} * B(q_i).$$

现有的硬盘动态 cache 的效益模型[10−12]都是基于内存机制的修改,均未考虑物化视图从硬盘回读的代价,即 $C_{q_i}(M_D)$.所以相比之下,本模型较为严格,充分考虑到硬盘回读的 I/O 开销和视图检索代价;又因为未采用文献[2,7]中关于任一查询只与物化集中一个视图相关的假设,因而客观地反映了查询代价与自身对应视图的大小及其整个物化集大小的相关性.

定义 **3**. 若给定一个静态物化集 M_S 和用户查询集 Q,,则动态 cache 优化问题为在一定约束条件下选择一个物化集 M_D,使其产生的效益 $B(Q,M_S,M_D)= \sum_{q_i \in M_D} f_{q_i} * B(q_i)$ 最大.

求 $B(Q,M_S,M_D)$的最大值,往往受到空间、维护代价等条件的约束,为此,下面我们分别加以讨论.

1.2 基于空间约束的**DCO-S**算法

首先考虑基于 cache 空间约束下选择最优物化集 M_D 的情况,提出了 DCO-S 算法.为了说明算法的含义,需要引入查询的直接物化视图的概念.

定义 **4**. 如果查询 q_i 的结果在物化集中(包括 M_D 和 M_S)有直接对应的视图,而无须通过切块、切片或上卷得到,则该视图为这一查询的直接物化视图 *direct_view*(q_i).

算法 **1**. DCO-S 算法.

输入:M_S, q_x. /*q_x 为属于 Q 的任一查询*/

输出:M_D.

```
if  direct_view(q_x)∈M_D∪M_S  then  return;
For each q_i∈M_D
    Φ(q_i)= f_qi *B(q_i)/ s_qi ;   /* s_qi 为 q_i 查询结果/查询集大小,用记录数表示*/
    Sortby Φ(q_i,SignA);          /*将 q_i 的标识 id 和参函 Φ以及查询结果大小 s 输入数组 SignA,并按照 Φ非递减
                                    排列*/
    S_D=S_D+ s_qi ;               /*S_D 为 cache 中所有视图大小之和,初值置 0*/
S_free=S_cache–S_D;               /*S_cache 为 cache 的总空间大小,S_free 为当前 cache 的剩余空间*/
s_qx =sizeof(q_x);                /* s_qx 为等候进入 cache 的查询视图 q_x 的大小*/
pre_evit={∅};                     /*预删除视图集*/
count=0;
while ( s_qx >S_free)             /*可用空间不足以保存该视图*/
{
    count++;
    q_j=SignA[count].id;
    pre_evit=pre_evit∪{q_j};      /*按照Φ的顺序,逐个将视图置为预删除,直至满足空间要求*/
    S_free=S_free+SignA[count].s;
}
Φ(q_x)= f_qs *B(q_x)/ s_qx
Φ(pre_evit)=max({Φ(q_j)|q_j∈pre_evit});   /*获得预删除集中具有最大Φ值的视图*/
if  Φ(pre_evit)<Φ(q_x)  then  /*保证 q_x 是最优的*/
    Del pre_evit from M_D;
```

Insert q_x into M_D

Return.

下面讨论该算法的解的最优性问题(本算法与文献[10]除了模型不同以外,算法上也有很大的区别,正是这些差异才能保证 DSO 算法的最优性).在 cache 空间约束下,使 M_D 物化效益最优的问题,可以用以下优化函数来表述:

$$\mathrm{Max}\left(\sum_{q_i\in M_D} f_{q_i} * B(q_i)\right) \tag{1}$$

$$\sum_{q_i\in M_D} s_{q_i} \le S_{cache} \tag{2}$$

由式(1)、式(2)定义的问题属于 NP-complete 的背包问题[10].这类问题没有有效算法,但如果假设 cache 的大小 S_{cache} 相对于单个视图的大小 s_{q_i} 足够大,则可以近似地认为 S_{cache} 可被很小的 s_{q_i} 填满.这样,式(2)可修改为

$$\sum_{q_i\in M_D} s_{q_i} = S_{cache} \tag{3}$$

对于由式(1)、式(3)定义的问题,可以证明算法 DCO-S 可获得最优解.

定理 **1**. DCO-S 算法获得的物化集是由式(1)、式(3)所定义问题的最优解.

证明:由 DCO-S 可知,算法按照 $\Phi(q_i)$也就是 $f_{q_i} * B(q_i)/ s_{q_i}$ 逆序逐个保存 q_i,直至充满空间 S_{cache},其最终选择的视图具有以下特征:

$$\sum_{q_i\in M_D} \frac{f_{q_i} * B(q_i)}{s_{q_i}} \ge \sum_{q_i\in K} \frac{f_{q_i} * B(q_i)}{s_{q_i}},\ K\neq M_D \tag{4}$$

也就是在 $\sum s_{q_i} = S_{cache}$ 的约束条件下,由 DCO-S 算法所选择的视图的 $\Phi(q_i)$之和具有最大值,大于其他任何可能的组合.

设 $M_D\cap K=\varnothing$,如果不成立,可以从 M_D 和 K 中同时消去其相交元素而保持不等式成立,即可得

$$\sum_{q_i\in I} \frac{f_{q_i} * B(q_i)}{s_{q_i}} \ge \sum_{q_i\in J} \frac{f_{q_i} * B(q_i)}{s_{q_i}},\ I\cap J=\varnothing \tag{5}$$

则表明 I 中任意元素的 Φ都比 J 中的要大,即 $\min(\{\Phi(q_i)|q_i\in I\})\ge\max(\{\Phi(q_i)|q_i\in J\})$.

设 $\Phi(q_k)=\min(\{\Phi(q_i)|q_i\in I\})$;$\Phi(q_l)=\max(\{\Phi(q_i)|q_i\in J\})$;

$$\frac{\sum_{q_i\in I} f_{q_i} * B(q_i)}{\sum_{q_i\in I} s_{q_i}} = \frac{\sum_{q_i\in I} \Phi(q_i) * s_{q_i}}{\sum_{q_i\in I} s_{q_i}} \ge \frac{\sum_{q_i\in I} \Phi(q_k) * s_{q_i}}{\sum_{q_i\in I} s_{q_i}} = \Phi(q_k),$$

$$\frac{\sum_{q_i\in J} f_{q_i} * B(q_i)}{\sum_{q_i\in J} s_{q_i}} = \frac{\sum_{q_i\in J} \Phi(q_i) * s_{q_i}}{\sum_{q_i\in J} s_{q_i}} \le \frac{\sum_{q_i\in J} \Phi(q_l) * s_{q_i}}{\sum_{q_i\in J} s_{q_i}} = \Phi(q_l),$$

则 $\frac{\sum_{q_i\in I} f_{q_i} * B(q_i)}{\sum_{q_i\in I} s_{q_i}} \ge \frac{\sum_{q_i\in J} f_{q_i} * B(q_i)}{\sum_{q_i\in I} s_{q_i}}$,即 $\sum_{q_i\in I} f_{q_i} * B(q_i) \ge \sum_{q_i\in J} f_{q_i} * B(q_i)$.

进而可得 $\sum_{q_i\in M_D} f_{q_i} * B(q_i) \ge \sum_{q_i\in K} f_{q_i} * B(q_i)$,其中 $K\neq M_D$.

因此,由 DCO-S 算法选取的物化集满足式(1)、式(3),证毕. □

对于实时频度 f_{q_i} 的测算,文献[10,12]中借鉴了 LRU-K 机制[13],计算中考虑该查询最近的 K 次引用(一般情况下,$1\le K\le 5$),避免了采用最近单次引用进行估算的不足,相对而言是一个表达即时频度很好的方法,但它忽略了查询长期分布的特征.对此,文献[12]在此基础上加入该查询发生的总次数,可看作是对长期累计值的考虑.其不足之处在于,为了抵消长期积累的过分影响而设置了一个较长的累计周期,在此周期中未发生的查询,其累积

值被清零.这一处理方法显得有些粗糙,会导致频度值的跳变.为此本文进行了修正,提出一种利用衰减因子进行逐步回零的算法 AIF(attenuator of instantaneous frequency),即每隔一个(较短)周期,若该查询没有发生,则其查询的累计次数按一定的衰减系数进行递减,避免了频度值的跳变引起物化集较大的抖动.具体算法如下:

算法 **2**. AIF 算法.

输入:q_x.

输出: f_{q_x}.

```
while (any query q_x occur)
{
    query_count_sum++;   /*查询发生的总次数*/
    q_x.ref_count++;   /*q_x 的引用次数累计*/
    q_x.ref_flag=USED;
    q_x.ts[ref_count%K]=query_count_sum;   /*时标:记录 q_j 发生时刻,这里用查询发生的总次数作为顺序号
                                             来表示*/
                                           /*设数组从 1 开始,K 表示用于计算实时频度的查询最近引用
                                             次数*/
    if q_x.ref_count>K then
        f_{q_x} =ref_count*K/(query_count_sum−q_x.ts[(ref_count+K)%K]);
    else
        f_{q_x} =ref_count/(query_count_sum−q_x.ts[1]);
    If query_count_sum%Time=0 then        /*Time 为一个较短的统计周期*/
        For all q_j∈Q
            If q_j.ref_flag=UNUESD then   /*q_j 在本周期未有发生*/
                If q_j.ref_count>K then   /*ξ衰减因子*/
                    q_j.ref_count=q_j.ref_count%K+K*(q_j.ref_count/K/ξ);
                else q_j.ref_count=0;
            q_j.ref_flag=UNUESD;
}
return.
```

本节最后讨论一下 cache 置换测度值 $\Phi(q_i)= f_{q_i} *B(q_i)/ s_{x_i} = f_{q_i} *(C_{q_i}(M_S) - C_{q_i}(M_D))/ s_{x_i}$.其实,整个算法实现主要就是围绕测度值$\Phi(q_i)$的计算和比较进行的,这是所有 cache 算法的共同特征,有所不同的是具体的测度值定义.相对于 WATCHMAN[10],Dynamat[11]等这些从内存机制中简单移植过来的算法,DCO 算法没有对物化视图由硬盘读出的代价,即 $C_{q_i}(M_D)$ 予以忽略,相对而言比较严谨.因为对于硬盘 cache 机制, $C_{q_i}(M_D)$ 的大小有可能与 $C_{q_i}(M_S)$ 相差不大,不能简单地予以忽略,否则会造成较大的误差,从而影响最优物化对象的选择.

1.3 DCO-SM算法

实际中,物化视图选择在受到空间条件约束的同时,往往还受到视图维护代价的约束,多重约束条件下的视图选择是一个很复杂的问题.为此,结合本文的算法特点对该问题进行一定程度的简化,分以下几种情况加以讨论:(1) 物化视图维护窗口很大,M_S,M_D 都可以得到同步的更新维护,维护代价不成为约束条件,只要考虑空间约束即可;(2) 视图维护窗口较大,保证 M_S 得到维护的同时,对 M_D 可以实现部分维护,这时,该问题转化为维护代价约束下的动态物化视图选择问题,仿照空间约束可以进行类似的讨论(这也极具特殊性,因篇幅所限,另文加以研究);(3) 视图维护窗口较小,仅能保证 M_S 得到维护.情况(3)具有一定的普遍性和实用性,为此,本文进行较详细的讨论,提出了一种考虑多约束条件的 DCO-SM 算法.该算法基于这样一种处理方法:对于物化集 M_D 不考虑同

步更新,一旦出现 M_D 中的视图因为数据一致性问题而要求更新时,只是简单地将其从 cache 中淘汰,这种处理方式使算法变得简洁而实用.

算法 **3**. DCO-SM 算法.

输入:M_S, q_x.

输出:M_D.

if $direct\_view(q_x)\in M_D\cup M_S$ then return;

For each $q_i\in M_D$

$\Phi(q_i)= f_{q_i}(1- f_{u_i})\cdot B(q_i)/ s_{q_i}$; /*采用了算法 1 不同的测度值*/

$Sortby\Phi(q_i,SignA)$; /*按照 Φ 值非递减排列*/

$S_D=S_D+ s_{q_i}$;

$S_{free}=S_{cache}-S_D$;

$s_{q_x}=sizeof(q_x)$;

$pre\_evit=\{\varnothing\}$;

$count$=0;

while ($s_{q_x}>S_{free}$)

{

$count$++;

$q_j=SignA[count].id$;

$pre\_evit=pre\_evit\cup\{q_j\}$;

$S_{free}=S_{free}+SignA[count].s$;

}

$\Phi(q_x)= f_{q_x}*(1- f_{u_x})*B(q_x)/ s_{q_x}$ /*计算考虑了更新淘汰影响的 q_x 的测度值*/

$\Phi(pre\_evit)=\max(\{\Phi(q_j)|q_j\in pre\_evit\})$; /*获得预删除集中具有最大 Φ 值的视图*/

if $\Phi(pre\_evit)<\Phi(q_x)$

Del pre_evit from M_D;

Insert q_x into M_D

Return.

算法 3 与算法 1 没有本质上的区别.但算法 3 考虑到由于物化视图更新的存在造成了动态物化视图的效益下降,其表现为测度值计算中需要考虑物化视图的更新频度,也就是该视图因需要更新而被淘汰的概率,具体更改为 $\Phi(q_i)= f_{q_i}*(1- f_{u_i})*B(q_i)/ s_{q_i}$,也就是增加了 $(1- f_{u_i})$ 项,其中 f_{u_i} 为查询 q_i 的更新频度.而算法 1 中的测度值为 $\Phi(q_i)= f_{q_i}*B(q_i)/ s_{x_i}$,它的前提是不考虑物化视图更新或者假设系统具有较强的更新维护能力,也就是本节开始部分提到的实际中不常见的情况(1),而当必须要考虑物化视图更新且情况(1)这种假设不存在时,将导致测度值的计算存在较大的误差.

对于该算法的最优性问题,同样可以近似用以下函数最优解来表述:

$$\mathrm{Max}\left(\sum_{q_i\in M_D} f_{q_i}*(1- f_{u_i})*B(q_i)\right) \tag{6}$$

$$\sum_{q_i\in M_D} s_{q_i}=S_{cache} \tag{7}$$

对于由式(6)、式(7)定义的问题,可以证明算法 DCO-SM 可获得最优解.定理如下:

定理 **2**. DCO-SM 算法获得的物化集是由式(6)、式(7)所定义问题的最优解.

证明:可以仿照定理 1 进行. □

1.4 动态**cache**机制的物理实现

在具体物理实现 cache 机制时,DCO 建议采用一种较新颖的空间申请方法,即借鉴操作系统中回收箱的实

现机制,充分利用系统的剩余硬盘空间作为 cache 空间,可以不占用数据库的有限数据区,也避免了与数据库中其他数据冲突的可能.而且,这种外挂式的物化视图动态 cache 实现技术可以与数据仓库形成一种松散耦合,使其不受限于具体的数据仓库,具有很强的可移植性和跨平台性.

如果系统剩余空间比较紧张,DCO 算法必须定时检测系统可用空间,或截获当应用程序出现空间不足时的告警,动态地调整其可用的剩余空间,以做到既充分利用系统的自由空间又不会影响系统其他应用的运行.当可用空间缩小需要淘汰部分视图时,可以利用 DCO 算法,在新的约束空间下对 M_D 进行重新选择;或者采用简易淘汰算法,依据测度值 Φ 的大小非递减顺序地逐个淘汰,直至达到空间要求为止.具体淘汰算法如下:

算法 **4**. 简易淘汰算法.

$\Delta S=S_{cache}-S'_{cache}$; /* S'_{cache} 为缩小后的 cache 空间大小*/

$S_{evit}=0$;

while ($S_{evit}<\Delta S$)

{

  $count$++;

  $q_j=SignA[count].id$;

  $pre\_evit=pre\_evit\cup\{q_j\}$;

  $S_{evit}=S_{evit}+SignA[count].s$;

}

Del pre_evit from M_D;

Return.

相反地,如果系统的剩余空间变大,可以适时地扩大动态 cache 的空间,从而使更多的视图得到物化,提高系统的查询响应性能.

2 实验与性能比较

2.1 实验设计

为了验证动态 cache 优化算法的有效性,实验中将它与仅有静态物化集的情况进行了一系列性能的比较,即为系统在 M_S+M_D 支持下与在 M_S 单独支持下查询性能的比较.PBS(pick by size)算法[9]作为静态物化视图集的选择算法,是目前运行最快和应用最广的算法,而最新的一些静态算法,如进化算法[4,5],虽然在选择最优物化集方面具有优势,但运行时间复杂度过高,使其具有较高的理论价值而缺乏真正的实用性.所以,本实验中的静态物化集 M_S 都采用 PBS 算法进行选择,而动态物化集 M_D 由 DCO-S 动态选择(本实验中没有考虑更新问题).所以,实际上进行的是 PBS 算法与 PBS+DCO-S 算法的比较.

实验的硬件平台为 Dell PowerEdge 6600(双 Xeon CPU1.5G, RAM 1G),数据库平台为 Oracle 8.实验数据集为某药品销售公司交易数据库所对应的数据仓库,与文献[14]有相同的结构,共有 4 个维表和 1 个事实表,测试用事实表记录数为 1.6M.用于频度估算的 AIF 算法的 K 值取 3,衰减因子 ξ 取 2,采用查询的平均响应时间来衡量不同物化集对查询的系统响应性能.本实验中系统剩余空间为 5.3G,使得实际的空间约束并不存在.为了测试算法的有效性,实验中采用人为设定和改变 cache 空间大小两种方式.

2.2 实验步骤和结果

实验步骤如下:首先设定空间约束为 100M 时,利用 PBS 算法确定静态物化集 M_S 后,由模拟的查询发生器利用预制的 6 个查询模板各生成 5 000 次随机查询,然后计算其平均响应时间;随后,在此基础上启用 DCO-S 算法,其约束空间取为 20M,测试在 M_S 和 M_D 共同作用下,系统对不同模板查询集的平均响应时间.具体结果如图 1 所示,其中 $T1$ 表示采用 1 号预制模板作为查询发生器的查询生成模板.

由图 1 可见,由于 DCO-S 的作用即动态视图集 M_D 的参与,使系统查询性能得到了较大提升.同时,对不同模板生成的查询集的响应性能显得比较均衡,不同于单纯静态视图集 M_S 条件下系统对不同的查询集具有很大的

倾斜性.这是由于 DCO-S 算法的即时调整策略,使整个物化集具有较好的动态适应能力,可以及时地根据查询集的差异进行针对性的调整,以适应查询的变化.但是,这些变化或改进并不能排除空间增大因素的影响(即便是这增加的空间利用的是系统的剩余空间).为此,继续进行了以下实验:将 M_D 删除,DCO-S 算法停用,而在空间约束为 120M 条件下由 PBS 算法重新选择静态物化集 M_S,然后重新测试其对各查询集的响应时间,具体结果也在图 1 中表述.从结果来看,其性能提高方面与 PBS+DCO-S 没有太大的差距,也就是说,当可用总空间均为 120M 时,PBS+DCO-S 算法在对不同查询集的均衡性和跟踪适应力方面有较大的优势以外,对查询性能提升的优势并不明显.

然而,当实验将空间约束改为 300M 时,cache 空间保持不变,按相同的顺序重复以上 3 组实验,其结果如图 2 所示.可见,PBS(300)+DCO-S(20)相对于 PBS(320)或 PBS(300)都有较大的提升;而 PBS(320)相对于 PBS(300)基本没有变化.通过对静态物化视图空间-性能曲线[9]的分析,不难对此现象进行解释:当空间值较小时,该曲线斜率较陡,空间的变化可以引起性能的较大变化(相比之下,使得 DCO 算法的优势不明显);当空间值较大时,曲线进入饱和区,再增加静态物化空间对性能的提高没有太大的影响.这时,由于 DCO 算法采用了不同机理,在一定范围内不会受到空间饱和因素的影响,其优势将有较大的体现.

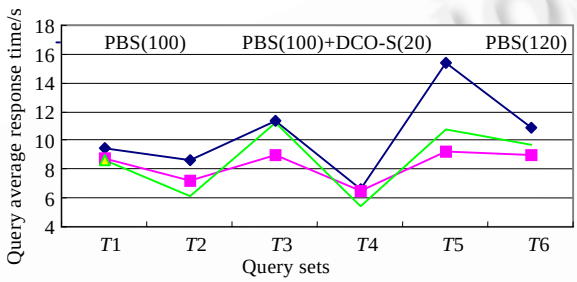

Fig.1 Comparison of query response time (1)

图 1 查询响应时间比较(1)

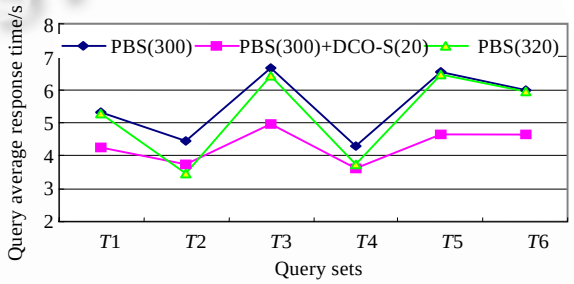

Fig.2 Comparison of query response time (2)

图 2 查询响应时间比较(2)

可见,经过 DCO 算法优化的物化集对于不同查询集都有较好的动态适应,而且无论静态物化视图是否达到饱和,对查询性能的提升都有一定的帮助.当静态物化视图基本达到饱和时,相对于同样的空间增幅,DCO 算法的性能优势能得到更好的体现.

另外,FPUS 算法[8]和 PMVS(preprocessor of materialized views selection)算法[7]虽然也可以对物化视图集进行动态性能改进,但从实际应用中看,不采用 cache 机制而使用其他改进算法对物化集在线调整是非常困难的:PMVS 算法仅能实现静态物化集周期性的动态调整,与 DCO 算法所进行的实时在线调整相差较大;而 FPUS 虽然从理论上讲可以进行实时在线调整,但每次查询发生后都要遍历整个视图空间进行效益的排序,相当于每次查询后都要运行一次静态选择算法 PBS,开销很大,这对于大数据集基本上是不可行的.

3 结束语

由于数据仓库的时变特点,导致传统物化视图选择算法确定的静态物化集对查询响应能力下降,对于即席查询的响应则更显薄弱.为此,本文提出了一种动态 cache 优化算法,作为静态物化视图选择的补充算法,可以对动态查询、即席查询提供有效的支持.同时,通过实验发现,由于结合了 cache 机制,可以在一定范围内有效地克服静态物化视图空间-性能的饱和效应 SPSE,使得通过增加物化空间进一步提高系统的查询响应性能成为可能.后续实验通过改变 cache 的大小,发现动态物化集同样也具有饱和效应.具体动态物化集对性能改进的有限性,是本文后续工作之一.

References:

[1] Gupta H. Selection of views to materialize in a data warehouse. In: Afrati FN, Kolaitis PG, eds. Proc. of the 6th ICDT. Heidelberg: Springer-Verlag, 1997. 98−112.

[2] Harinarayan V, Rajaraman A, Ullman JD. Implementing data cubes efficiently. In: Jagadish HV, Mumick IS, eds. Proc of the 1996 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 1996. 205−227.

[3] Theodoratos D, Sellis T. Designing data warehouse. Data & Knowledge Engineering, 1999,31(3):279−301.

[4] Zhang C, Yao X, Yang J. An evolutionary approach to materialized views selection in a data warehouse environment. IEEE Trans. on Systems, Man and Cybernetics, Part C, 2001,31(3):282−294.

[5] Horng JT, Chang YJ, Liu BJ. Applying evolutionary algorithms to materialized view selection in a data warehouse. Soft Computing, 2003,7(8):574−581.

[6] Theodoratos D, Sellis TK. Dynamic data warehouse design. In: Mohania MK, Tjoa AM, eds. Proc. of the 1st Int'l Conf. on Data Warehousing and Knowledge Discovery. London: Springer-Verlag, 1999. 1−10.

[7] Zhang BL, Sun ZH, Sun X. Preprocessor of materialized views selection. Journal of Computer Research and Development, 2004, 41(10):1645−1651 (in Chinese with English abstract).

[8] Tan HX, Zhou LX. Dynamic selection of materialized views of multi-dimensional data. Journal of Software, 2002,13(6): 1090−1096 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/1090.htm

[9] Shukla A, Deshpande PM, Naughton JF. Materialized view selection for multidimensional datasets. In: Gupta A, Shmueli O, Widom J, eds. Proc. of the 24th Int'l Conf. on VLDB. San Francisco: Morgan Kaufmann Publishers, 1998. 488−499.

[10] Scheuermann P, Shim J, Vingralek R. Watchman: A data warehouse intelligent cache manager. In: Vijayaraman TM, Buchmann AP, Mohan C, Sarda NL, eds. Proc. of the 22nd Int'l Conf. on VLDB. San Francisco: Morgan Kaufmann Publishers, 1996. 51−62.

[11] Kotidis Y, Roussopoulos N. DynaMat: A dynamic view management system for data warehouses. In: Delis A, Faloutsos C, Ghandeharizadeh S, eds. Proc. of the 1999 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 1999. 371−382.

[12] Otoo E, Olken F, Shoshani A. Disk cache replacement algorithm for storage resource managers in data grids. In: Proc. of the IEEE/ACM SC 2002 Conf. on Supercomputing. Los Alamitos: IEEE Computer Society, 2002. 1−15.

[13] O'Neil EJ, O'Neil PE, Weikum G. The LRU-K page replacement algorithm for database disk buffering. In: Buneman P, Jajodia S, eds. Proc. of the 1993 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 1993. 297−306.

[14] Kimball R. The Data Warehouse Toolkit. 2nd ed., New York: John Wiley & Son, 2002.

附中文参考文献:

[7] 张柏礼,孙志挥,孙翔.物化视图选择的预处理算法.计算机研究与发展,2004,41(10):1645−1651.

[8] 谭红星,周龙骧.多维数据实视图的动态选择.软件学报,2002,13(6):1090−1096. http://www.jos.org.cn/1000-9825/13/1090.htm

张柏礼(1970－),男,江苏盐城人,博士,工程师,主要研究领域为数据仓库,数据挖掘和 Web.

杨宜东(1979－),男,博士,主要研究领域数据挖掘,知识发现.

孙志挥(1941－),男,教授,博士生导师,主要研究领域为复杂系统信息集成,数据库系统及应用.

朱玉全(1966－),男,博士,副教授,主要研究领域为复杂信息系统集成,知识发现,入侵检测.

周晓云(1979－),男,博士生,主要研究领域数据挖掘,知识发现.