数据仓库联机维护中一致性问题的研究*

李子木 孙利民 周兴铭

(长沙工学院计算机科学系 长沙 410073)

E-mail: doctor2@nudt.edu.cn

摘要 数据仓库是存储供查询和决策分析用的集成化信息仓库,它的信息来源于不同地点的数据库或其他信息源.实体化视图是数据仓库中存储的主要信息实体,当原始数据发生变化时,数据仓库中的实体化视图也必须作相应的更新维护.在数据仓库实体化视图的联机维护过程中,由于联机分析处理(On-line Analytical Process,简称OLAP)查询的介入,会产生数据不一致的问题.文章提出了一种MVCA(multiversion compensating algorithm)算法来解决这一问题.MVCA采用版本控制方法,利用补偿思想和应答机制协调数据库和数据仓库之间的更新维护操作,达到保证数据一致的目的.最后,文章通过一个典型示例说明了该算法在实际中的具体应用.

关键词 数据仓库,实体化视图,联机维护,数据一致性,MVCA(multiversion compensating algorithm).

中图法分类号 TP311

数据仓库用来为决策提供支持,它的数据来源于数据库或其他信息源(本文只考虑数据库的情况).类似于早期的数据库,现有的数据仓库产品采用的都是脱机维护策略:系统先搜集记录,不立刻加入数据仓库,而是使用Bulk-load技术周期地刷新数据仓库.由于Bulk-load技术在系统刷新期间,数据仓库被禁止使用,所以,许多采用数据仓库产品的公司都是利用夜晚时间对数据仓库进行刷新维护.这种维护方式面临着3个重要的问题.

· 随着公司业务的全球化,时区问题会变得越来越明显,专门对系统进行更新维护的"夜晚时间"将越来越难以确定;

· 随着应用的不断深入,当数据量非常大的时候,这种维护方式所用的时间也会越来越长.由于Bulk-load技术在刷新系统期间,对数据仓库是禁止使用的,而系统的维护工作必须在次日清晨用户开始使用数据仓库之前完成,因此,时间是一个必须认真考虑的限制因素;

· Bulk-load技术容易使信息过时,在一些对实时性要求比较高的关键任务中,如战场决策,这种脱机维护方式是不能够被接受的.

因此,数据仓库联机维护技术受到了越来越广泛的重视.

1 问题的产生及研究现状

在数据仓库联机维护技术中,实体化视图的联机维护是一个关键技术.它是指,在数据仓库为用户提供服务的同时,当数据库中的原始数据发生改变时,实时地将这种变化反映到数据仓库中,使相应的实体化视图得到及时的刷新.

数据仓库中存储的是部分原始数据的拷贝和实体化视图集合,它并不能完全满足用户的所有查询.对于某

---

* 作者**李子木**,1971年生,博士生,主要研究领域为数据仓库,数据库协同工作,高速计算机网络.**孙利民**,1968年生,博士,主要研究领域为分布并行计算,高速计算机网络.**周兴铭**,1938年生,教授,博士生导师,中国科学院院士,主要研究领域为数据库协同工作,分布并行计算,高性能计算机体系结构.

本文通讯联系人:李子木,长沙 410073,长沙工学院计算机科学系

本文1998-05-05收到原稿,1998-09-01收到修改稿

些 OLAP 查询(如"下钻"),数据仓库必须通过访问数据库才能给出最终的查询结果.由于实体化视图是建立在原始数据之上的,它的更新维护总是或多或少地落后于原始数据的更新.当一个 OLAP 查询既用到实体化视图,又用到原始数据,而且原始数据和实体化视图是相关的(实体化视图用到了该原始数据),如果原始数据已被修改,而这种变化尚未反映在相应的实体化视图中时,便会产生查询上的不一致问题.

斯坦福大学的 D. Quass 提出了一个 2VNL(two-version no-locking)算法[1],利用版本控制来解决实体化视图的联机维护和一致性问题.2VNL 算法采用当前版本与影子版本分别负责实体化视图的查询与维护,当前版本只允许进行读操作,不允许更新,所有的写操作和更新操作全部在影子版本上进行.当影子版本提交后,当前版本被废除,影子版本成为当前版本,以后的维护操作将在新的影子版本上进行.2VNL 算法在解决数据仓库联机维护和一致性方面作出了有益的尝试,但它却存在 3 个主要的缺点.

• 由于算法只采用了两个版本,在进行版本切换时,有些查询事务将不可避免地被中止,从而导致事务的重启.特别是对 OLAP 这种费时较长的查询,事务重启的可能性就更大了;

• 2VNL 算法基于对查询情况的预测,只对某些汇总属性建立了影子版本.当这种预测不够准确、完备,或者对某些查询情况无法预测时,2VNL 算法将无法工作.而且由于这种预测需人工进行,这将增加维护的成本,并带来许多其他问题;

• 由于算法只采用了两个版本,而且总是在每天的一个固定时间进行切换,所以这种维护方式满足不了类似于"战场决策"这样的对实时性要求较高的关键任务.

本文提出的"MVCA 多版本补偿算法"(multiversion compensating algorithm,简称 MVCA),除了能够保证在联机维护状态下用户查询的一致性以外,还很好地解决了上述这些问题.MVCA 算法分为 MVCA-dw 和 MVCA-db 两部分,分别在 DW 端和 DB 端执行.它采用补偿思想[2]和应答机制,联合控制原始数据的多个版本,以保证在对数据仓库中实体化视图联机维护的同时,OLAP 查询也能得到一致的结果.

2 MVCA 算法

2.1 MVCA 中的消息类型

db_up:　DB 端发出的数据变化通知,它包括 3 个参数:

ins(r,t):在关系 r 中插入元组 t;

del(r,t):从关系 r 中删除元组 t;

mod(r,t_1,t_2):在关系 r 中用元组 t_2 更新元组 t_1;

dw_mv:　DW 端视图维护进程发出的查询通知,它返回的结果用于实体化视图的维护;

dw_olap:　DW 端发出的用户查询通知,它返回的结果将直接提交给 OLAP 应用;

db_mv:　DB 端发出的完成 *dw_mv* 查询后的返回结果;

db_olap:　DB 端发出的完成 *dw_olap* 查询后的返回结果;

dw_ack:　DW 端发出的维护完成确认通知,DB 端在收到这一消息后,将对数据库中的老版本元组和标志位作相应处理.

2.2 MVCA-dw 算法

MVCA-dw 算法是 DW 端视图维护进程执行的算法.它加强了 ECA 算法[3]对 modify 操作的支持,使实体化视图的维护更具完备性,并通过与 MVCA-db 建立的应答机制保证用户查询的一致性.具体算法如下.

初始化:$i=0$,$k=0$,$Collect=\varnothing$,$UQS=\varnothing$,$T_{op}=\varnothing$;

对 DB 端发来的消息作相应处理:

db_up (U):

If $U=ins(r,t)$ then

$\{i=i+1$;$Q_i=V(r,t)-\sum_{Q_j\in UQS}P_jQ_j(r,t)$;

发送 *dw_mv*(Q_i)到 DB 端;

$UQS \leftarrow UQS+Q_i; P_i=1; T_{op} \leftarrow T_{op}+\langle ins(r,t)\rangle;\}$

Else if $U=del(r,t)$ then

$\{i=i+1; Q_i=V(r,t)-\sum_{Q_j \in UQS} P_j Q_j(r,t);$

发送 $dw_mv(Q_i)$到 DB 端;

$UQS \leftarrow UQS+Q_i; P_i=-1; T_{op} \leftarrow T_{op}+\langle del(r,t)\rangle;\}$

Else if $U=mod(r,t_1,t_2)$ then

$\{i=i+1; Q_i=V(r,t_1)-\sum_{Q_j \in UQS} P_j Q_j(r,t_1);$

发送 $dw_mv(Q_i)$到 DB 端;

$P_i=-1; i=i+1; Q_i=V(r,t_2)-\sum_{Q_j \in UQS} P_j Q_j(r,t_2);$

发送 $dw_mv(Q_i)$到 DB 端;

$P_i=1; UQS \leftarrow UQS+Q_{i-1}+Q_i; T_{op} \leftarrow T_{op}+\langle mod(r,t_1,t_2)\rangle;\}$

$db\_mv(A_i)$:

$Collect \leftarrow Collect+P_iA_i; UQS \leftarrow UQS-Q_i;$

If $UQS=\varnothing$ then$\{MV \leftarrow MV+Collect;$

Repeat$\{\forall s \in T_{op}$,发送 $dw_ack(s)$到 DB 端;$\}$

Until T_{op}中记录的操作全部被发送出去$\}$;

$db\_olap(A^*)$:

将 A^*提交给 OLAP 应用.

当某一时刻 DB 端数据更新的速度较快时,DW 端发出的查询需要作补偿处理,UQS(unanswered query set)即是用于记录那些尚未收到结果的查询的.*Collect* 是一个临时变量,为了防止中间状态的不一致,只有当发出的所有 dw_mv 查询都收到结果后,DW 端的视图维护进程才用 *Collect* 更新实体化视图.V 是数据仓库中实体化视图的定义,如 $V=r_1 \bowtie r_2$,则 $V(r_1,[2,3])=[2,3] \bowtie r_2$(用 r_1的元组$[2,3]$替代表达式中相应的关系).Q_i 是 DW 端发往 DB 端的查询,形式上与 V 类似.*MV* 是与 V 对应的实体化视图中元组的集合.T_{op}用于记录 DB 端的数据更新操作,当这些更新已经反映到实体化视图中后,DW 端的视图维护进程将利用 T_{op},通知 DB 端将旧版本的数据从数据库中清除掉.

2.3 DB 端的数据表示

为了使数据仓库用户的查询得到一致结果,DB 端的数据在刷新时必须将刷新前的旧值保存下来,同时又不能阻碍 DB 端用户对数据库的正常使用,因此 MVCA 采用多版本方法对数据库中的刷新数据进行控制和维护.对于数据库中元组的每个"modify"操作,都相应生成一个新版本的元组,旧版本的元组只作逻辑删除,只有在收到 DW 端相应的应答之后才可将其作物理删除.对数据库中元组的"delete"操作亦是如此.每个元组含有 hp 和 tp 两个指针,分别指向最旧的版本和次新版本,我们用一标志位 *flag* 来区分新旧版本数据,并限制它们所能参与的运算.*flag* 有4种可能的值.

I: insert 标记.表明当前元组是刚插入的,只有当收到 DW 端发来的 $ins(r,t)$时,才能将此位置为"Λ"(空标记,后面会讲到).*flag* 为"*I*"的元组不能参加 DW 端发来的 dw_olap 查询运算(因为它还没有反映到相应的实体化视图中),但是可以参与 dw_mv 查询和正常的数据库运算.

D: delete 标记.表明当前元组是刚被删除的(逻辑删除),只有当收到 DW 端发来的相应的 $del(r,t)$时,才可将此元组物理删除.*flag* 为"*D*"的元组只能参与 dw_olap 查询运算.

M: modify 标记.新生成的元组被置上此标记,更新前的元组作为旧版本被移入 Cache 或缓冲池.它的最新版本参与 dw_mv 运算和正常的数据库运算,最旧版本则参与 dw_olap 运算.只有在收到 DW 端发出的相应的 $mod(r,t_1,t_2)$时,才可将相应的旧版本物理删除.当它只有一个版本时,即最新版本和最旧版本都是它自身时,将 *flag* 置为"Λ".

Λ: 空标记.空标记的元组可以参与所有运算.

2.4 MVCA-db算法

在DB端,MVCA-db算法采用多版本方法对历史数据进行控制维护,通过标志位和应答机制确保数据的一致操作.因为DB端的数据库是相对独立的,DB端并不知道自己的哪些数据参与了DW端实体化视图的计算.因此,对于数据库的每一步更新操作,DB端都需通知数据仓库,以确认是否会影响到相应的实体化视图.DB端对数据库的更新操作有3种.

$ins(r,t)$:在关系 r 中插入元组 t;

$del(r,t)$:从关系 r 中删除元组 t;

$mod(r,t_1,t_2)$:用 t_2 更新 r 中的元组 t_1.

相应地,DB端发往DW端的消息是:

$db\_up(ins(r,t))$;$db\_up(del(r,t))$;$db\_up(mod(r,t_1,t_2))$.

这些消息是紧跟着相应的操作发出的.发出消息后,DB端会期待着收到来自DW端的消息,MVCA-db算法根据DW端发回的不同的消息类型作相应的操作处理.

$dw\_mv(Q_i)$:

(1) 对于$\forall r\in Q_i$(r 是作 Q_i 运算时涉及到的关系),使用算法1读出 r 中所有元组;

(2) 使用读出的元组作 Q_i 运算,$A_i\leftarrow$ 结果;

(3) 发送 $db_mv(A_i)$到DW端.

$dw\_olap(Q^*)$:

(1) 对于$\forall r\in Q^*$,使用算法2读出 r 中所有元组;

(2) 使用读出的元组作 Q^* 运算;$A^*\leftarrow$结果;

(3) 发送 $db_olap(A^*)$到DW端;

$dw\_ack(s)$:

if $s=ins(r,t)$ then $t.flag=\Lambda$ /* 将 t 的 $flag$ 置为空 */

else if $s=del(r,t)$ then 物理删除 r 中的元组 t;

else if $s=mod(r,t_1,t_2)$ then

{物理删除 r 中的元组 t_1;

if $t_2.hp=t_2.tp=\Lambda$ /* 即只剩下一个版本的元组时 */

then $t_2.flag=\Lambda$}

算法1.

Repeat {

对于 $t\in r$,

if $t.flag=M$ then 读出t的最新版本

else if $t.flag\neq D$ then 读出t}

Until r中元组已全部扫描.

算法2.

Repeat {

对于 $t\in r$,

if $t.flag=\Lambda$或D then 读出t

else if $t.flag=M$ then 读出t的最旧版本}

Until r中元组已全部扫描.

3 典型应用示例

在第2节中,我们已经对MVCA算法进行了详细的描述.本节我们将通过一个典型示例来说明MVCA在实际应用中是怎样进行联机维护,并解决与"下钻"查询的一致性问题的.

在DB端的数据库中有两个关系 r_1 和 r_2.

r_1:

$flag$	A	B
Λ	1	4
Λ	2	4

r_2:

$flag$	X	Y	Z
Λ	4	5	2

数据仓库中相应的实体化视图的定义为 $V=\prod_{A,Z}(r_1\bowtie r_2)$,则 V 的元组集合为 $MV=\{[1,2],[2,2]\}$.在后面的一段时间里,DB端的数据库进行了3个更新操作.

$del(r_1,[2,4])$;

$mod(r_2,[4,5,2],[4,7,8])$;

$ins(r_2,[4,9,6])$.

而数据仓库在实体化视图的维护期间(不妨假设为在收到DB端这3个更新操作的消息之后)启动了一个OLAP查询Q.

Q: select X,Y,Z
from r_2
where Z in
(select Z
from V)

OLAP查询将Q分成两个按顺序执行的子查询:Q'和Q^*(设Q'返回的结果为α).

Q': select Z
from V

Q^*: select X,Y,Z
from r_2
where Z in α

在不使用MVCA算法的情况下,首先执行Q',由于此时数据仓库中V的元组集合为$MV=\{[1,2],[2,2]\}$,所以,Q'返回的结果为$\alpha=\{2\}$;接着执行"下钻"查询Q^*,此时DB端的数据库已经作了更新操作,原r_2中的元组[4,5,2]已经被新元组[4,7,8]替代,r_2中不再含有$Z=2$的元组,所以Q^*执行完成后返回的结果为$\varnothing$.这个结果是错误的,正确的结果应该是[4,5,2].显然,OLAP查询出现了数据的不一致性.DW端的实体化视图中含有$Z=2$的元组,而DB端的数据库中却没有与之对应的原始数据.出现这种情况的原因是,Q'用的是数据仓库中尚未更新的实体化视图,而Q^*则使用了数据库中更新后的数据.

下面,我们将MVCA算法应用到数据仓库的联机维护中,并按照事件发生的时间顺序来说明MVCA是怎样工作并解决这种不一致问题的.

在DB端(T_i为时标):

T_1:将r_1中元组[2,4]的$flag$置为D;发出$db_up(del(r_1,[2,4]))$;

T_2:将r_2中元组[4,5,2]移入Cache,将[4,7,8]存入原来[4,5,2]的位置,置[4,7,8]的$flag$为M;发出$db_up(mod(r_2,[4,5,2],[4,7,8]))$;

T_3:在r_2中插入[4,9,6],并将其$flag$置为I;发出$db_up(ins(r_2,[4,9,6]))$.

DB端数据库进行了上述操作之后,r_1和r_2就成为如下形式:

r_1:

$flag$	A	B
Λ	1	4
D	2	4

r_2:

$flag$	X	Y	Z	旧版本
M	4	7	8	[4, 5, 2]
I	4	9	6	

在DW端:

T_4:收到$db_up(del(r_1,[2,4]))$;发送$dw_mv(Q_1)$,其中

$$Q_1=V((r_1,[2,4]))=\Pi_{A,Z}([2,4]\overline{\ltimes}r_2);$$
$$P_1=-1;$$
$$UQS=\{Q_1\};$$
$$T_{op}=\{(del(r_1,[2,4]))\};$$

$T5$:收到$db_up(mod(r_2,[4,5,2],[4,7,8]))$;发送$dw_mv(Q_2)$,其中

$$Q_2=V((r_2,[4,5,2]))+Q_1((r_2,[4,5,2]))$$
$$=\Pi_{A,Z}(r_1\ltimes[4,5,2])+\Pi_{A,Z}([2,4]\ltimes[4,5,2]);$$
$$P_2=-1;$$

发送$dw_mv(Q_3)$,其中

$$Q_3=V((r_2,[4,7,8]))+Q_1((r_2,[4,7,8]))$$
$$=\Pi_{A,Z}(r_1\ltimes[4,7,8])+\Pi_{A,Z}([2,4]\ltimes[4,7,8]);$$

$P_3=1$;

$UQS=\{Q_1,Q_2,Q_3\}$;

$T_{op}=\{\langle del(r_1,[2,4])\rangle,\langle mod(r_2,[4,5,2],[4,7,8])\rangle\}$;

T_6:收到 $db\_up\langle ins(r_2,[4,9,6])\rangle$;发送 $dw\_mv(Q_4)$,其中

$Q_4=V\langle(r_2,[4,9,6])\rangle+Q_1\langle(r_2,[4,9,6])\rangle+Q_2\langle(r_2,[4,9,6])\rangle-Q_3\langle(r_2,[4,9,6])\rangle$

$=V\langle(r_2,[4,9,6])\rangle+Q_1\langle(r_2,[4,9,6])\rangle+(V\langle(r_2,[4,9,6])\rangle+Q_1\langle(r_2,[4,9,6])\rangle)$

$-(V\langle(r_2,[4,9,6])\rangle+Q_1\langle(r_2,[4,9,6])\rangle)$

$=V\langle(r_2,[4,9,6])\rangle+Q_1\langle(r_2,[4,9,6])\rangle$

$=\Pi_{A,Z}(r_1\bowtie[4,9,6])+\Pi_{A,Z}([2,4]\bowtie[4,9,6])$;

$P_4=1$;

$UQS=\{Q_1,Q_2,Q_3,Q_4\}$;

$T_{op}=\{\langle del(r_1,[2,4])\rangle,\langle mod(r_2,[4,5,2],[4,7,8])\rangle,\langle ins(r_2,[4,9,6])\rangle\}$;

T_7:用户启动OLAP查询 Q,首先执行 Q',从MV中得到 $Z=2$后,OLAP向DB端发出"下钻"查询 $dw\_olap(Q^*)$,其中 $Q^*=\Pi_{X,Y,Z}(\sigma_{Z=2}(r_2))$.

在DB端:

T_8:收到 $dw\_mv(Q_1)$,应用MVCA-db算法,计算出 $A_1=\{[2,8],[2,6]\}$;发送 $db\_mv(A_1)$;

T_9:收到 $dw\_mv(Q_2)$,应用MVCA-db算法,计算出 $A_2=\{[1,2],[2,2]\}$;发送 $db\_mv(A_2)$;

T_{10}:收到 $dw\_mv(Q_3)$,应用MVCA-db算法,计算出 $A_3=\{[1,8],[2,8]\}$;发送 $db\_mv(A_3)$;

T_{11}:收到 $dw\_mv(Q_4)$,应用MVCA-db算法,计算出 $A_4=\{[1,6],[2,6]\}$;发送 $db\_mv(A_4)$;

T_{12}:收到 $dw\_olap(Q^*)$,应用MVCA-db算法,计算出 $A^*=\{[4,5,2]\}$;发送 $db\_mv(A^*)$.

在DW端:

T_{13}:收到 $db\_mv(A_1)$;

$Collect=\{-[2,8],-[2,6]\}$;

$UQS=\{Q_2,Q_3,Q_4\}$;

T_{14}:收到 $db\_mv(A_2)$;

$Collect=\{-[2,8],-[2,6],-[1,2],-[2,2]\}$;

$UQS=\{Q_3,Q_4\}$;

T_{15}:收到 $db\_mv(A_3)$;

$Collect=\{-[2,6],-[1,2],-[2,2],[1,8]\}$;

$UQS=\{Q_4\}$;

T_{16}:收到 $db\_mv(A_4)$;

$Collect=\{-[1,2],-[2,2],[1,8],[1,6]\}$;

$UQS=\varnothing$;

$MV\leftarrow\{[1,2],[2,2]\}+Collect=\{[1,8],[1,6]\}$(这是实体化视图维护完成后的正确结果);

发送 $dw\_ack\langle del(r_1,[2,4])\rangle$;

发送 $dw\_ack\langle mod(r_2,[4,5,2],[4,7,8])\rangle$;

发送 $dw\_ack\langle ins(r_2,[4,9,6])\rangle$;

T_{17}:收到 $db\_mv(A^*)$;将[4,5,2]提交给OLAP应用(这是OLAP"下钻"查询的正确结果).

在DB端:

T_{18}:收到 $dw\_ack\langle del(r_1,[2,4])\rangle$;从 r_1中彻底删除[2,4];

T_{19}:收到 $dw\_ack\langle mod(r_2,[4,5,2],[4,7,8])\rangle$;从 r_2中彻底删除[4,5,2],并将[4,7,8]的 *flag* 置为Λ;

T_{20}:收到 $dw\_ack\langle ins(r_2,[4,9,6])\rangle$;将 r_2中[4,9,6]的 *flag* 置为Λ;

至此可以看到,MVCA 算法不仅保证了 DW 端的实体化视图得到了正确的维护结果:$MV=\{[1,8],[1,6]\}$,而且保证了在对实体化视图进行联机维护的同时,用户对数据仓库的"下钻"查询 Q 也得到了一致的结果:[4,5,2],而不是∅.

4 结束语

随着数据库应用的不断扩展和深入,人们对数据仓库的需求也与日俱增.数据仓库由脱机维护到联机维护是一个必然的过程,24(小时)×7(天)的工作模式是下一代数据仓库的重要发展方向.MVCA 算法解决了数据仓库联机维护过程中所产生的一种数据一致性问题,并且由于它是在数据仓库联机状态下工作的,所以完全满足"战场决策支持"等对实时要求比较高的关键应用,它可以为指挥员提供即时的、一致的战场决策信息.

参考文献

1 Quass C. Materialized views in data warehouses [Ph. D. Thesis]. Stanford University, 1997

2 Srinivasan V. On-line processing in large-scale transaction systems [Ph. D. Thesis]. University of Wisconsin-Madison, 1992

3 Zhu-ge Y. View maintenance in a warehousing environment. In: Proceedings of ACM SIGMOD Conference. URL: http: //www-db. stanford. edu/warehousing/warehouse. html, 1995

Consistency Algorithm for Data Warehouse On-line Maintenance

LI Zi-mu SUN Li-min ZHOU Xing-ming

(*Department of Computer Science Changsha Institute of Technology Changsha* 410073)

Abstract A warehouse is a data repository containing integrated information for efficient querying and analysis, which data come from the databases or the other info-sources at different places. Materialized view is the primary information entity stored in the data warehouse. It must be refreshed when the corresponding data changed in the database. Consistency problem will be arisen during materialized views on-line maintenance because of the invention of OLAP query. In this paper, the authors introduce a new algorithm, MVCA (multiversion compensating algorithm), using multiversion and compensating techniques, along with acknowledgement mechanism to synchronize the maintenance process between the database and the data warehouse so as to ensure the data consistency. At the end of the paper, the authors illustrate the application of the algorithm by a typical example.

Key words Data warehouse, materialized view, on-line maintenance, data consistency, MVCA (multiversion compensating algorithm).