基于视图自维护理论的高效视图维护机制*)

Efficient View Maintenance based on View Self-Maintenance Theory

韩伟红　贾　焰　王志英　杨晓东

(国防科技大学计算机系　长沙410073)

Abstract　A materialistic view in multidatabase is derived from base database that may not reside at the multidatabase. Maintaining these views efficiently in response to base update is difficult. This paper considers the problem of view self-maintenance, including how to maintain a view and how to determine whether a view is maintainable. Then we combine the theory of self-maintenance with application, and provide a new multidatabase self-maintenance algorithms: EMVM. Our new algorithms make full use of the information in multidatabase views, and minimize the number of queries sent to the local database systems for view maintenance, which makes the multidatabase system more efficient.

Keywords　View maintenance, View self-maintenance, Multidatabase system, Dangling tuple

1　介　绍

多库系统中的全局视图是在全局数据模式上定义的，而全局数据模式由各局部数据库(Local Database: LDB)的数据模式通过异构模式消解得到。因此，对全局视图的存取操作要分解为针对多个LDB的子操作，可能还要对数据模式及查询语言进行相应的转换。如果每一次对全局视图的访问都要转换为对底层数据库的访问，代价是很昂贵的，所以多库系统一般都保持物理的全局视图。底层关系的修改会导致全局视图的改变，因此在保存物理视图的多库系统中，当底层关系发生变化时，要对全局视图进行视图维护，使其与底层关系保持一致。视图维护过程中一般要访问底层的LDB，通信代价很高，研究视图维护的一个重要方面就是如何减少对底层关系的访问。以往视图维护方面的研究都只考虑了单个视图的维护机制，实际上多库系统中往往有很多视图，而且我们在开发多库系统CBMS(CORBA Based Multidatabase System)过程中发现，多库系统中的物理视图包含了大量的底层LDB关系的信息，如果对其进行综合考虑，充分利用所用视图所包含的信息，对某些底层关系的修改，就不需要访问LDB，多库系统本身就可以进行视图维护，即视图的自维护。

例1　假设多库系统中有两个视图$V_1(a,b,c)$，$V_2(b,c,d)$，定义在LDB的$R_1(a,b)$，$R_2(b,c)$，$R_3(b,d)$之上。

```
CREATE VIEW V1(a,b,c) AS
SELECT R1.a,R1.b,R2.c FROM R1,R2
WHERE R1.b=R2.b
CREATE VIEW V2(b,c,d) AS
SELECT R2.b,R2.c,R3.d FROM R2,R3
WHERE R2.b=R3.b
```

V_1是R_1与R_2的自然连接，V_2是R_2与R_3的自然连接，假设V_1为空，关系R_1中插入了行(2,3)，我们考虑以下两种不同情况。

首先，假设$V_2=\{(1,2,3),(3,4,5)\}$，要对V_1进行视图维护，只需要把R_1中新插入的行(2,3)与R_2中的行进行连接操作，并把得到的结果插入V_1即可。通过视图V_2可以知道底层关系R_2中一定包含(3,4)，而且属性b=3的行只有这一行(如果还有其他b=3的行，一定会与R_3中的行(3,5)连接生成视图V_2中的行)，所以V_1的视图维护只需要把行(2,3,4)插入即可，在这种情况下视图V_1是可以自维护的。

考虑另一种情况，假设$V_2=\{(1,2,3)\}$，此时我们不能确切知道R_2中属性b=3的行的内容，如果$R_2=\{(1,2)\}$，则行(2,3)的插入不会对视图V_1产生任何影响；但如果$R_2=\{(1,2),(3,4)\}$，则为了与底层关系保持一致，V_1中必须插入行(2,3,4)。所以，此时视图V_1不能实现自维护。

从上面的例子可以看出，要实现全局视图的自维护，首先要充分利用所有的全局视图，不能只考虑要进

*)本课题受863高技术项目支持。韩伟红　博士研究生，研究方向为数据库和分布式对象技术。贾　焰　副教授，硕士生导师，研究方向为数据库和分布式对象技术。王志英　教授，博士生导师，研究方向为数据库和分布式对象技术。杨晓东　教授，博士生导师。

行视图维护的全局视图.另外,并不是所有情况下视图都可以自维护,视图自维护要解决的另一个问题就是判断视图在给定的修改下是否可以自维护。本文系统地讨论了全局视图的自维护问题,包括判断视图是否可以自维护及如何进行视图自维护,并利用视图自维护理论实现了一个高效的视图维护机制,在多库系统CBMS中取得了很好的应用效果.

2 视图自维护方法

假设多库系统包括全局视图$V_1,V_2,\cdots,V_m$,它们是定义在底层关系$R_1,R_2,\cdots,R_n$之上的,关系R_i在站点S_i上,统称这些底层关系为数据库D,其中每个全局视图V_i都是通过查询Q_i从数据库D中得到的,记为$V_i=Q_i(D)$。如果一个数据库D的$Q_i(D)=V_i$,则称数据库D与视图V_i是一致的。底层关系的修改记为U,U也可具体写为:delete(t_i,R_i)或insert(t_i,R_i),其中delete(t_i,R_i)表示删除关系R_i中的t_i行,insert(t_i,R_i)表示在关系R_i中插入t_i行。

把所有Q_i中出现的底层关系称为与Q_i相关的底层关系,如果一个底层关系R_i在Q_i中出现,则称Q_i为与R_i相关的全局查询,相应的全局视图V_i为与R_i相关的全局视图.另外,在下面的讨论中,我们假设所有与Q_i相关的底层关系的属性都为V_i的属性,这个假设条件是为了简化问题,在本文的最后我们将讨论如何去掉这个假设条件。

下面首先讨论如果知道一个视图是可以自维护的,如何进行维护操作。如果视图是可以自维护的,则不需要确切知道生成视图的底层关系是什么,而只需要找到一个与要进行维护的视图及相关视图都一致的数据库就可以,为了找到这样的数据库,定义导出数据库如下:

定义1 (导出数据库) 设$V_1,V_2,\cdots,V_m$是视图,Q_i是定义视图V_i的查询,底层关系为$R_1,R_2,\cdots,R_n$,导出数据库(记为D')中的关系称为导出关系,导出关系与底层关系一一对应,分别记为$R'_1,\cdots,R'_n$,而这些导出关系是这样得到的:设与Q_i相关的底层关系为$R_j,j=1,\cdots,n$,我们定义R_{ij}为$R_{ij}=\pi_{R_j}V_i$,则$R'_j=\cup_i R_{ij},i=1,\cdots,m,j=1,\cdots,n$.

例2 假设V_1和V_2定义如下:

$V_1(a,b,c)=R_1(a,b)$JOIN $R_2(b,c)$

$V_2(b,c,d)=R_2(b,c)$JOIN $R_3(b,d)$

其中JOIN表示连接操作。$V_1=\{(1,2,3),(1,2,4)\}$,$V_2=\{(3,1,5)\}$,则导出数据库D'包括$R'_1=\{(1,2)\}$,$R'_2=\{(2,3),(2,4),(3,4)\}$,$R'_3=\{(3,5)\}$。

实际上,导出数据库的目的是尽量从所有的全局视图中推导出底层关系的信息,以便在视图维护时不再访问底层关系。

定理1 导出数据库与所有的全局视图都是一致的,即$Q_i(D')=V_i,i=1,\cdots,m$。

证明:设D为底层关系组成的数据库,在生成导出数据库D'时,D'中的每一行都是由V_i中的一行导出的,即D'中的每一行与V_i中的某一行的生成有关,因此这一行一定也属于D,即$D'\subseteq D$,由此可以得到$Q_i(D')\subseteq Q_i(D)$,即$Q_i(D')\subseteq V_i$。

对V_i中的每一行t,由$R_{ij}=\pi_{R_j}V_i,j=1,\cdots,n$可知:$t\in Q_i(\cup_j R_{ij})$,所以$V_i\subseteq Q_i(\cup_j R_{ij})$,另外,由导出数据库的定义可知$\cup_j R_{ij}\subseteq D'$,则$Q_i(\cup_j R_{ij})\subseteq Q_i(D')$,所以$V_i\subseteq Q_i(D')$。

由$Q_i(D')\subseteq V_i$,$V_i\subseteq Q_i(D')$可知:$Q_i(D')=V_i$。□

定义2 (可自维护) 设$V_1,V_2,\cdots,V_m$是全局视图,Q_i是V_i在底层数据库D上的定义,D'是导出数据库,U是D上的修改,V_i在修改U下是可自维护的当且仅当$Q_i(U(D'))=Q_i(U(D))$。

如果视图V_i在给定的修改U下是可自维护的,由定义2可以得出下面视图维护的算法:首先由所有的全局视图计算出导出数据库D',之后计算修改后的导出数据库U(D'),最后把生成视图V_i的查询Q_i用于修改后的导出数据库,得到视图自维护后的V_i,即$V_i=Q_i(U(D'))$。

这个算法的缺点是时间代价较大,要求算出完整的导出数据库D',并在其上重新生成视图V_i。因此,我们对算法进行了改进,使其成为增量式的视图维护算法,即只计算视图改变的部分,而不是重新生成视图。

算法1 增量式视图维护算法

设修改U为insert(t_i,R_i)或delete(t_i,R_i),在修改U下对全局视图V_i进行视图自维护。

1. 对Q_i进行修改,把R_i改为t_i。

2. Q_i中其他的底层关系都改为相应的导出关系。

3. Q_i中的每一个导出关系R'_k都写为$\cup_l(\pi_{Rk}V_l)$,其中V_l为所有与R_k相关的视图。

4. Q_i中所有与R_i相同的属性都改为t_i中该属性的值。

5. 把生成的Q_i用于全局视图,并把得到结果插入V_i(修改为insert(t_i,R_i))或从V_i中删除(修改为delete(t_i,R_i))。

下面我们举一个例子来说明如何进行视图自维护.

例3 仍以例2定义的V_1、V_2为例,假设R_1中插入行(2,3),对V_1进行视图维护。

第1步 把Q_1中的$R_1(a,b)$改为(2,3),修改后的Q_1为:(2,3)JOIN $R_2(b,c)$。

第2步 把R_2替换为相应的导出关系,即:(2,3)JOIN

$R'_2(b,c)$。

第3步 $R'_2(b,c)$为$(\pi_{b,c}V_1)\cup(\pi_{b,c}V_2)$，修改后的$Q_1$为：(2,3)JOIN$((\pi_b,V_1)\cup(\pi_b,V_2))$。

第4步 a属性改为2，b属性改为3，修改后的Q_2为：(2,3)JOIN$((\pi_{3,c}V_1)\cup(\pi_{3,c}V_2))$

第5步 把(2,3)JOIN$((\pi_{2,c}V_1)\cup(\pi_{2,c}V_2))$的查询结果插入$V_1$。

以上的算法是建立在假设视图是可自维护的基础之上的，如果视图不是可自维护的，使用算法1将得到错误的结果，因此在进行视图自维护之前首先要判断视图是否是可自维护的。

3 如何判断视图是否可以自维护

在给定修改下判断视图是否可以自维护的问题可以转换为如下问题：设底层数据库为D，导出数据库为D'，对所有数据库X，如果X在修改前与所有的视图是一致的，则修改后X与D'得到相同的结果，即$Q_i(U(D'))=Q_i(U(X))$，这里的X就是所有可能的底层数据库状态，如果对所有可能的底层数据库状态X，修改后X都与D'得到相同的结果，显然底层数据库D修改后也会与D'得到相同的结果，因此视图是可自维护的(具体证明和判定方法见文[1])，这种方法的缺点是判定复杂，且能否在多项式时间内得出结果还是一个未知问题。因此，我们结合多库系统CBMS中原来的视图维护机制，提出了一个新的快速判断视图能否自维护的算法，并在实际应用中取得了很好的效果。

为了快速判断视图在给定的修改下是否为可自维护的，我们在全局视图管理器的全局目录中多保存了一个记录信息表，表中保存生成全局视图的底层关系的所有连接属性的取值，以及这些关系中连接属性取该值的行数。仍以前面的V_1、V_2为例，假设R_1、R_2和R_3分别为：

关系R_1

a	b
1	2
2	5
3	2

关系R_2

b	c
2	3
3	6
4	6

关系R_3

b	d
3	5
3	6
5	6

则$V'_1=\{(1,2,3),(3,2,3)\}$，$V_2=\{(3,6,5),(3,6,6)\}$，全局视图管理器中要保存的记录信息表V'如下：

记录信息表V'

b	Count-R_1	Count-R_2	Count-R_3
2	2	1	0
5	1	0	1
3	0	1	2
4	0	1	0

有了这个记录信息表，就可以很快地判断出视图是否可以自维护，例如，如果向R_1中插入行(2,8)，通过记录信息表V'可知，关系R_2中没有属性b=8的行，因此R_1的修改对视图V_1没有影响，即视图V_1是可自维护的，自维护的结果是V_1不变；如果向R_2中插入行(2,3)，通过记录信息表V'可知R_2中属性b=3的行只有一行，而通过导出关系可知R_2中有(3,6)行，所以V_1是可自维护的，自维护的结果是向V_1插入(2,3,6)；如果向R_1中插入行(3,4)，通过记录信息表可知R_2中有一行的属性b=4，但无法通过导出关系获得这一行的值，所以V_1不是可自维护的。

定理2 $V_1,V_2,\cdots,V_n$是全局视图，V'为全局视图管理器中保存的记录信息表，Q_i是V_i在数据库D上的定义，设Q_i为：

$$V_i=\pi_a\sigma_P(JOIN_{JA}(R_1,R_j))$$

其中a为投影属性集，P是选择条件集，JOIN是连接操作，JA是连接属性，底层数据库的修改U为Operation(t_i,R_i)，Operation为insert或delete，R'_1为R_i的导出关系，如果：

$$Count(\sigma_{JA=t_i.JA}R'_1)=\pi_{countR_1}(\sigma_{JA=t_i.JA}V^*)$$

则视图V_i是可自维护的，且自维护后的V_i为删除(如果Operation为delete)或插入(如果Operation为insert)$\pi_a\sigma_P(JOIN_{JA}(t_i,R'_1))$。

证明：为简单起见，这里只证明Operation为delete的情况，即底层数据库的修改U为delete(t_i,R_i)。由$Count(\sigma_{JA=t_i.JA}R'_1)=\pi_{countR_1}(\sigma_{JA=t_i.JA}V^*)$可知，$R'_1$中连接属性与$t_i$相同的行的个数与$R_i$中连接属性与$t_i$相同的行的个数相等，即$R_i$中所有连接属性与$t_i$相同的行都在$R'_1$中，因此有$\pi_a\sigma_P(JOIN_{JA}(t_i,R_i))=\pi_a\sigma_P(JOIN_{JA}(t_i,R'_1))$。由传统视图维护算法可知(见文[2])，当底层数据库的修改U为delete(t_i,R_i)时，$Q_i(U(D))=V_i-\pi_a\sigma_P(JOIN_{JA}(t_i,R_i))$，即视图维护操作为从$V_i$中删除$\pi_a\sigma_P(JOIN_{JA}(t_i,R_i))$；由算法1可知$Q_i(U(D'))=V_i-\pi_a\sigma_P(JOIN_{JA}(t_i,R'_1))$，即视图维护操作为从$V_i$中删除$\pi_a\sigma_P(JOIN_{JA}(t_i,R'_1))$，所以$Q_i(U(D'))=Q_i(U(D))$。由可自维护的定义可知，在这种情况下视图为可自维护的，且自维护后的V_i为删除$\pi_a\sigma_P(JOIN_{JA}(t_i,R'_1))$。□

定理2既给出了判断视图在给定的修改下是否可自维护的方法，也给出了如何进行视图自维护的方法。一般情况下，当视图管理器从LDB收到Operation(t_i,R_i)时，视图管理器首先查记录信息表V'，看R_i中JA=t_i.JA的行数，如果视图管理器看到要插入或删除的t_i行本身是悬空行，即R_i中JA=t_i.JA的行数为0，则该行的插入或删除不会对V_i产生任何影响；如果t_i不是悬空行，则计算R_i的导出关系R'_1，判断V_i可否自

维护，如果可自维护，则进行视图自维护；如果 V_i 不可自维护，则全局视图管理器要向 R_j 查询匹配行，修改全局视图的工作由全局视图管理器收到 R_j 的回答后完成。

4 新视图维护方法的算法实现

我们把新的解决视图维护问题的算法称为 EMVM 算法（Efficient Multidatabase View Maintenance），因为全局视图管理器可能要向 LDB 发送一个查询，所以可能出现由于 LDB 上又执行了其他请求而产生的异常情况，因此算法中还考虑了对异常情况的处理。如果出现异常，全局视图管理器根据当前操作和引发异常的操作来执行补偿操作，不需要再向 LDB 发额外的查询，从而避免了嵌套异常。如果由于异常情况引起算法要删除全局视图中不存在的行，则删除操作被延迟到补偿操作之后执行。

设 $V_1, V_2, \cdots, V_m$ 是全局视图，V^* 为记录信息表，Q 是 V_i 在数据库 D 上的定义，底层关系为 $R_1, R_2, \cdots, R_n$，全局视图 V_i 的定义为：

$$V_i = \pi_a \sigma_p (JOIN_{JA}(R_1, R_2, \cdots, R_n))$$

其中 a 为投影属性集，P 是选择条件集，JOIN 是连接操作，JA 是连接属性，底层数据库的修改 U 为 Operation(t_k, R_k)，Operation 为 insert 或 delete。

EMVM 算法在全局视图管理器中为每个全局视图 V_i 保持一个视图维护操作队列 MOQ(V_i)（Maintance Operation Queue），当全局视图管理器收到 LDB 的 Operation(t_k, R_k)，要对全局视图 V_i 进行视图维护操作时，就把它放入队列 MOQ(V_i)，并激活 EMVM 算法，视图维护操作正在处理期间，其它与 V 相关的视图维护请求都放入队列 MOQ(V_i)。如果全局视图管理器向 LDB 发送了一个查询，当全局视图管理器还没有从 LDB 处收到查询结果以前，任何其它从 LDB 收到的 OP(t_k, R_k) 都可能引发异常。但是如果全局视图管理器已经从 LDB 处收到查询结果，之后收到的任何视图维护操作都不会引发异常。为了标识这种情况，全局视图管理器从 LDB 处收到查询结果以后，在 MOQ(V_i) 中插入一个标志行 NULL，标志前的视图维护请求都可能引发异常，而标志后的请求则不会。

定理3 视图维护异常的检测如下：假设全局视图管理器正在处理请求 Operation(t_i, R_j)，对 V_i 进行视图维护，另一个请求 OP(t_k, R_k) 到达，涉及到与前一个请求相同的全局视图，如果满足以下几个条件，就会引发视图维护操作异常：

1. $\pi_{count}\sigma_{JA=t_i.JA}(V^*) \neq 0$；

2. 全局视图 V_i 在修改 Operation(t_i, R_j) 下不是可自维护的；

3. OP(t_k, R_k) 在 MOQ(V_i) 中的标志行 NULL 之前；

4. $R_k = R_j$；

5. $t_k.JA = t_i.JA$。

证明：条件1说明 t_i 不是悬空行，由条件1和条件2可知，V_i 的视图维护操作需要向 S_j 发请求查询 R_j 中与 t_i 匹配的行，条件3表示在查询结果没有返回给全局视图管理器之前操作 OP(t_k, R_k) 就执行了，所以可能引发异常。只有当 OP(t_k, R_k) 修改的是 R_j 中与 t_i 有相同连接属性的行时，异常才会真的发生。条件4表示 OP(t_k, R_k) 修改的是关系 R_j 中的某一行，条件5表示 OP(t_k, R_k) 修改的行与 t_i 有相同的连接属性，因此，如果两个操作同时满足以上的五个条件，视图维护异常就发生了。 □

算法2 EMVM 算法

1. Repeat（进入循环）
2. 从 MOQ(V_i) 中拿出第一个请求 Operation(t_i, R_j)，并把它从队列中去掉；
3. 如果 $\pi_{count}\sigma_{JA=t_i.JA}(V^*)=0$（$t_i$ 是悬空行），进入9；
4. 否则（t_i 不是悬空行），调用视图自维护算法；
5. 如果返回值为 TRUE（视图可自维护），进入9；
6. 否则，向 S_j 发送查询 $\sigma_{JA=t_i.JA}(R_j)$（从 R_j 中查找与 t_i 有相同连接属性的行），从 S_j 收到查询结果 r_i 后向 MOQ(V_i) 插入标志行 NULL；
7. 对 r_i 中的每一行 t_j，如果 Operation 为 delete，从 V_i 中删除 $\pi_a\sigma_p(JOIN_{JA}(t_i, t_j))$；
   如果 Operation 为 insert，向 V_i 插入 $\pi_a\sigma_p(JOIN_{JA}(t_i, t_j))$；
8. 对 MOQ(V_i) 中的每一个请求 OP(t_k, R_k)（检查是否会发生异常）：
   如果 $R_k = R_j$ 且 $t_k.JA = t_i.JA$，调用补偿函数 compensation(Operation(t_i, R_j), OP(t_k, R_k))；
   如果 OP(t_k, R_k) = NULL（可能引发异常的操作结束）
   把 NULL 从 MOQ(V_i) 中删除，进入9（Operation(t_i, R_j) 的处理结束）；
9. 如果 Operation 为 delete，把 V^* 中连接属性为 $t_i.JA$ 的 R_j 的行的个数减1；如果 V^* 中出现 ($t_i.JA$, 0, 0)，则把它从 V^* 中删除（从 V^* 中删除没有用的行）；
   如果 Operation 为 insert，把 V^* 中连接属性为 $t_i.JA$ 的 R_j 的行的个数加1；
10. until MOQ(V_i) 为空。

算法3 视图自维护算法

1. 计算 R_j 的导出关系 $R'_j = \bigcup_j(\pi_{R_j}(V_i))$；
2. 如果 $Count(\sigma_{JA=t_i.JA}(R'_j)) < \pi_{count_{R_j}}(\sigma_{JA=t_i.JA}(V^*))$，返回 False（视图不可自维护）；
3. 否则，从 V_i 中删除（Operation 为 delete）或向 V_i 插入（Operation 为 insert）$\pi_a\sigma_p(JOIN_{JA}(t_i, R'_j))$；
4. 返回 TURE。

算法4 补偿算法 compensation(Operation(t_i, R_j), OP(t_k, R_j))

1. 情况1：Operation = delete，OP = delete，从 V_i 中删除 $\pi_a\sigma_p(JOIN_{JA}(t_i, t_k))$；
2. 情况2 Operation = delete，OP = insert，向 V_i 中插入 $\pi_a\sigma_p(JOIN_{JA}(t_i, t_k))$；
3. 情况3：Operation = insert，OP = delete；向 V_i 中插入 $\pi_a\sigma_p(JOIN_{JA}(t_i, t_k))$；
4. 情况4 Operation = insert，OP = insert；从 V_i 中删除 $\pi_a\sigma_p(JOIN_{JA}(t_i, t_k))$。

定理4 补偿算法是正确的。

（下转第71页）

观察到包丢失率的变化：当自相似系数 H 逐渐减小时，伴随着缓存容量的下降，包丢失率将增加。如图4所示，为不同链路缓存容量(LB)下的包丢失率情况。

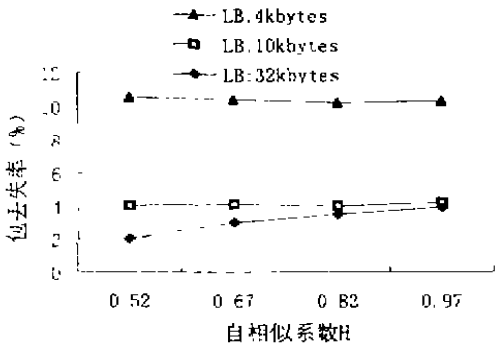

图4 不同链路缓存容量的包丢失率与自相似系数的关系

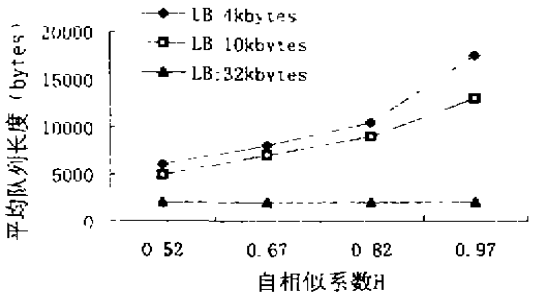

图5 不同链路缓存容量的平均队列长度和自相似系数的关系

在实现可靠流控分组传输时，伴随着自相似性的增长，分组丢失和重发率平滑下降，而自相似性对平均队列长度影响更大，可观察到自相似下的平均队列长度比 Poisson 减少得慢，如增加网络资源配置(诸如链路带宽和缓存空间)，可使网络性能获得超线性的提高，同时增大了队列延迟，而在 IP 多媒体网络环境中，网络尽最大努力传输音频和视频流，只有大大减少队列延迟才能获得较低的包丢失率。

如图5所示，通信量处于较高的自相似情况下，通过增加链路带宽，提供较大的缓存容量可大大减少队列延迟的影响，而对于较低的自相似，效率就没有这么明显。因此，IP 多媒体网络中的高带宽通信链路应能消除队列延迟和包丢失(吞吐量)之间的指数均衡，以支持对 QoS 敏感的业务。

参 考 文 献

1 Guerin R, Peris V. Quality-of-Service in packet network: basic mechanisms and directions. Computer Networks, 1999, 31: 169～189
2 Feldmann A, Gilbert A C, Willinger W. Dynamics of IP traffic: A study of the role of variability and the impact of control. Computer Communication Review, June 1999
3 Zeadally S, Cui Weiyou. Experiences with multimedia applications over native ATM. Journal of Network and Computer Applications, 1998, 21: 107～123
4 Savage S, Cardwell N, Wetherall D, Anderson T. TCP Congestion Control with a Misbehaving Receiver. Computer Communication Review, March 1999
5 Heyman D P, Lakshman T V. What are the Implication of Long-Range Dependence for VBR-Video Traffic Engineering. IEEE Trans. Networking, 1996, 4(3)
6 Leland W E, et al. On the self-similar nature of Ethernet traffic. IEEE Trans. Networking, 1994, 2(1): 1～15

(上接第67页)

证明：为了简单起见，这里只证明第一种情况，即 Operation=delete，OP=delete，从 R_i 中删除 t_i，如果 t_i 是悬空行或者 V_i 可自维护，则全局视图管理器不需要向 R_j 发送任何查询，不会有异常情况产生，所以不需要进行补偿操作(见 EMVM 算法第3步和第5步)，否则，全局视图管理器向 R_j 发查询，查找 R_j 中与 t_i 有相同连接属性的行(见 EMVM 算法第6步)，如果 S_j 在执行这条查询语句之前，先从 R_j 中删除了一行 t_k，且 t_k 与 t_i 有相同的连接属性，此时就发生了异常情况(见 EMVM 算法第8步)，既然 t_k 已经删除，全局视图管理器收到的查询结果中就不会有 t_k，因此视图维护算法的执行结果没有从 V_i 中删除本应该删除的 $\pi_{attr}\sigma_{pred}(JOIN_{jk}(t_i,t_k))$(见 EMVM 算法第7步)，为了使全局视图回到一致状态，补偿算法要从 V_i 中删除 $\pi_{attr}\sigma_{pred}(JOIN_{jk}(t_i,t_k))$。□

现在我们讨论一下前面的假设条件：假设所有与 Q 相关的底层关系的属性都为 V_i 的属性，这个假设条件的目的是简化问题，因为如果有查询中出现的底层关系的属性不在全局视图中出现，则导出关系中不在全局视图中出现的属性只能用一个变量来表示，这会使视图自维护问题变得更加复杂，这里我们不再讨论这个问题。但是，EMVM 算法可以不要这个假设，只要在判断视图是否可自维护时，除了要求 R_j 的导出关系 R'_j 中，连接属性与 t_i 相同的行数等于 V^* 记录的 R_j 中有该属性的行数，再加上一个条件：R'_j 中连接属性与 t_i 相同的行中，视图 V_i 中出现的属性中没有变量，此时 EMVM 算法就可以不要前面的假设条件。

参 考 文 献

1 Huyn N. Multiple-View Self-Maintenance in Data Warehousing Environments. VLDB, Greece, 1997. 26～35
2 Zhuge Y, Molina H G, Hammer J, Widom J. View Maintenance in a Warehousing Environment. ACM SIGMOD Conference. 1995. 316～327
3 Levy A, Sagiv Y. Query Independence of Updates. VLDB, Dublin, Ireland, August 1993. 171～181
4 Harrison J V, Dietrich S W. Maintenance of Materialized View in a Deductive Database: An Update Propagation Approach. In: Proc. of the 1992 JICSP Workshop on Deductive Database, 1992