

数据仓库视图的一致性维护

Consistency Algorithms for Data Warehouse View

焦容 陈金海

(复旦大学计算机科学系 上海200433)

Abstract Consistency of datawarehouse plays a basic role in the application of datawarehouse. The article focuses on one kind of consistency algorithms—DW-Consistency algorithm. In order to maintain the consistency of datawarehouse, an unanswered query set serves as the key role in the DW-Consistency algorithm.

Keywords Datawarehouse, Data consistency, DW-Consistency algorithm, View maintenance

数据仓库是分布、自治的数据源的数据集合,其基本结构可用图1描述:

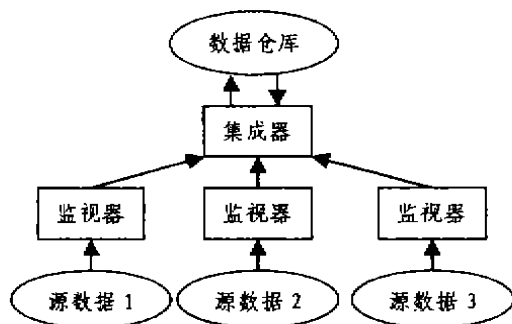


图1 数据仓库结构图

图1中,监视器是很关键的部件,它在相应的数据源收集感兴趣的数据并向数据仓库发送,其功能是:既能识别数据流的变动并通知数据仓库,又能计算出数据源的快照间的差异。

集成器则在收到经监视器处理过的数据后,进行数据综合或转译,并附加一些用户期待的信息后,传给数据仓库。集成器对数据进行集成时,不可避免地会发生不一致问题。举个例子,数据仓库下面的两个数据源,一个在上海,而另外一个在纽约。由于时区差异使得它们对数据更新的时刻不同,容易发生数据仓库未及时更新引起的数据不一致问题。另外,用户从数据库中抽取有用的数据,然后这些数据提取出来后又放入新的数据库中,而这些新的数据库

又可能被再次抽取数据,这种不加控制的连续抽取导致数据间错综复杂的网状结构,可形象地称为“蜘蛛网”结构。很显然,“蜘蛛网”结构容易导致多个应用间的数据不一致问题。既然不一致问题是较普遍的,那么在不停止查询的情形下,如何更新数据仓库的具体视图而又不破坏数据仓库的一致性呢?这就是本文着重研究的问题。我们将构造一个一致性算法来解决这个问题。

一、数据仓库视图的正确性和一致性

在描述一致性算法之前,先给出数据仓库视图正确性和一致性的较形式化定义。由于数据仓库视图和数据源是随时间动态变化的,有必要引进数据仓库和数据源的状态来刻画发生的变化。

1. 数据仓库的状态:每当视图变化,数据仓库的状态也会相应变化。引进记号 $ws_0, ws_1, ws_2, \dots, ws_l$ 来表示这一系列的状态。其中 ws_0 表示数据仓库的初始状态, ws_l 表示数据仓库的最终状态。

2. 数据源的状态:假定有 U 个数据源,每个数据源依次编号为 $1, 2, \dots, u$ 。那么第 x 个数据源的状态可记为 $SS[x]$,注意这是对某个瞬时而言的瞬时状态。一个数据仓库的所有 U 个数据源的状态可组成一个 U 元组的向量 SS ,也就是说 SS 表示数据源的状态。每一次数据仓库的更新都是由一串有先后顺序的数据更新导致的,亦即由一个串行化调度 S 导致。 SS_q 表示调度 S 最终完成后的状态。用 $V(SS)$

焦容 硕士生,研究方向为数据库理论与应用。陈金海 教授,研究方向为数据库理论与应用,并行处理

表示在数据源状态 SS 上的视图。

下面定义四个层次的一致性均基于这样一个前提:初始时源数据和数据仓库下的视图是一致的,即 $V(ss_0) = V(ws_0)$ 。

1. 最终一致性:数据源最终更新,并且所有针对数据仓库的操作停止后,在数据仓库的视图和在数据源上的视图是一致的。

2. 弱一致性:对于某时刻 j 的源数据向量 SS_j ,存在这样一个数据仓库状态,经过一个串行化调度 R ,使得 $V(ws_j) = V(ss_j)$ 。 SS_j 是由所有数据源组成的向量,在 j 时刻数据源 x 的状态 $SS_j[x]$ 也对应 R 的子集 R' 使得 $result(R')[x] = SS_j[x]$ 。

3. 强一致性:数据仓库的状态与数据源的状态 SS 有一一对应关系,记为 m 。即对 j 时刻的数据源状态 SS_j ,存在 m ,使得 $m(SS_j) = WS_j$ 。而且数据仓库的状态与数据源的状态之间应严格遵守时间先后关系,如果 $WS_k < WS_l$ (状态 WS_k 在状态 WS_l 之前),则有 $m(SS_k) < m(SS_l)$ 。

4. 完整性:对于由调度 R 定义的每个数据源状态 SS_j ,必定有一个 WS 到 SS 的映射 m ,使得 $m(SS_j) = WS_j$ 。

二、DW-Consistency 算法

首先引进三个专用术语来描述该算法。

定义1 数据仓库在 n 个关系上的视图 V 由投影-选择-连接(PSJ)来表达: $V = \Pi_{proj}(\sigma_{cond}(r_1 \bowtie r_2 \dots \bowtie r_n))$,在 n 个关系的笛卡儿积的基础上,再进行选择和投影操作。

定义2 视图 V 的视图 MV 是数据仓库的视图 V 的当前状态。

定义3 $next_source(Q)$ 是一个函数,其结果为 $pair(x, Q')$, x 是下一个数据源, Q' 是能在 x 运算的那部分。

在 DW-Consistency 算法中将用到下面的 Source-evaluate(Q)算法:

1. Begin
2. $I=0; WQ=Q; A^0=Q;$
3. $(x, Q') = next_source(WQ);$
4. While x is not nil do
5. Let $I=I+1;$
6. Send Q' to source $x;$
7. When x returns A^i , Let $WQ=WQ(A^i);$
8. Let $(x, Q^{i+1}) = next_source(WQ);$
9. Return(A^i);
10. End.

在 DW-Consistency 算法的实现中将涉及以下一些术语:

• 82 •

1. MV (Materialized View):实例化视图。

2. AL (Action Lists):视图上执行的动作列表集合。

3. pending(Q):当调用 source-evaluate()过程处理查询 Q 时,同时发生了数据更新,该更新暂时放进集合 pending(Q)中。待查询 Q 处理完后,再对 MV, AL 作相应的处理。

4. UQS(Unanswered query set):查询已由数据仓库送至相关数据源,但还没收到相应查询结果。

下面给出 DW-Consistency 算法。

数据源部分的处理:

1. 执行了数据更新 U_i 后,送其至数据仓库。
2. 根据收到 Q_i 查询,在当前数据源状态 $SS[x]$ 下计算 A_i 。

数据仓库部分的处理:

1. AL 初始化为空集
2. 收到更新 U_i 的信息后:
3. 如果 U_i 是删除操作
4. 把任意没处理过的查询 U_i 加到 pending(Q_i);
5. 加入 key-delete(MV, U_i)函数到 AL 中
6. 如果 U_i 是插入操作
7. 令 Q_i 赋值成 $V(U_i)$ 并且将集合 pending(Q_i) 设置成空集
8. 执行函数 source-evaluate(Q_i), 且将结果赋值给 A_i
9. 对 pending(Q_i) 中的每一个元素 U_i , 执行键删除函数 key-delete(A_i, U_i)
10. 在 AL 集合中加入 insert(MV, A_i)
11. UQS 为空集时,将 AL 应用到 MV 作为单事务处理,把与 MV 中重复的元组删除。
12. AL 置空
13. End

下面举例说明该算法如何避免更新异常的情况。本例中定义视图为 $V=r_1 \bowtie r_2 \bowtie r_3$, 其中 r_1, r_2, r_3 是数据源 x, y, z 上的三个关系。关系的初始化为:

r_1 :	A	B
	5	6

r_2 :	B	C
	-	-

r_3 :	C	D
	7	8

初始固化视图 $MV = \Phi$ 。再考虑两个数据源更新操作插入操作 $U_1 = insert(r_2, [6, 7])$ 和更新操作 $U_2 = update(r_3, [7, 8], [8, 9])$ 。须指出的是, update 操作在效果上等价地看作两个操作: delete 和 insert 的共同作用。因此,先执行 $U_1 = delete(r_2, [5, 6])$ 操作,接着执行 $U_1 = insert(r_2, [6, 7])$ 操作,最后执行 $U_2 = insert(r_3, [8, 9])$ 操作。下面具体描述操作步骤:

1. AL = {}。数据仓库从数据源 y 收到更新信息 $U_1 = insert(r_2, [6, 7])$, 按照算法语句行 10, 计算查询 $Q_1 = r_1 \bowtie [6, 7] \bowtie r_3$, 并且 pending(Q_1) 设置为空集, 为了计算 Q_1 , 数据仓库首先将查询 $Q_1 = r_1 \bowtie [2, 3]$ 送到数据源 x 。

(下转封四)

(上接第82页)

2. 数据仓库从数据源 x 收到 $A_1 = [5, 6, 7]$ 。查询 $Q_1^2 = [5, 6, 7] \bowtie r_1$ 被送至数据源 z 继续演算。

3. 在演算过程中, 数据仓库从数据源 z 收到第二个更新信息 $U_1^2 = \text{delete}(r_1, [7, 8])$ 。这时按照算法语句行7进行以下工作: 更新操作来不及处理, 先存进集合 pending(Q_1), 接着把 $\text{key_delete}(MV, U_1^2)$ 放进集合 AL, $AL = \{\text{key_delete}(MV, U_1^2)\}$ 。数据仓库从数据源 z 收到演算结果 $A_1^2 = [5, 6, 7, 8]$ 。注意到算法的语句行13开始处理更新 U_1^2 。因为 pending(Q_1) 有一个元素 $\text{delete}(r_1, [7, 8])$, 因而 $\text{key_delete}(A_1^2, U_1^2)$ 函数应用于数据仓库后查询结果为 $A_2 = \Phi$, 算法的语句行14也就无何东西可加入 AL 了。

4. 数据仓库从数据源 z 收到第三个更新信息 $U_2^2 = \text{insert}(r_2, [8, 9])$ 。按照算法语句行10, 计算查询 $Q_1 = r_1 \bowtie r_2 \bowtie [8, 9]$, 并且 pending(Q_1) 设置为空集。为了计算 Q_1 , 数据仓库首先将查询 $Q_1^2 = r_2 \bowtie [8, 9]$ 送到数据源 y 。

5. 数据仓库从数据源收到 $A_1^2 = \Phi$ 。查询 $Q_1^2 = \Phi \bowtie r_1$ 被送至数据源 x 继续演算。

6. 这时两个更新引起的相应查询动作已全部完

成, 也即 $UQS = \Phi$ 。执行算法的语句行15, AL 中有一个元素 $\text{key_delete}(MV, U_1^2)$, 对数据仓库应用此函数的结果为 $MV = \Phi$ 。

算法分析及结论 该算法很好地解决了数据更新导致的一致性问题的, 其关键在于维护了两个集合: 实例化视图 MV 的动作列表集合 AL 和在处理查询当中并发产生的查询集 pending(Q)。当然这是以增加系统维护开销为代价的, 但经过算法实现检验, 发现 DW-Consistency 一致性算法给系统增加的开销还是在可采纳范围之内。

参考文献

- 1 王珊, 等著. 数据仓库技术与联机分析处理. 科学出版社, 1998
- 2 Alonso R. Data caching issues in an information retrieval system. ACM Trans. on Database System, 1990, 15(3): 359~384
- 3 Bernstein P. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, Massachusetts, 1987

计算机科学

(1974年1月创刊)

第26卷第9期 (月刊)

1999年9月25日出版

中国标准刊号: ISSN 1002-137X
CN50-1075/TP

定价: 7.50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 传真: (023) 63502473

主 编: 朱宗元

印刷者: 国家科技部西南信息中心印刷厂

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: 6210M