数据库系统 事务分片算法 事务处理 26

计算机科学2000Vol. 27№ 6

95-98

事务分片算法*)

The Algorithm of Transaction Chopping

娄勤俭 杨玉萍 卢正鼎 TP311.13

(华中理工大学计算机科学与技术学院 武汉 430074)

Abstract This paper puts forward an algorithm of transaction chopping. It illustrates when the chopping is correct and how to find the finest chopping. And it discusses the algorithm combined with graph theory. Using this algorithm, we can not only obtain more intertransaction concurrency but also enhance intratransaction parallelism.

Keywords Transaction instance, Transaction program, Chopping

1 引言

随着分布式计算的发展,对数据源的远程访问显得越来越重要。新一代的数据库系统(如多数据库系统)要求支持复杂的对象操作、高难度查询和多媒体应用程序。在异构操作环境中,各个站点的自治性意味着可以采用不同的并发控制机制,因此,分布式事务的管理就成为数据库研究领域中的一个重要问题。

为改善事务执行的性能,可将其划分成片,但这有可能导致不可串行化执行。国外许多文献为解决这一问题,要么采用一种新的并发控制机制[1],要么减弱事务的串行性[2]。但是这些方法在很大程度上损害了局部数据库的自治性,我们希望用户尽可能地通过修改应用程序来提高事务处理的性能,从而减少对底层系统自治性的破坏。本文中提出了一种新的事务分片算法,该算法将事务划分成片,在不牺牲可串行性的同时,帮助用户缩短死锁时间,提高事务的并发性。

2 约定

在讨论事务分片算法之前,我们先给出一些约定。首先,数据库用户必须给出如下信息:

a. 能给出在一定时间间隔内运行的所有事务处理程序的特征,这些特征必须被参数化。

b. 用户可使用多版本读一致性,或将事务处理程序划分成更小的片等方法来保证可串行性。

c. 用户知道回滚语句的位置,从而调整代码,使回滚语句尽早地出现。

d. 在出错的情况下,能够决定哪些事务处理例程已经完成,哪些还没有完成。

给出了上述信息后,算法就能根据性质——如果分片之后所得的每一片是可串行化执行的,则事务集的执行是可串行化的——来找到事务集的最优分片。分片之后,所得片按如下规则执行:

a. 遵循事务处理程序定义的先后次序。

b. 各个片根据某种并发控制算法执行以确保串行性,并在终止时提交其变化。

c. 如果某个片因为锁冲突而被取消,则它将被反复提交直到提交成功为止。

d. 如果某个片由于系统错误而被取消,则它将被重新启动。

e. 如果某个片由于回滚语句而被取消,则该事务处理程序其它尚未开始执行的片将不被执行。

全文用顺序、循环、if-then-else 语句等简单的控制结构来考虑事务处理程序。

3 分片的正确性

回滚安全:若事务处理程序 TP 没有回滚语句,或 TP 中所有回滚语句都在它的第一片中,则说 TP 的分片是回滚安全的。

这里第一片中所有的语句必须在 TP 的其它语句之前执行。如果各个事务处理程序的分片是回滚安全的,则说整个分片是回滚安全的。

我们借助于一个无向图来考虑分片的正确性。其结点表示片,边由如下两组集合表示:

(1)C-边:C 表示冲突。假设片 p 和片 p′分别属于不同的事务处理例程,如果颠倒 p 和 p′的执行顺序将

*)本课题得到国防预研基金资助。娄勤俭 博士生;杨玉萍 硕士生;卢正鼎 教授,博士生导师。

改变结果状态或返回值，则说 p 和 p′发生冲突。如果我们不知道 p、p′的语义，那么当 p、p′同时访问某个数据项，且至少有一个对其进行修改时，p、p′发生冲突。若 p、p′之间存在冲突，则在 p、p′之间画一条边，标记为 C。

(2)S_边：S 表示同属，若片 p 和片 p′来自同一事务，则说 p、p′是同属。此时，在 p、p′间画一条边，标记为 S。

这样，我们称所得的图为分片图。若分片图中存在一个环，它至少包含一条 C_边和一条 S_边，则说该分片图具有 SC_环。

定理1 如果一个分片是回滚安全的，且它的分片图不包含 SC_环，则说该分片是正确的。

证明：(1)考虑一组事务处理例程$\{T_1, T_2, \cdots, T_n\}$，它的一个分片是无 SC_环执行的(若一个分片的分片图不包含 SC_环，则称该分片的执行为无 SC_环执行)。假设该执行产生的串行化图中存在一个环。为 $T_i \rightarrow T_1 \rightarrow \cdots \rightarrow T_i$。找出该环中与各个事务处理例程相应的片：$p \rightarrow p' \rightarrow \cdots \rightarrow p''$，p 和 p″都属于 T_i。由于各个片按照执行规则使用两阶段锁，则所有已提交的事务处理例程将产生一个无环串行化图，因此 p、p″互不相同。由于 p、p″是同一事务处理例程 T_i 的不同的片，则在分片图中 p、p″之间存在一条 S_边。串行化图中环的每一条有向边均对应分片图中的一条 C_边，它表示冲突。因此，串行化图中有环意味着在分片图中存在 SC_环。由此可见，$\{T_1, T_2, \cdots, T_n\}$的一个无 SC_环执行等价于已提交的事务处理例程的串行化执行。

(2)对任何一个事务处理例程 T 而言，若其回滚语句处于第一片 p，则 T 的无 SC_环执行不会对数据库产生影响，因为此时分片是回滚安全的。假设 p 与事务处理例程 $T_1, \cdots, T_k$ 的片发生冲突，且 p 跟随它们之后执行，那么按照串行执行的等价性将 T 紧接在这些事务处理例程的最后。此时，T 的首批读操作恰好就是 p 的读操作。由于 p 回滚了，所以 T 也回滚。 □

4 找最优分片

4.1 基本定理

引理1 如果一组被分片的事务处理例程包含 SC_环，那么任何进一步的分片都不能够打破这个 SC_环。

证明：我们用 p 表示事务处理例程 T 的一片，对 p 进一步划分后所得的分片称为 pieces(p)。若 p 不处于 SC_环，则 p 的分片对环无影响；若 p 处于一个 SC_环，则 p 的所有不同子片(例如 q、q′)之间用一条 s_边相连，并且若 p 和 p′间存在 S_边，则 q、q′均以一条 S_边连到 p′，这时有四种情况：

(1)在这个环中有两条 C_边与 p 相连，并且这两条边均连到 pieces(p)中的片 q，那么 q 在这个 SC_环中所处的地位与 p 是一样的。

(2)该环中有两条 C_边与 p 相连，但这两条边分别连到 pieces(p)中的片 q、片 q′，那么此 SC_环包含 q、q′，两者之间以 S_边相连。

(3)有一条 C_边和一条 S_边与 p 相连，假设 C_边连向 pieces(p)中的片 q，那么 q 在这个 SC_环中所处的地位与 p 是一样的。

(4)该环中有两条 S_边连到 p，那么这两条边与 pieces(p)中的每一片都会相连，这样原来的一个环就会变成几个环。

由此可见，无论在何种情况下，任何进一步的分片都不能够打破这个 SC_环。 □

引理2 假设事务处理例程 T 的某个分片 $chop_1$ 将 T 分成两片 p 和 p′，它们处于一个 SC_环中，并假定 T 满足封装冲突假设(指如果操作 o 与操作 o′产生冲突，那么 o 将与包含 o′的任何操作集发生冲突)。那么 p、p′在分片 $chop_2$ 中也将处于一个 SC_环，这里 $chop_2$ 相对于事务处理例程 T 来说与 $chop_1$ 是完全相同的，但是 $chop_2$ 不对其它的事务处理例程进行分片(也就是说，除 T 之外的所有其它的事务处理例程分别被表示成一个单独的片)。

证明：由于片 p 和片 p′来自 T，所以在分片 chop1、chop2中 p、p′之间均存在一条 S_边。又由于 p、p′处于一个 SC_环中，因此该环至少还包含某个事务处理例程 T′的一片 p″。将 T′的所有片合并为一片只会缩短环的长度，原因是原来与 T′的某个片相连的 C_边现在与 T′本身相连，而介于 T′的片之间的 S_边则会被去掉。(为了支持这一论证，我们需要用到封装冲突假设，因为若没有此假设，则可能存在与 T′的一个片发生冲突而不与 T′本身冲突的情况)。 □

由引理1和引理2我们可采用一种系统的方法，尽可能好地对事务进行分片。考虑一组事务处理例程$\{T_1, T_2, \cdots, T_n\}$。我们称$\{C_1, C_2, \cdots, C_k\}$是 T_i 的一个私有分片，记为 private(T_i)，当(1)$\{C_1, C_2, \cdots, C_k\}$是 T_i 的一个回滚安全的分片；(2)由结点$\{T_1, \cdots, T_{i-1}, C_1, C_2, \cdots, C_k, T_{i+1}, \cdots, T_n\}$构成的图中不存在 SC_环。

引理3 在封装冲突假设下，由$\{private(T_1), private(T_2), \cdots, private(T_n)\}$构成的分片是回滚安全的，且无 SC_环。

证明：——回滚安全。由于该分片的各个组成部分(private(T_i))是回滚安全的，所以整个分片是回滚安全的。

——无 SC_环。假设存在一个 SC_环，它包含 private(T_i)的两片，那么由引理2可知，即使所有其它的

事务处理例程未被分片，这个环仍然存在，但这与 private(T_i)的定义矛盾，所以无 SC_环。□

4.2 优化分片的算法

引理3表明，如果我们能够找到各个 T_i 的最优粒度的私有分片 private(T_i)，那么通过对它们进行合并，就可得到最优分片。通常，T_i 的最优分片为：

——T_i 的一个私有分片；

——如果片 p 是这个私有分片的一个成员，那么 T_i 不存在其它包含片 p_1 和 p_2 的私有分片，这里 p_1 和 p_2 划分 p 且都不为空。

下面我们给出一个找到{T_1，T_2，…，T_n}的最优私有分片的算法。这个算法的基本思想是，对各个 T_i 首先根据简单的存取操作将其划分成片，然后查找与其它未分片事务的 C_连通图，各个连通部分组合成一个片，任何更优的分片都将导致 SC_环。该算法如下：

```
Procedure FineChop(T1,T2,…,Tn)
    for i:=1 to n do
        初始化：if 存在回滚语句 then
                pl:=在回滚语句之前发生或与之并发执行
                    的 Ti 中所有数据库写操作；
            else
                pl:=由第一个简单数据库存取操作构成的
                    集合；
            end if;
            P:={x|x 是不在 pl 中的简单数据库存取操
                作}∪{pl}；
        归并片：假设事务 Ti 的分片 P={p1,p2,…,pj}，用
                集合 AllButTi 表示除 Ti 之外所有事
                务的集合；
                考虑结点由 P 和 AllButTi 构成的图，
                边为定义在结点上的冲突边(C_边)，
                构造图中由 C_边形成的连通部分(具
                体可见5.1节 C_连通图算法)；
                根据如下规则修改 P：
            对于各个连通部分 pe1,pe2,…,pek，如果 P 的 k 片
            具有性质 e1<e2<…<ek(r，那么将 pe1,pe2,…,
            pek，放到 pe1中，并去掉 pe2,…,pek。
                        产生的结果是各个连通部分用一个
                        代表性的结点表示。
        称所得的划分为 FineChop(Ti)。
    end for;
    最优分片为{FineChop(T1),FineChop(T2),…,FineChop
    (Tn)}；
    end Procedure;
```

引理4 FineChop(T)是 T 的最优分片。

证明：我们必须证明两点：FineChop(T)是 T 的一个私有分片，并且是最好的私有分片。

——FineChip(T)是 T 的私有分片。(1)回滚安全：检查算法，初始化步骤创建了一个回滚安全的分片，归并步骤仅仅引起 p_1 变大。

(2)无 SC_环：冲突图中任何一个 SC_环将包含一条存在于两个不同片之间的路径。根据 Finechop(T)算法，归并步骤已经把这样的两片合成一片。

——FineChop(T)中任何片都不能进一步被划分。假设有 T 的一个私有分片 TooFine，它将 p 划分成两个非空子集 q 和 r。(1)q 和 r 中的通路产生于归并步骤。在这种情况下，存在一条经过其它事务处理例程从 q 到 r 的路径，它由 C_边组成。这就意味着 TooFine 包含一个 SC_环。(2)如初始化步骤中所构造的，片 p 是第一片 p_1，q 和 r 分别包含 p_1 的回滚语句。这样，q、r 之一会在构成 p_1 的其它回滚语句之前提交。这就违反了回滚安全性。□

4.3 代码分析

(1)构造算法输入　前面我们假设事务处理例程和事务处理程序之间存在一一对应的关系。一般情况下，一个事务处理程序有一个或多个事务处理例程。我们这样构造算法 FineChop 的输入：如果知道一个给定的事务处理程序不会有两个例程并发执行，那么将在 FineChop 算法中对该事务处理程序表示一次；否则，我们将表示两次。表示两次的理由是当一个事务处理程序有多个例程时，SC_图中存在一个环；而当只有一个例程时，环不存在，则该环必定包括对 TP 的两次或多次访问，此时，由于 C_边的对称性，同一程序的两个例程足以用来显示这个环。并且，由于优化分片算法也是对称的，TP 的两个副本将以同种方式分片，因此事务处理程序的所有例程将以同种方式分片。

(2)构造简单数据库存取操作集　下面我们解释怎样获取优化分片算法中所需的简单数据库存取操作集 P。对于非循环代码比较直观，只需在事务处理例程和事务处理程序的数据库访问操作之间构造一个一一对应关系。对循环而言，每次执行是一次独立的数据库访问操作；因此两次执行可以在同一片中，也可在不同的片中。大多数情况下，要么所有的执行都在同一片中，要么都处于不同的片。

4.4 根据依赖图排序片

下面确定怎样执行最优分片算法所产生的各个事务处理例程的片。首先，我们在各个事务处理例程内部建立片之间的数据依赖关系。这些依赖关系用一个有向图表示，称作依赖图，其结点为片，边表示成 D_边。片 p 到片 p'之间存在 D_边，当：(1)p 中有语句 s 在 p'中的语句 s'之前执行，且 s 和 s'冲突；或(2)p 是第一片且包含回滚语句。这里，不再考虑事务间的联系，因为最优分片算法已经考虑了它们。D_边仅关心事务内部

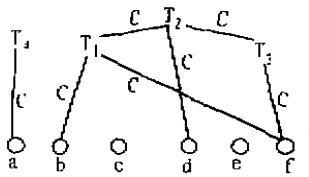

图1　事务处理例程 T_5 的最优分片

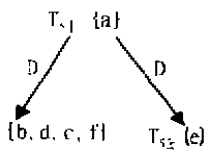

图2　事务处理例程 T_5 的依赖图

的依赖性。处于由D_边构成的有向环中的所有片必须被合并成一个"超片"。

举例说明，如图1所示，假设事务处理例程 T_i 的各个片 a，b，…，f 分别由一条 SQL 语句构成，根据最优分片算法，所得分片为{{a}，{b，d，f}，{c}，{e}}。若在原始事务处理程序中 b 先于 c，c 先于 d，并且 c 与 b、d 发生冲突，那么存在一条从{b，d，f}到{c}的 D_边和一条从{c}到{b，d，f}的 D_边。在这种情况下，这两片就不能以任意顺序执行。我们必须将两片合成一片{b，c，d，f}，这个片就称为超片。获得的依赖图如图2所示(假设 a 包含一条回滚语句)。

根据依赖图，可重新组织事务处理程序。若超片 p 到超片 p′之间存在一条 D_边，那么 p 的所有语句必须在 p′的所有语句之前执行。另外，若一个事务处理例程 T 的两个超片之间不存在 D_边，则这两个超片可并行执行或颠倒顺序执行。这样，当有 D_边指向超片 p 的所有超片执行完时，可立即执行 p。这不仅减短了锁时间，而且增强了事务内部的并行性。

5 结合图论分析

5.1 找 C_连通图算法

在最优分片算法中寻找由 C_边引起的连通部分时，可采用与图论中相似的找连通图的算法。设 G=(V，E)为无向图，其中 $V=\{v_1,v_2,\cdots,v_s\}$，$s=|V|$；$E=\{e_1,e_2,\cdots,e_m\}$，$m=|E|$。令 $e_i=(v_{ai},v_{bi})$，$i=1,2,\cdots,m$，这里 V 中的结点由两部分组成，一部分对应事务 T 的分片，各个分片表示为一个结点；另一部分对应除 T 之外的所有其它事务，各个事务表示为一个结点。s 为这些结点的总个数。E 中的边对应除 T 以外的所有其它事务间的冲突边及这些事务与 T 的分片之间的冲突边。m 为边的总条数。

我们给各个结点赋予特殊的标志位 t。若 $t(v_i)$ 为真，表示结点 v_i 对应 T 的某个分片；否则，对应其它某个事务。

现在判断图 G 是否连通，若不是连通的，有几个连通块，并且各个连通块中有哪些结点对应 T 中的片。算法如图3所示。通过该算法我们就可将输出的各个集合中的分片分别合成一片，从而得到事务 T 的最优分片。

5.2 效率分析

算法的花费主要在于寻找由 C_边引起的连通部分。对各个事务而言，在最坏的情况下算法的代价为 O(m+e)，其中 m 对应事务处理图中 C_边的数目，e 表示标志位 t 为真的结点数，即对任一事务访问的最大次数，也就是简单数据库存取操作集 P 的大小。由于有 n 个事务，因此总的时间是 O(n(m+s))。

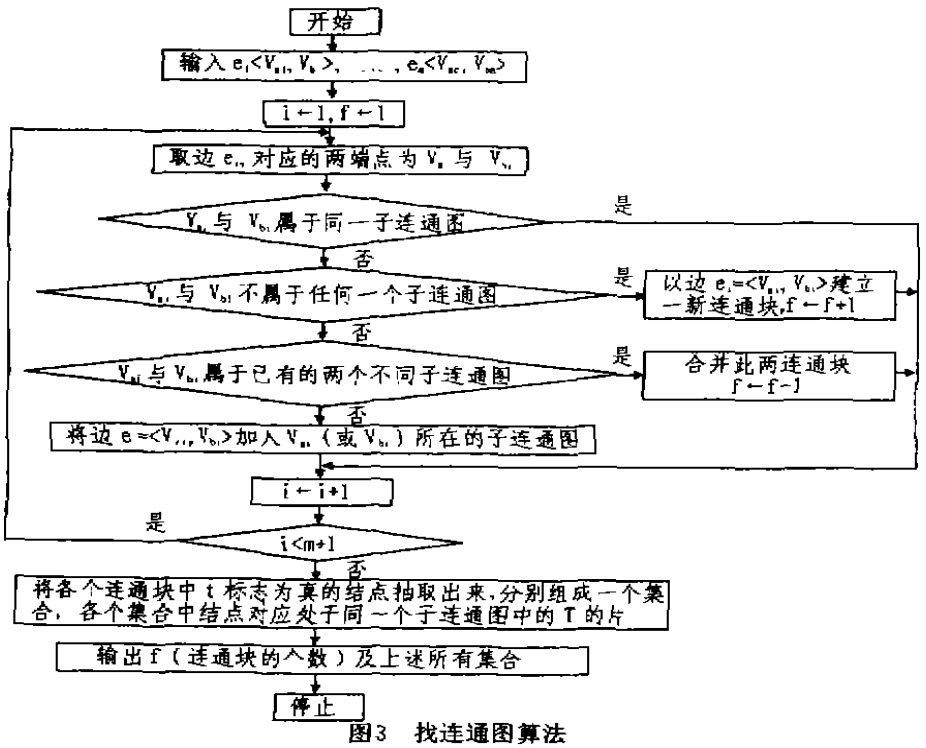

图3 找连通图算法