基于分布式事务流的动态可串行调度算法

梁雄友[1,2],薛永生[1]
LIANG Xiong-you[1,2],XUE Yong-sheng[1]

1.厦门大学 计算机科学系,福建 厦门 361005
2.东莞市塘厦理工学校,广东 东莞 523710
1.Department of Computer Science,Xiamen University,Xiamen,Fujian 361005,China
2.Tangxia Technical School,Dongguan,Guangdong 523710,China
E-mail:dgliangyou@hotmail.com

LIANG Xiong-you,XUE Yong-sheng.Concurrency control algorithm of dynamic adjustment of serialization order for distributed database based on transactions.Computer Engineering and Applications,2010,46(8):144-147.

Abstract: This paper presents an algorithm of dynamic adjustment of serialization order for distributed database on real-time transactions.Running this algorithm can process continual real-time transactions and carry out with an adjustment of serialization order,so the concurrency degree can be improved simultaneously,and the parallel degree of every site of distributed database can be raised too.

Key words: distributed database;concurrency control;dynamic adjustment of serialization order;transactions

摘 要:针对分布式数据库中发生待处理的事务流提出一种动态可串行调度算法。通过执行此算法,能够高效地处理源源不断的事务流,使之尽可能串行调度地执行,从而提高并发度,以及分布式数据库各站点的并行处理度。

关键词:分布式数据库;并发控制;动态可串行调度;事务流
DOI:10.3778/j.issn.1002-8331.2010.08.041 **文章编号**:1002-8331(2010)08-0144-04 **文献标识码**:A **中图分类号**:TP393

1 引言

按串行方式调度总是可以保证数据库状态的一致性。然而,串行调度时,系统运行效率可能会降低,所以实行有关操作时,应该着重强调那些执行正确的可串行化调度,提高并发处理度。在分布式数据库中,还可以提高各站点的并行处理度。

在讨论事务的并发控制时,很多参考文献都只是局部地以 n 个事务为前提加以考虑,在事务串行调度时,又只是着重强调使用锁来实现。文章针对分布式实时事务流中每一个全局事务分解后,在每个站点根据短子事务动作优先以及出现写-读冲突、读-写冲突、写-写冲突时按事务请求的先后次序方式处理,生成执行效率比较高的可串行调度子事务动作序列,从而提高并发度和并行处理度。

2 基本概念及有关定义

2.1 串行调度

定义 1 如有一组事务$\{T_1,T_2,\cdots,T_i,\cdots,T_n\}$,假设事务 T_i 的操作都先于 T_j,记为 $T_i<T_j$。如一个调度 S,其每个事务的执行均有 $T_i<T_j$(i,j 的序号无关),记为 $S=\{\cdots<T_i<T_j<\cdots\}$,称 S 是一串行调度。

2.2 可串行化原理

定义 2 一个事务是一个偏序集:$T_i=(\sum_i,\angle)$,其中

(1) $\sum_i$ 是操作符集,包含$\{r_i[x],w_i[x]|x$ 是数据项$\}\cup\{a_i|c_i\}$;

(2) $\sum_i$ 中 a_i 与 c_i 两者只出现一个;

(3) a_i 或 c_i 是 $\sum_i$ 中的最后一个操作符;

(4) 如果 $r_i[x],w_i[x]\in\sum_i$,则它们必满足 $r_i[x]<w_i[x]$ 或 $w_i[x]<r_i[x]$。

这里只给出了事务一个简单的定义。事实上,事务可能不止以上几种操作符,如以锁方式实现并发控制时,事务包含加锁、解锁操作,分布式事务还包含通讯原语等。

定义 3 事务的一个操作指组成事务 T_i 操作符序列中的任一个 $r_i[x],w_i[x],c_i$ 或 a_i 或公式。

定义 4 如果两个操作 p 和 q 对同一数据项 x 进行操作,其中一个是写操作 $w[x]$,则 p 和 q 称为冲突操作。

定义 5 对于一组并发事务 $T_1,T_2,\cdots,T_n$,一个调度 S,S 是一个偏序集 $S(\sum,\angle)$,其中:

(1) $\sum=\bigcup_{i=1}^{n}\sum_i$;

(2) $\angle\supseteq\bigcup_{i=1}^{n}\angle i$;

作者简介:梁雄友(1971-),男,厦门大学在职硕士,研究方向:分布式数据库、数据挖掘;薛永生(1946-),男,硕士生导师,研究方向:分布式数据库、数据挖掘。

收稿日期:2008-10-10 修回日期:2008-11-28

(3) 对于任意两种冲突操作 p 和 $q \in S$，则有 $p<q$ 或 $q<p$。

定义 6 如果调度 S 中的任意两个事务 T_i，T_j，若 $\bigcup_{i=1}^{n} \sum_i < \bigcup_{j=1}^{n} \sum_j$，或 $\bigcup_{j=1}^{n} \sum_j < \bigcup_{i=1}^{n} \sum_i$，则称调度 S 是串行调度。

定义 7 在调度中，使得数据库从一个一致的状态改变到另一个数据一致状态的调度，称为一致性调度。

定义 8 两个调度 S_1 和 S_2 是等价的，当且仅当：

(1) 调度 S_1 和 S_2 在同一事务上定义，且具有相同的操作集。

(2) 对于有冲突的操作的事务，有：

①$a_i, a_j \notin S_1$，且 $a_i, a_j \notin S_2$；

②若 $p_i <_{s_1} q_j$，则 $p_i <_{s_2} q_j$。

定义 9 与串行调度等价的调度，称之为可串行化调度。

定义 10 两个调度是冲突等价的，如果通过一系列相邻动作的非冲突交换能将它们中的一个转换为另一个，如果一个调度冲突等价于一个串行调度，称之为冲突可串行化。

例 1 考虑如下调度：

$r_1(A)$；$w_1(A)$；$r_2(A)$；$w_2(A)$；$r_1(B)$；$w_1(B)$；$r_2(B)$；$w_2(B)$；

说这个调度是冲突可串行化。图 1 给出了将这一调度转换为串行调度 (T_1, T_2) 的一系列交换，在此串行调度中，T_1 的所有动作都调度到 T_2 的所有动作之前。在每一步中要交换的相邻动作上加了下划线。

$r_1(A)$；$w_1(A)$；$r_2(A)$；$\underline{w_2(A)}$；$\underline{r_1(B)}$；$w_1(B)$；$r_2(B)$；$w_2(B)$；

$r_1(A)$；$w_1(A)$；$\underline{r_2(A)}$；$\underline{r_1(B)}$；$w_2(A)$；$w_1(B)$；$r_2(B)$；$w_2(B)$；

$r_1(A)$；$w_1(A)$；$r_1(B)$；$r_2(A)$；$\underline{w_2(A)}$；$\underline{w_1(B)}$；$r_2(B)$；$w_2(B)$；

$r_1(A)$；$w_1(A)$；$r_1(B)$；$\underline{r_2(A)}$；$\underline{w_1(B)}$；$w_2(A)$；$r_2(B)$；$w_2(B)$；

$r_1(A)$；$w_1(A)$；$r_1(B)$；$w_1(B)$；$r_2(A)$；$w_2(A)$；$r_2(B)$；$w_2(B)$；

图 1 通过交换相邻动作将对冲突可串行化调度

2.3 优先图及冲突可串行性判断规则

已知调度 S，其中涉及事务 T_1 和 T_2，可能还有其他事务，说 T_1 优先于 T_2，写做 $T_1 <_s T_2$，如果有 T_1 的动作 A_1 和 T_2 的动作 A_2，满足：

(1) 在 S 中 A_1 在 A_2 前；

(2) A_1 和 A_2 都涉及同一数据库元素；

(3) A_1 和 A_2 中至少有一个写动作。

这正是不能交换 A_1 和 A_2 顺序的情况。因此，在任何冲突等价于 S 的调度中，A_1 将出现在 A_2 前。所以，如果这些调度中有一个串行调度，那么该调度必然使 T_1 在 T_2 前。

定义 11 优先图的结点是调度 S 中的事务。当这些事务是具有不同的 i 的 T_i 时，仅用整数 i 来表示 T_i 的结点。如果 $T_i <_s T_j$，则有一条从结点 i 到结点 j 的弧。

例 2 下面的调度 S 涉及 3 个事务 T_1、T_2 和 T_3。

S：$r_2(A)$；$r_1(B)$；$w_2(A)$；$r_3(A)$；$w_1(B)$；$w_3(A)$；$r_2(B)$；$w_2(B)$

如果看关于 A 的动作，可以找到 $T_2 <_s T_3$ 的多个原因。例如，在 S 中，$r_2(A)$ 在 $w_3(A)$ 前，而 $w_2(A)$ 既在 $r_3(A)$ 前又在 $w_3(A)$ 前。

类似地，如果看关于 B 的动作，同样可以找到 $T_1 <_s T_2$ 的多个原因。

调度 S 是否冲突可串行化的简单规则 构造 S 的优先图，

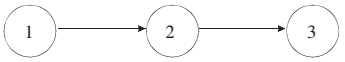

图 2 例 2 中的调度优先图

并判断其中是否有环。如果有，那么 S 不是冲突可串行化的。但如果该图是无环的，那么 S 是冲突可串行化的，而且结点的任一个拓扑顺序都是一个冲突等价的串行顺序。

例 3 考虑调度

S'：$r_2(A)$；$r_1(B)$；$w_2(A)$；$r_2(B)$；$r_3(A)$；$w_1(B)$；$w_3(A)$；$w_2(B)$；

查看关于 A 的动作，仍然只能得到先后次序 $T_2 <_{S'} T_3$。但是，当检查 B 时，不仅有 $T_1 <_{S'} T_2$（因为 $r_1(B)$ 和 $w_1(B)$ 出现在 $w_2(B)$ 前），还得到 $T_2 <_{S'} T_1$（因为 $r_2(B)$ 出现在 $w_1(B)$ 前）。因此，得到图 3 调度 S' 优先图。由于该图有环，断定 S' 不是冲突可串行化的。

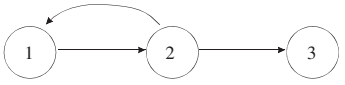

图 3 一个有环的优先图

2.4 时间戳及参考时间戳计算

用时间戳的方法实现事务的并发控制时，给每一个事务赋予一个唯一的时间戳，事务的执行等价于按时间戳的先后次序串行执行，即事务的冲突操作按时间的次序进行调度。

定义 12 时间戳是事务在某一站点激活时由系统赋予的全系统唯一的且能够标识事务激活的先后次序的一个标识。

参考时间戳是该文有关算法中生成实时事务流可串行调度队列时的参考。当没有写–读冲突、读–写冲突、写–写冲突时就参考时间戳的次序来调度。

子事务动作参考时间戳 (TS) 的计算方法：

$$TS = TN \times TIME \quad (1)$$

其中：TN 为事务请求处理时的序号；$TIME$ 为估算该子事务分解后的某个具体动作时所需的机器时间。

在分布式数据库中，事务分解为子事务在相应的场地上执行。对一组分布事务 $T_1, T_2, \cdots, T_n$，在 m 个场地上的一次执行，是由每个场地上的调度序列 $S_1, S_2, \cdots, S_m$ 来安排。而每个场地的调度序列由 LTM 调度，其可串行性由前面的可串行化理论进行判断。当然，仅对局部进行可串行化调度是不够的，还需考虑全局的串行执行，因为全局不串行时，冲突操作可能互相锁住对方的资源，从而造成分布式死锁。

下面，在分布式环境中就源源不断的实时事务流提出一种动态可串行调度算法。

3 算法的基本步骤

(1) 有全局事务请求处理；

(2) 全局事务分解，生成各站点事务，如：$T_A, T_B, \cdots, T_M$（设有 M 个场地）；

下面几个步骤可以同时在 M 个场地间并行处理：

(3) 各站点局部事务分解，生成相应的子事务动作，以站点 A 为例：$T_{A1}, T_{A2}, \cdots, T_{An}$，并估算各子事务动作完成所需的机器时间；

(4) 计算各子事务动作的参考时间戳；

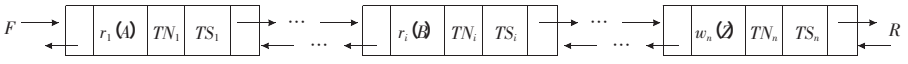

图 4 某个站点生成的动态可串行调度队列存储结构

(5) 根据该子事务动作按参考时间戳的先后次序,将其加入到站点内的可串行调度的队列中。加入队列时若有写-读冲突、读-写冲突、写-写冲突时按事务号的先后次序优先加入队列中;

(6) 将已经完成的子事务动作从可串行调度队列中删除;

(7) 返回步骤(1),不断处理全局事务的请求。

4 算法处理过程描述

4.1 主要的数据结构

4.1.1 各站点可串行调度的队列结构

```
type
pDSQNode=^QNode ;
QNode=record
Op :string;
//事务分解后的具体动作,如读、写、加锁、解锁、提交、回滚等等
TN :longint; //事务号 :记录全局待处理的事务的编号
TS :longint; //参考时间戳
Prior :pDSQNode;     //指向上一个结点的指针
Next :pDSQNode;      //指向下一个结点的指针
End;
Var F,R,P,Q :pDSQNode;
//其中 F 是队头指针,R 是队尾指针,P、Q 是临时结点
```

4.1.2 各个站点分解后待处理的事务及操作序列

```
type
pQWPNode=^QWPNode;
QWPNode=record
Op :string;
//事务分解后的具体动作,如读、写、加锁、解锁、提交、回滚等等
TS :longint;          //参考时间戳
Flag :Boolean;        //是否加入到动态可串行调度队列的标志
Prior :pQWPNode;
//指向上一个结点的指针。特殊地,每个待处理事务的头结点的Prior 指向下一个待处理事务的动作序列的头结点
Next :pQWPNode;
End;
Var T,TH,temp :pQWPNode;
//TH 指向目前最末一个待处理子事务动作序列的头结点
//T 指向是某一待处理子事务动作序列的头结点
```

注:每个待处理事务的头结点的 TS 域内记录的是该全局待处理的事务编号。

4.1.3 全局事务分解序列

```
type
pGQNode=^GQNode;
GQNode=record
TSN :string;
//全局事务分解后某站点的事务代号,如全局 T1 分解后在站点 A 为 :T1A
Flag :Boolean;  //执行标志
Prior :pGQNode;
//指向上一个结点的指针。特殊地,每个全局事务的头结点的Prior 指向下一个全局事务的动作序列的头结点
Next :pGQNode;
End;
Var GT,GTH,Gtemp :pGQNode;
//GTH 指向目前最末一个全局事务分解后的序列的头结点
//GT 指向是某一待处理子事务动作序列的头结点
```

4.2 算法过程具体描述

4.2.1 在各站点生成动态可串行调度序列算法

```
procedure process-transaction-request
begin
 while 系统没有发生中止处理请求时
  begin
   if 有全局事务请求处理 then
     begin
   将该全局事务分解生成各站点事务;
   GT←new (pGQNode) ;
   //为该子事务动作序列创建一头结点 GT
   GT^.TSN←该全局待处理的事务编号;
   GTH^.prior←GT;
   GTH←GT;
   For 刚分解完成的全局事务中的每个站点子事务 do
    Begin
      GTemp←new (pGQNode) ;
      GTemp^.TSN←该站点事务;
      GTemp^.Flag←false;
       //标志为假,表示还没有执行
```

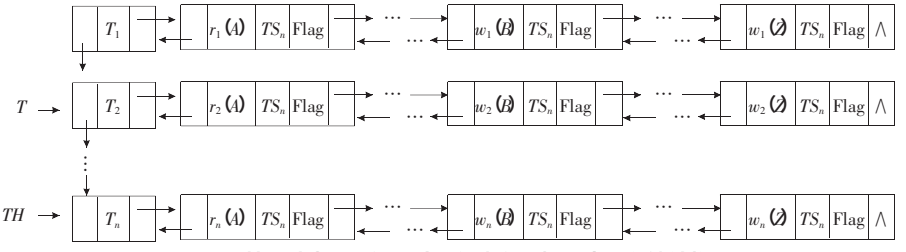

图 5 某个站点已分解待处理的事务及操作序列存储结构

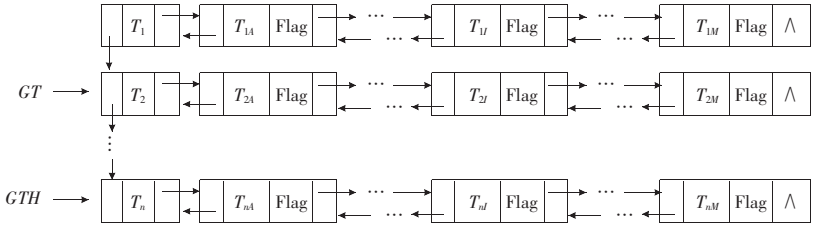

图 6 全局事务分解成各站点事务的存储结构

把 GTemp 按站点编号的先后顺序追加到以 GT 开头的双向链表中;

end for;

//以下步骤在各站点间并行处理:

站点内的事务分解,生成相应的子事务动作;

T←new (pQWPNode);

//为该子事务动作序列创建一头结点 *T*;

T^.*TS*←该全局待处理的事务编号;

TH^.prior←*T*;

TH←*T*;

For 站点内每一个刚分解生成的子事务动作 do

Begin

Temp←new (pQWPNode);

Temp^.*op*←该子事务动作;

Temp^.*TS*←按公式 (1) 计算该子事务动作的参考时间戳;

Temp^.Flag←false;

//标志为假,表示还没有执行

把 *Temp* 按其参考时间戳的先后顺序插入到以 *T* 开头的双向链表中;

P←new (pDSQNode);

P^.*op*←*Temp*^.*op*;

P^.*TN*←*T*^.*TS*;

P^.*TS*←*temp*^.*TS*;

Q←*R*; //*Q* 指向可串行调度的队列尾

While (*P*^.*TS*<*Q*^.*TS*) and (*P*≠*F*) and (*P*^.*Op*∩*Q*^.*Op*≠) do

Q←*Q*^.Prior;

//若没有写_读冲突、读_写冲突、写_写冲突,且参考时间戳较小,则 *Q* 指针向队头方向移动

position←*Q*;

将 *P* 插入到 Position 后面;

end for;

end if;

end while;

end procedure;

4.2.2 各站点内动态可串行调度事务序列出队算法

procedure de_serialization_queue;

begin

if *F*≠NIL then

begin

Q←*F*;

F←*F*^.*next*;

执行 *Q*^.*Op*;

找到 *Q* 在该站点待处理的同一个事务号内各操作动作中对应的结点 *Temp*;

temp^.Flag←True;

if 该站点待处理的同一事务号每一个操作动作结点的 Flag 都为 True then

begin

删除该事务序列;

在全局事务中同一个事务号对应场地结点的 Flag 为 True;

if 全局事务中与该事务号相同的各场地结点的 Flag 都为 True then

begin

在全局事务中删除该事务序列;

end if

end if;

Free (*Q*);

End if;

end procedure;

5 算法复杂性分析及正确性证明

由于 3 个数据结构都是改造过的双向链表,是一种线性结构,其存储空间复杂性为 $O(n)$,程序处理过程也只是进队、出队的普通算法,其时间复杂性为 $O(n)$。

在论文的基本概念及有关定义描述中,只要生成的每个子事务动作序列都能保持全局事务的请求次序,则可以保证其优先图是无环的,即冲突可串行调度是正确的。而该文算法正是能保证这一关键。

6 实验环境模拟与结论

6.1 实验条件

硬件环境:P4 2.4 GHz,内存 512 MB,5 台 (其中 1 台作主场地,4 台作子站点)。

网络环境:100 Mbit 以太网。

软件平台:Windows2000 Server+VC6.0。

6.2 实验环境模拟

6.2.1 主场地中全局分解过程模拟 (两线程同时进行)

线程 1

1.随机产出一全局模拟事务

2.读出其中事务号,各站点信息

3.发送信息到各相关站点

4.把当前全局事务信息添加到全局队列尾

5.延时一个随机产生的暂短时间,返回 1

线程 2

1.等待接收来自各站点的信息

2.若各站点把同一全局事务号都处理完成,从全局事务删除该事务号结点

3.返回 1

6.2.2 各站点内的事务分解,执行过程模拟 (该实验只是模拟事务执行,故没有考虑死锁的情况)

线程 1

1.接收主场地发来的信息

2.生成相应的子事务模拟动作

3.按 4.2 算法过程将子事务动作可串行调度插入链表

4.返回 1

线程 2

1.检测同一事务号的各子事务动作是否完成,完成则删除

2.返回 1

线程 3

1.读取链表队头元素

2.按原估计完成时间模拟执行

3.返回 1

线程 4

1.读取链表队头元素

2.若该元素中的子事务动作与线程 3 无冲突,则按原估计完成时间模拟执行;否则本线程进入等待状态,让线程 3 优先执行

3.返回 1

6.3 实验结果比较

重点考虑分布式实时事务流中每一个全局事务分解后,在

(下转 152 页)

[2] Heeger D J,Bergen J R.Pyramid-based texture analysis/synthesis[C]//Proc of ACM SIGGRAPH.Los Angeles:ACM Press,1995:229-238.
[3] Portilla J,Simoncelli E P.A parametric texture model based on joint statistics of complex wavelet coefficients[J].International Journal of Computer Vision,2000,40(1):49-71.
[4] Bonet J S.Multiresolution sampling procedure for analysis and synthesis of texture images[C]//Proc of ACM SIGGRAPH.Los Angeles:ACM Press,1997:361-368.
[5] Efros A A,Leung T K.Texture synthesis by non-parametric sampling[C]//International Conference on Computer Vision.Greece:ACM Press,1999:1033-1038.
[6] Wei L Y,Levoy M.Fast texture synthesis using tree-structured vector quantization[C]//Proc of ACM SIGGRAPH.Los Angeles:ACM Press,2000:479-488.
[7] Ashikhmin M.Synthesizing natural textures[C]//Proc of ACM Symposium on Interactive 3D Graphics.Los Angeles:ACM Press,2001:217-226.
[8] Xu Y,Guo B,Shum H Y.Chaos mosaic:Fast and memory efficient texture synthesis,MSR-TR-2000-32[R].2000.
[9] Liang L,Liu C,Xu Y Q,et al.Real-time texture synthesis by patch-based sampling [J].ACM Transactions on Graphic,2001,20(3):127-150.
[10] Efros A A,Freeman W T.Image quilting for texture synthesis and transfer[C]//Proc of ACM SIGGRAPH.Los Angeles:ACM Press,2001:341-347.
[11] 张岩,李文辉,孟宇,等.应用PSO的快速纹理合成算法[J].计算机研究与发展,2005,42(3):424-430.
[12] Wei L Y,Levoy M.Texture synthesis over arbitrary manifold surfaces[C]//Proc of ACM SIGGRAPH.Los Angeles:ACM Press,2001:355-363.
[13] Turk G.Texture synthesis on surfaces[C]//Proc of ACM SIGGRAPH.Los Angeles:ACM Press,2001:347-354.
[14] Praun E,Finkelstein A,Hoppe H.Lapped textures[C]//Proc of ACM SIGGRAPH.Los Angeles:ACM Press,2000:465-470.
[15] Ying L,Hertzmann A,Biermann H,et al.Texture and shape synthesis on surfaces[C]//Europe Graphic Work on Rendering.Paris:CA Press,2001:135-141.
[16] Bar-Joseph Z.Statistical learning of multi-dimensional textures[D].Israel:The Hebrew University of Jerusalem,1999.
[17] Schödl A,Szeliski R,Salesin D H,et al.Video textures[C]//Proc of ACM SIGGRAPH.Los Angeles:ACM Press,2000:489-498.
[18] Schödl A,Essa I A.Controlled animation of video sprites[C]//Proc of ACM Symposium on Computer Animation.Los Angeles:ACM Press,2002:121-127.
[19] Wang Y,Zhu S C.A generative model for textured motion:Analysis and synthesis [C]//Proc of European Conf on Computer Vision(ECCV),Copenhagen,2002:583-598.
[20] Doretto G,Chiuso A,Soatto S,et al.Dynamic textures [J].International Journal of Computer Vision,2003,51(2):91-109.
[21] Kwatra V,Schödl A,Essa I A,et al.Graphcut Textures:Image and video synthesis using graph cuts[C]//Proc of ACM SIGGRAPH.Los Angeles:ACM Press,2003:277-286.
[22] Bhat K S,Seitz S M,Hodgins J K,et al.Flow-based video synthesis and editing [C]//Proc of ACM SIGGRAPH.Los Angeles:ACM Press,2004:360-363.
[23] Holland J.Adaptation in natural and artificial systems[M].[S.l.]:University of Michigan Press,1975.
[24] Nagasaka A,Tanaka Y.Automatic video indexing and full-video search for object appearances[C]//Knuth E,Wegner L M.IFIP Proceedings of Visual Database Systems,Amsterdam.The Netherlands:North-Holland,1992:113-127.
[25] Schödl A.Multi-dimensional exemplar-based texture synthesis[D].Georgia Institute of Technology,USA,2002.

(上接147页)

表1 实验结果

所用算法	单位时间事务发生数(10 min)	事务完成数	运行效率/(%)
该文算法	300	292	97.3
事务流顺序执行算法	300	256	85.3

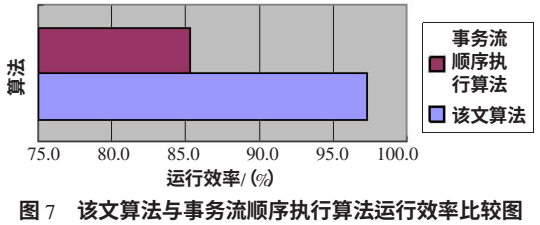

图7 该文算法与事务流顺序执行算法运行效率比较图

每个站点根据短子事务动作优先以及出现写-读冲突、读-写冲突、写-写冲突时按事务请求的先后次序方式处理,生成执行效率比较高的可串行调度子事务动作序列,实验数据表明该算法有更高并发度及并行处理度。

参考文献:

[1] 郑振楣,于戈,郭敏.分布式数据库[M].北京:科学出版社,1998.
[2] Drozdek A.Data structures and algorithms in C++[M].2nd ed.[S.l.]:Brooks/Cole Publishing Co,2003.
[3] Connolly T,Begg C.Database systems:A practical approach to design,implementation,and management[M].3rd ed.[S.l.]:Addison-Wesley,2004.
[4] 刘云生,覃飙,杨进才.一种分布式实时事务调度算法[J].小型微型计算机系统,2003,24(6).
[5] 陈子军,刘国华,周傲英.基于语义可串行性的乐观并发控制算法[J].小型微型计算机系统,2003,24(7).
[6] 张晓芳,张文彬.实时数据库系统中动态调整串行次序的乐观并发控制算法[J].计算机工程与应用,2002,38(9):168-172.
[7] Wu Jian-guo,Liu Ming-ye.An interruptalbe lock and unlock algorithm[J].Journal of Beijing Institute of Technology,1997,6(3).
[8] 梁晟,施伯乐.嵌套事务可序列化调度中的隐式约束[J].计算机研究与发展,2003(2):325-329.
[9] Lynch N A.Distributed algorithms[M].北京:机械工业出版社,2004.
[10] Özsu M T,Valduriez P.Principles of distributed database systems[M].2nd ed.[S.l.]:Prentice Hall,2002.
[11] Garcia-Molina H,Ullman J D,Widom J.数据库系统全书[M].北京:机械工业出版社,2003.
[12] Gray J,Reuter A.Transaction processing:Concepts and techniques[M].[S.l.]:Morgan Kaufmann,2002.
[13] 贾焰,王志英,韩伟红,等.Technology of Distributed Database[M].北京:国防工业出版,2000.
[14] 刘云生,李国徽.实时数据库的准一致性可串行化并行控制[J].计算机学报,1999(4):420-423.