一种嵌入式移动实时数据库的并发控制策略

吴 海 陈 巍 卢炎生
(华中科技大学计算机学院 武汉 430074)

摘 要 嵌入式移动实时事务的并发控制除了满足传统事务的基本特性外,还要着重考虑优先级颠倒、不必要的事务重启和全局数据一致性及混合事务系统的性能等问题。基于绝对串行化时序与选择重启的乐观并发控制算法(OCC-ASTOSR),利用绝对时标在广播循环的支持下调整移动客户端和中心数据库服务器上的事务串行化顺序,检测事务的数据访问冲突,并应用选择重启的方法解决冲突。事务随着运行过程更新读写数据集信息,每个数据对象都维护相关的时间信息,通过无线网络通信传递这些控制信息,交互地完成移动客户端和中心服务器中的本地事务验证和移动事务两阶段验证提交。

关键词 实时事务,乐观并发控制,绝对串行化时序,选择重启

Concurrency Control Strategy of Embedded Mobile Real-time Database

WU Hai CHEN Wei LU Yan-sheng
(Computer College, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract Besides maintaining the characteristics of traditional transactions, the embedded mobile real-time transactions should focus on how to resolve the problem of priority reversing, fruitless restarting, and consistency of global data and high performance of the mixed transactions system. Optimistic Concurrency Control with Absolute Serialize Time Order and Selected Restart (OCC-ASTOSR) uses an absolute timestamp to adjust the serialize order of transactions in mobile client and central server, checks over the confliction of data access and takes the selected restart method to resolve the confliction. Transactions update their read set and write set when access the data and every data maintains some relational time information, they exchange message via wireless network and complete two phase validation in the control of the whole system.

Keywords Real-time transactions, Optimistic concurrency control, Absolute serialize time order, Selected restart

1 概述

乐观并发控制的最大问题在于事务重启。怎样减少不必要的事务重启,是提高系统性能及减少实时事务错失率的关键。传统的动态调整串行化顺序方法在简单的实时数据库系统中能起到很好的效果,但若直接应用到移动环境下,各移动主机上的并行移动事务的状态信息如果全部交由中心服务器维护并验证,由于无线网络的不稳定和延时,时效性很差。

覃飙提出的 HDOCC-E2PC 算法[1]将分布式封锁方法应用到乐观控制协议中,在客户端的局部操作时给数据加预读写锁 PR-lock 和 PW-lock,利用局部串行化时标调整事务串行化顺序,提交到中心服务器进行全局验证时,通过 PR-lock 和 PW-lock 更新为真正的读写锁,将调整串行化顺序的规则放到更新锁的规则中,间接地实现了分布式环境下的全局一致性。Lam 等人在 OCC-PVTO 算法[2]中采用时间戳间隔调整事务的串行化顺序,传输时间戳间隔附加的数据量较多,且它是后向验证,不利于更早地发现冲突和实施分级控制。

本文提出基于绝对串行化时序与选择重启的乐观并发控制 OCC-ASTOSR(OCC with Absolute Serialize Time Order and Selected Restart)协议。利用绝对串行化时标调整事务的局部和全局串行化顺序,结合广播循环协调两阶段提交过程中的时间控制信息,在发生冲突时采用选择重启策略,提出一种前向验证的支持混合事务模型的乐观并发控制方法。

2 调整串行化顺序

当采取前向验证方式检测验证事务与当前所有其它处于读阶段的事务的数据冲突时,如果事务之间不存在冲突的话,按乐观控制中通过验证即提交的方法,验证的时间顺序即可作为事务的串行化时序;如果事务存在冲突,则根据冲突类型决定是否通过验证和调整活动事务的串行时序。要处理的冲突有如下 3 种:

(1)验证事务(Tv)的读和执行事务(Te)的写有冲突。因 Te 的写操作暂时是在私有空间完成的,无法影响到当前 Tv 读到该数据的值,按验证顺序 Te 将在 Tv 之后提交,所以串行化顺序无需调整;

(2)验证事务(Tv)的写和执行事务(Te)的读有冲突。若

到稿日期:2008-03-20 本文受国家部委重点科技攻关项目(项目号:513150402)资助。
吴 海(1972—),男,博士生,讲师,研究方向为移动环境下的数据库技术,E-mail:hust_wuhai@163.com;陈 巍 硕士研究生,研究方向为实时数据库;卢炎生(1949—),男,博士生导师,研究领域为移动数据库、软件测试、构件技术。

Tv 串行化在 Te 之前，则 Tv 的写应该影响到 Te 的读，但此时 Te 读取的数据是 Tv 写之前的值，故按验证顺序提交将产生数据的不一致，必须将 Te 的串行化顺序调整到 Tv 之前；

(3)验证事务(Tv)的写和执行事务(Te)的写有冲突。写写冲突本来也会造成数据不一致错误，但按照以上两点规则进行对事务串行化顺序的调整之后，任何两个写事务之间将不会存在读事务，因此只需把相连的写操作中的最后一个作为对数据库的有效更新，前面的全部忽略，故这种情况下事务的时序也无需调整。Thomas 证明了在写写冲突中的这种调整是满足视图可串行化的[3]。

为避免串行化图中出现环，需确保那些已经提前的事务既不会存在无效读写，又不会和正在验证的事务存在写读冲突。引入绝对时标后，在事务开始时时标无限大，表示不知道事务何时可串行化完成提交，事务进入验证阶段即开始根据上述冲突处理方法决定时标怎样调整，并检测那些无法解决的冲突进行事务重启，最终确定验证事务的串行化顺序。整个过程由移动客户端和中心服务器交互完成。

3 移动客户端的算法流程

事务 T 在移动主机上产生并开始运行，其串行化时标记为 *serialize*(*T*)。在执行阶段(读阶段)，若事务 *T* 对数据对象 *a* 有读或写操作，则将 *a* 加入事务的读数据集 *RS*(*T*)或写数据集 *WS*(*T*)中。同时移动主机周期性监听无线网络的下行通道，接收中心服务器的广播消息。每隔一定时间，中心服务器周期性地产生对所有相关移动主机的广播。广播消息中就包含有上一周期中在中心数据库通过验证的所有事务操作的数据对象集，对每个数据对象指定一个对象名，记为 *a*，*b*，*c* 等，并维持两个属性：最近一个通过验证的事务写数据对象的时间，记为 *RWT*(*a*)；最近一个通过验证的事务成功读数据对象的时间，记为 *RRT*(*a*)。对于本地数据，这两个属性直接由嵌入式数据库实时更新。

由于乐观并发控制方法中事务的写是在私有空间中执行的，只有提交之后才会真正改变数据库内部的值，所以当一个事务读取某个数据对象时，它所读取的值实际上就是最近一个通过验证的事务写入的值。因此，为方便以后算法的判断，再为事务的读数据集 *RS*(*T*)中的每个数据对象 *a* 维持一个属性 *readVersion*(*T*,*a*)，用于标识读取的数据对象值的版本，它设置为事务读 *a* 时的 *RWT*(*a*)值。其逻辑意义在于，小于 *RWT*(*a*)时数据对象 *a* 的值是最近一个通过验证的事务对它更新之前的版本，大于 *RWT*(*a*)时 *a* 的值则是更新后的版本。

多个事务并发运行时，当一个事务 Tv 进入验证阶段后，将当前时间作为它的绝对串行化时标，依次检测所有当前处于执行阶段的其它事务 Te。若某个 Te 和 Tv 存在读-写冲突，由上节所述冲突处理方法(2)可得，为保证视图可串行化 Te 的提交顺序应在 Tv 之前，则把 Te 加入到 Tv 的前序事务集合 $BG(Tv)$中。之后，对所有在 $BG(Tv)$中的事务 Te，如果又有 Te 和 Tv 存在写-读冲突，则说明 Te 的串行化顺序应在 Tv 之后，之前的调整导致串行化图产生了环，不能得到视图可串行化的调度。此时，就需要执行选择重启来解决事务间的数据访问冲突。

验证阶段的算法伪码如下：

```
validate(Tv)
{
  If (current time < serialize(Tv))
      serialize(Tv) = current time;
  For each Te in active transactions do
    {
      If(serialize(Te) < serialize(Tv))
//之前被其它事务调整过
      BG(Tv) = BG(Tv)∪{Te} ;
        If(WS(Tv) ∩ RS(Te) ≠ Φ)
        BG(Tv)=BG(Tv) ∪{Te} ;
      }
    For each Te in BG(Tv) do
//验证所有前调事务是否无冲突
  {
  If (serialize(Te) >= serialize(Tv))
  serialize(Te) = serialize(Tv) −ε;
  For each one in RS(Te) does
  If (serialize(Te) < readVersion(Te,a))
    Restart Te;
For each one in WS(Te) do
  If (serialize(Te) < RRT(a))
    Restart Te;
  If(RS(Tv)∩WS(Te)≠Φ)
    {
    selected_restart(Tv,Te);
    //选择重启算法
    If (Tv need restart)
     {
    for each Te in BG(Tv) do    //还原不需前调的事务
    eset serialize(Te) to their old values;
    BG(Tv) = Φ;
    Return false;
    }
    }
  }
  return true;
}
```

上面的算法中，*serialize*(*T*)<*current time* 和 *serialize*(*Te*)<*serialize*(*Tv*)的条件可能发生在下列的情况：事务 *T*1，*T*2，*T*3 并发运行，*T*1 与 *T*3 有写-读冲突，*T*1 先验证时 *T*3 的绝对时标被调整到小于 *serialize*(*T*1)，之后 *T*2 在 *T*3 之前达到验证阶段，*T*2 验证时其绝对时标已大于 *T*1，显然更比 *T*3 大；*T*3 验证时由于绝对时标已经被调整过，此时将不再把当前时间赋给它。

当事务成功通过验证后，如果它是本地事务，则进入提交阶段，把所有更新从私有缓存复制到嵌入式数据库中，并对所有它操作过的数据对象进行读写时间的更新，方法如下：

```
For each one in RS(Tv) do
{
    If (serialize(Tv) > RRT(a))
        RRT(a) = current time ;
}
For each a in WS(Tv) do
{
    If (serialize(Tv) > RWT(a))
    {
```

```
        RWT(a) = current time;
        write new value of a to EDB ;
      }
    }
```

以上过程同时保证了只将相连多个写操作中的最后一次有效的写操作从事务私有空间复制到数据库中，而之前的无效写操作则全部忽略。

4 中心数据库的处理

中心服务器周期性地向移动客户端发送广播消息，每两个相邻的广播操作之间的间隔称为一个广播循环。广播循环在逻辑上也可按照乐观控制的方法划分执行、验证、提交3个阶段。与移动客户端稍有不同的是：在执行阶段它的活动事务集是来自于上行消息而非自己产生，故不必再为其设置私有缓存，改写直接在中心数据库上进行；而验证和提交阶段则一起完成，数据库更新后，需将新的数据值和成功完成提交的事务等一系列信息封装到下行消息中，广播给所有相关移动主机，实现全局数据副本的更新。

执行阶段，采用一个事务池管理移动客户端通过上行消息发送过来的事务信息。中心服务器记录上一个广播操作的时刻 *lastBC*，对半提交的移动事务 *T*，若 *serialize*(*T*)<*lastBC*，则表明 *T* 的第二阶段提交由于无线网络延迟至少错过了一个广播循环，那么它验证时采用的 *WRT* 和 *RRT* 都将是无效数据，所以事务 *T* 必须重启，将其加入失败事务集 *FTG* 中。没有超时的移动事务则加入事务池中按优先级排序，这些事务将作为验证阶段的活动事务集。全局验证算法为：

```
global_validate()
{
  While(transactions pool is not empty)
  {
    Get T from the transactions pool ;
    For each one in RS(T) do
    {
    If (RWT(a)>readVersion(T,a))
    //后向验证读写冲突
      {
      If (serialize(T)>RWT(a))
      {
        FTG = FTG∪{T};
        Return;
      }
    For each b in WS(T) do
    If (serialize(T)<RRT(b))
    {
      FTG = FTG∪{T};
      Return;
      }
    }
  If (serialize(T)>RRT(a))
  //更新读时间
    RRT(a) = serialize(T);
  }
  For each one in WS(T)
  {
  If (serialize(T)>RWT(a))
  //更新写时间
  {
    RWT(a) = serialize(T);
    write new value of a to CDB ;
  }
  }
  STG = STG ∪ {T};
 }
}
```

可见服务器上的验证是后向方式的，通过验证的事务将被加入到成功事务集 *STG* 中，其新数据值提交后即可向移动客户端发送广播。

接收到广播消息后，移动客户端将重启 *FTG* 中的半提交事务，恢复所有受其影响的 *serialize*(*Te*) 并清空其 *BG* 集。而对于 *STG* 中的事务，则从半提交阶段进入完成提交状态，实现全局数据在本地副本的更新，同时也利用这些新信息支持上节中所述的并发控制算法。就这样，移动客户端和中心服务器利用广播循环交互地实现了整个嵌入式移动实时数据库中混合事务的并发控制。

5 正确性证明

OCC-ASTOSR 算法采用绝对时标在乐观控制方法中进行两阶段验证，能正确地解决事务访问数据的冲突，满足并发事务调度视图可串行化，从而保证了嵌入式移动实时数据库的一致性。本节将给出定理及证明。

定理1 如果事务 *T* 会被移动客户端局部验证算法重启，那么假如不采用两阶段验证提交方式，它也一定会在中心服务器的全局验证中失败。

证明：在移动客户端事务重启有3种情况。假如不进行两阶段验证，而把它直接提交给中心数据库进行验证，则这几种情况下分别有：

(1)移动客户端事务 *T*1 在 *T*1 ∈ *BG*(*T*2) 且 *RS*(*T*2) ∩ *WS*(*T*1) ≠ Φ 的情况下重启，此时说明有 *serialize*(*T*1)<*serialize*(*T*2)，但 *T*2 先执行了写操作又发现它与 *T*1 有读-写冲突；若在中心服务器端，*T*2 先写 *a* 就会使 *RWT*(*a*) 更新为大于原来的 *RWT*(*a*)'，而 *readVersion*(*T*1, *a*) = *RWT*(*a*)' < *RWT*(*a*)，又有 *T*2 和 *T*1 读-写冲突，即存在数据 *b* 使得 *serialize*(*T*1)<*serialize*(*T*2) = *RRT*(*b*)，则在全局验证中 *T*1 失败。

(2)移动客户端事务 *T*1 在 *T*1 ∈ *BG*(*T*) 且 *serialize*(*T*1)< *readVersion*(*T*1, *a*) 的情况下重启，这说明 *T*1 本应读取数据 *a* 最近一次被更新之前的值，但实际却没有；在中心服务器端，就有某事务先更新了 *RWT*(*a*)，使 *readVersion*(*T*1, *x*)< *RWT*(*a*)，但实际却存在 *serialize*(*T*1) > *RWT*(*a*) 的关系，则 *T*1 验证失败。

(3)移动客户端事务 *T*1 在 *T*1 ∈ *BG*(*T*) 且 *serialize*(*T*1) < *RRT*(*a*) 的情况下重启，这说明最近一次读 *a* 操作本应该读取 *T*1 对 *a* 更新后的值，但实际却没有；在中心服务器端，就有 *serilize*(*T*1) 小于上一次广播的 *RRT*(*a*)，那必然有 *serialize*(*T*1)<*lastBC*，则 *T*1 也提交失败。

综上3点，可得移动客户端需重启的事务假如不采用两阶段验证，在中心服务器也一定会失败而需要重启，说明嵌入

(下转第166页)

2006:123-132

[10] Kapoor H K, Josephs M B. Decomposing specifications with concurrent outputs to resolve state coding conflicts in asynchronous logic synthesis// DAC-04:830-833

[11] Agyekum M Y, Nowick S M. A Cycle - based Decomposition Method for Burst Mode Asynchronous Controllers// 13th IEEE International Symposium on Asynchronous Circuits and Systems. 2007:129-142

[12] Nowick S M, Yun K Y, Dill D L. Practical asynchronous controller design// ICCD-92

[13] Marshall A, Coates B, Siegel P. Designing an asynchronous communications chip. IEEE Design and Test of Computers, 1994, 11(2):8-21

[14] Nowick S M, Dean M E, Dill D L, et al. The design of a high-performance cache controller: a case study in asynchronous synthesis. Integration: the VLSI Journal, 1993, 15(3):241-262

(上接第 157 页)

式数据库中的局部验证能正确地检测并解决事务冲突，减轻了中心数据库和网络的负担。

引理 1 设 Ti 和 Tj 是由 OCC-ASTOSR 算法产生的调度 S 中的两个已提交的事务，若串行化图 $SG(S)$ 中有边 $Ti \rightarrow Tj$，则通过 OCC-ASTOSR 算法调整后，必有 $serialize(Ti) < serialize(Tj)$。

证明：根据 2 节的可串行化理论，事务串行化图 SG 中有边 $Ti \rightarrow Tj$ 可能存在以下一种或多种冲突，现证明该引理在各种冲突下均成立即可：

(1) $write(Ti,x) <_S read(Tj,x)$。若 Ti 先提交后 Tj 再读取 Ti 改写过的数据，则按绝对时间的顺序自然有 $serialize(Ti) < serialize(Tj)$；若 Tj 先验证，要产生这种冲突必须存在另一事务 T' 将 Ti 加入 $BG(T')$ 使其时序提前至 Tj 读 x 之前，在 Tj 提交后即得到 $serialize(Ti) < RRT(x)$，而据算法 $Ti \in BG(T')$ 且 $serialize(Ti) < RRT(a)$ 时 Ti 必须重启，也就是说这种情况不存在成功提交。故该冲突只有 Ti 先验证时存在，其满足 $serialize(Ti) < serialize(Tj)$。

(2) $read(Ti,x) <_S write(Tj,x)$。若 Ti 先验证提交，算法不会更改两者的 $serialize$ 值，按验证时间的绝对顺序自然就有 $serialize(Ti) < serialize(Tj)$；若 Tj 先验证，Ti 将被加入到 $BG(Tj)$ 中，如 Ti 不被重启，则有 $serialize(Ti) = serialize(Tj) - \varepsilon < serialize(Tj)$。所以这种情况下总会满足 $serialize(Ti) < serialize(Tj)$。

(3) $write(Ti,x) <_S write(Tj,x)$。第 2 节已经说明过算法将调整使连续两个相邻的写操作之间不存在读操作，且只有最后一个写操作有效。也就是说这种情况总能转化为以上两种情况。

定理 2 通过 OCC-ASTOSR 算法产生的事务调度 S 是可串行化的。

用反证法证明：若 $Ti \rightarrow Tj$ 是 $SG(S)$ 的一条边，则必有冲突操作 $op(Ti,x) <_S op(Tj,x)$，其中 op 为 $read$ 或 $write$，由以上引理可得必有 $serialize(Ti) < serialize(Tj)$。假设 $SG(S)$ 中存在任意一个环 $T1 \rightarrow T2 \rightarrow \cdots \rightarrow Tn \rightarrow T1$，则可推出 $serialize(T1) < serialize(T2) < \cdots < serialize(Tn) < serialize(T1)$，即 $serialize(T1) < serialize(T1)$，显然产生了矛盾。故假设不成立，即 $SG(S)$ 中不存在环，所以 S 是可串行化调度。

结束语 和传统数据库系统相比，在嵌入式移动实时数据库系统中，事务管理的目的除了保证事务的逻辑正确性和执行结果的有效性，还须满足事务实时性和移动性。通过和 OCC-Wait，OCC-PVTO 等并发控制策略进行的模拟对比实验表明，OCC-ASTOSR 在影响嵌入式移动实时事务并发控制性能的主要指标事务（如重启率），以及系统吞吐率和事务平均响应时间等方面具有较好的性能。下一步的工作主要有：

基于乐观并发控制的策略在事务冲突率高的情况下效果不是很好，基于锁的方法虽受到无线网络环境制约但也能在适当条件下改善系统性能，如何确定系统环境的临界点，如何定量评估锁和乐观方法在临界条件下对系统的影响，是将来工作中的难点问题。

目前的两阶段验证提交在移动客户端有完善的前向验证控制，但中心服务器上的后向验证则并非非常严格，以致仍然会存在少许不必要的事务重启。如何进一步改进，也是将来需要考虑的。

参 考 文 献

[1] 覃飙. 分布式实时数据库并发控制和提交处理策略. 博士学位论文. 华中科技大学, 2003

[2] Lee V C S, Lam K W, Son S H. On Transa-ction Processing with Partial Validation and Timestamp Ordering in Mobile Broadcast Environments. IEEE Transaction on Computers, 2002, 51(10):1196-1211

[3] Thomas S, Seshadri S, Haritsa J R. Integrating Standard Transactions in Real-time Database Systems. Information Systems, 1996, 21(1):3-28