数据库中长事务解决方法

王先平[1] 李双庆[1] 张永芬[2]

(重庆大学计算机学院 重庆 400030)[1] (重庆信息工程专修学院计算机科学系 重庆 402160)[2]

摘 要 争用公共资源在数据库系统的长事务中应用非常广泛,如果处理得不好,当客户端的数量比较多并且数据量非常大时,往往由于争用公共资源占用时间较长而且长事务引起并发度较低,因此会导致客户端超时不能成功执行。本文正是在这种情况下提出了用争用公共资源与事务更新分离的解决方案,并将这一方案应用到一个实际的信息系统升级中,成功地解决了以前程序出现的问题,在实际执行的效率上取得了比较好的效果。

关键词 事务,DBMS,长事务,数据库系统

1 引言

随着网络、信息、数据库技术的进一步发展,信息技术应用非常广泛,原来单机程序已不能适应现代信息系统的需求,现代信息系统是面向多用户的,因此,这些用户必须共享数据,最终造成多用户对共享数据的争用。有些商业逻辑是一个整体,这就要求在操作时必须看作一个整体来运作,于是采用事务来处理以保证数据库的一致性和完整性[1,2]。

在诸如大型实时交易这类数据应用系统中,数据库应用程序的设计除了考虑系统安全性,还要考虑服务器响应速度和数据完整性等多方面的问题。例如,像铁路客票、学校学生信息系统等基于C/S(Client/Server,客户机/服务器)环境的实时交易系统,系统死锁是不容许的,有时系统虽然没有死锁,但由于数据量大并且长事务处理占用时间较多,导致效率低,反应非常迟缓,有时经常出现系统超时而发生错误,有时因系统响应过于缓慢,让"客户"等待时间过长并在社会中造成不良影响。然而,通过高性能并发控制保证事务一致性和完整性[3,4]是一个技术难题。当企业应用程序以长事务[5,6]的方式对某一共享数据进行修改时,底层系统平台将对这一共享数据上锁,但数据量比较大且用户数目较多时,由于一个客户端程序更新的时间较长,另一些客户端等待的时间较长,从而引起客户端超时失败,因此,必须想办法解决这个问题。当然,一方面可以从硬件上提高性能之外,另一方面要从程序自身来提高软件的效率。因此,我们在数据库应用程序争用共享资源且长事务更新性能优化方面进行了深入研究工作,并将其运用于一个学校的信息系统升级中,并取得了良好的效果。本文先进行问题描述,然后阐述现有的解决方法,最后重点介绍将争用共享资源与事务更新分离方法来解决出现的问题。

2 问题的描述

在做数据库应用程序时,经常会碰到一种情况,那就是主明细表数据插入问题,同时可能还附带有其它的商业逻辑的更新。但在数据的插入时,首先必须去争用主表中某个共享资源,争用到共享资源共并作一定的处理后,然后才是主从表和其它的商业逻辑同时更新,这是一类典型的数据库中多表更新问题,可以采用事务处理保证数据的一致性和完整性[7,8]。假设多个用户还去访问主表中的共享资源时,在能抢到主表中某个共享资源时才能进行后面的操作,问题就变得复杂多了,如果不采用有效的办法来解决,当数据量比较大并且客户端较多时,往往由于某个客户端执行长事务的时间较长,导致其它客户端超时而不成功,最终对用户来说表现系统失败。

下面描述一类问题:假设有关系数据库中的一类二维表(DataSet)中主明细表:主表(mainid,name,……)、明细表(……,mainid),主表与明细表是一对多的关系,明细表中的mainid是主表的外码。要求对主明细表进行数据的插入,有可能还附带其它商业逻辑数据的更新,并且还要求主表中的mainid必须进行自动统一编码,并按某种意义上的连续。

3 现有的处理方法

这是一类典型的数据库中事务更新的例子,下面举两个典型的处理方法。

3.1 两种常见的处理方法

王先平 硕士研究生,主要研究方向为计算机应用;李双庆 博士,副教授,主要研究方向为电子商务、计算机应用、嵌入式系统、计算机网络与通信;张永芬 助教,主要从事数据库相关方面的工作。

方法一：将读取主表中 mainid 的最大值，根据用户定义的规则产生新的待插入的 mainid 值，然后写主表、写明细表和其它相关的商业逻辑表，整个处理过程作为一个事务来提交，如图 1 所示。

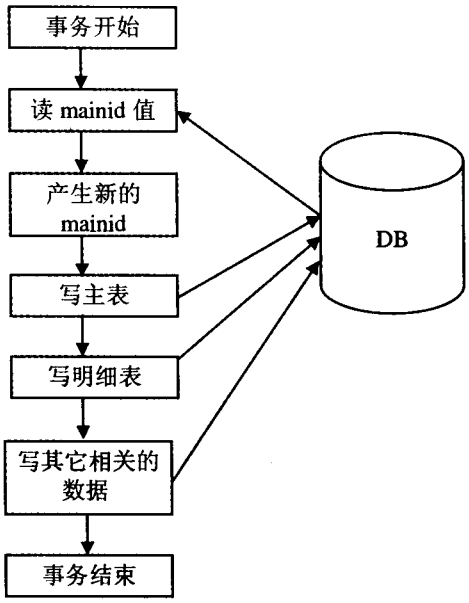

图 1　整个处理过程作为一个长事务

用伪代码实现的片段如下：

```
Try
  Begintransac; //事务开始
    Read(max(mainid));//从主表中读取 mainid 的最大值
  //根据规则产生新的待插入的 mainid 值
    Generate(new mainid);
    Write(主表); //写主表内容
    Wirte(明细表);//写明细表内容,即主表的从表
  //写其它相关的企业逻辑表的内容
    Write(其它相关的企业逻辑表);
  Committransac; //事务提交
Except
    Rollbacktransac;//如果更新不成功,事务回滚
  Raise;//抛出异常
End;
```

方法二：就是将先读取主表中的 mainid 的最大值，然后根据规则产生待插入的 mainid，而后将写主表、写明细表、写其它商业逻辑数据一起作为事务提交，如图 2 所示。

用伪代码的片段实现如下：

```
Read(max(mainid)); //从主表中读取 mainid 的最大值
//根据规则产生新的待插入的 mainid
Generate(new mainid);
Try
  Begintransac //事务开始
  Write(主表);//写主表内容
  Wirte(明细表);//写明细表内容,即主表的从表
  //写其它相关的企业逻辑表的内容
  Write(其它相关的企业逻辑表);
  Committransac; //事务提交
Except
  Rollbacktransac;//如果更新不成功,事务回滚
  Raise;//抛出异常
End;
```

3.2　已有方法的评价

这两种方案都采用了事务更新机制保证了数据的一致性和完整性，在单用户下不会存在问题，但在多用户操作时可能会出现问题：

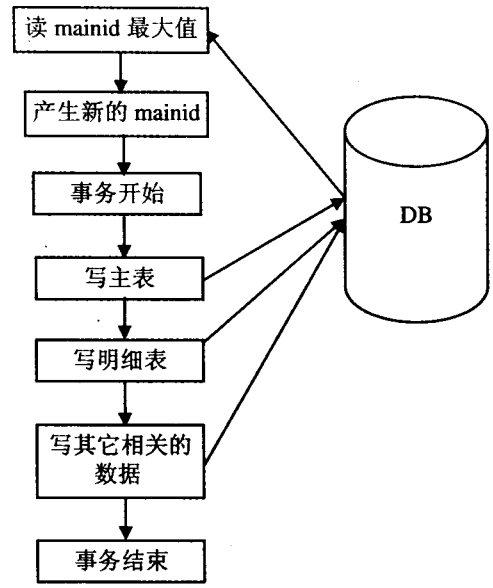

图 2　数据更新过程作为一个事务

对于方法一，当数据量特别大时，由于把整个处理过程作为一个长事务处理，这样处理时间会比较长，在多个用户同时操作时，有些客户端会因为超时而出错。

对于方法二，相对于方法一而言，事务处理的时间相对短些，但也会出现问题。由于先取得主表中的 mainid 的最大值，按照 DBMS(数据库管理系统)的三级封锁协议，没有显式事务，就是用隐式事务，读完主表中数据后马上释放主表资源。由于是多个用户同时在操作，那么其它很多客户端就可以对主表进行操作，可能读到同样的数据，因此，很多用户读到的是一个相同的 mainid 的最大值，进行相应的规则处理后同样产生相同的新的 mainid 值，这时大家都以同样的 mainid 值进行插入主表操作，由于主表中 mainid 是主关键字，是唯一索引，唯一索引要求其值不能重复。因此，插入数据时违反了主关键字唯一性规则，DBMS 会拒绝存取，马上会报告错误，客户端越多，客户端不能成功的机率就越大。

当然也有人这样处理，在第二种方法的基础上再加一个循环，就是本次不成功，马上重新循环再试，如果循环一定次数还是不能成功，那就该退出。但这样情况与前两种差不多，可能从某种情况下可以减少客户端因主键重复而马上报告出错的机会，但是当客户端数量比较多时，有很多客户端可能等很长时间后还是要报告错误，即出错的机率还是比较明显。

出现错误的主要原因是：由于数据量比较大，并且是多个用户对共享数据资源(主表中的 mainid 的最大值)进行争用，导致长事务更新时间较长，一些客户端执行不成功。

4 基于争用与大量数据更新分开的解决方案

4.1 解决方法

解决思想：将争用共享资源（主表中 mainid 的最大值）与大量数据更新分开的办法，即分两步走：第一步是先争用主表中的 mainid 的最大值，将抢到的待插入的主表的 mainid 值存放在另一个临时表中；第二步根据争用到的 mainid 值去进行相应的插入更新。根据这一设计思想，增加两个临时二维表，一个表用来存放已产生的主表中 mainid 的最大值，另一表用来存放用户所抢到的 mainid 值，即当前已产生过的最大 mainid 值表：Maxmainid(currentmainid)，currentmainid 表示主表中已产生的所有 mainid 的最大值；用户所抢 mainid 情况表；usermainid (username，robmainid，robtime)，username 代表用户名，robmainid 代表用户所抢到的 mainid，robtime 代表抢到 mainid 的时间（主要用于处理长时间没有用此 mainid 值，就应该释放此资源，供其它用户用）。其处理流程如图 3 所示。

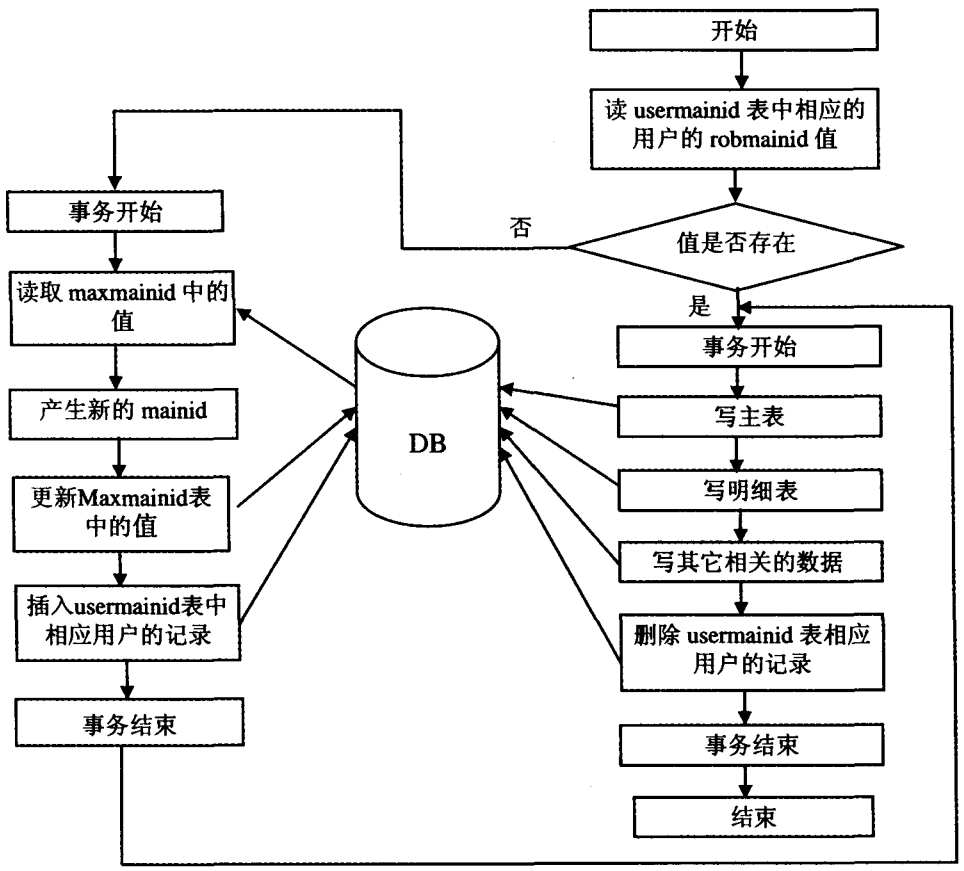

图 3 基于争用与大量数据更新分开处理方法

4.2 伪代码实现部分：

（1）先编一个函数 F_robid(用户名)，返回已抢到的 currentmainid 值，用来实现抢夺 maxmainid 表中的 currentmainid 值。

Function F_robid(用户名)：mainid 对应的数据类型；

定义变量 resultmainid；

```
Begin
  Try
  初始化变量(Resultmainid);
Try
  Begintransac;//事务开始
    //读取临时表 currentmainid 表中的值
    Read(maxmainid(currentmainid));
    //根据 currentmainid 和规则产生新的主表的 mainid 值
    Generate(new mainid);
  //将新产生的 mainid 值更新回 maxmainid 表中，表此值已被占
    Update(maxmainid(currentmainid));
    //将用户抢占的 mainid 值写入临时表 usermainid 中，供更新时用
      Insert(usermainid 表中相应记录);
    Committransac;//事务提交
    //将新的 mainid 值赋值给 resultmainid 变量
    Resultmainid=new mainid;
  Except
    Rollbacktransac;//如果提交不成功，事务回滚
    Raise;//抛出异常
  End;
Finally
  Return(resultmainid);//函数返回值
End;
End;
```

当然，也可能存在抢占不成功的现象，在实际中也可以加入循环，让系统多去抢几次，由于执行时间很短，可以尽量解决抢占共享资源超时而不成功的现象。

（2）主要的处理程序伪代码如下：

```
Procedure updatebuttonclick;
Var robno;//定义一个 mainid 对应类型的变量;
Begin
//从临时表 usermainid 中读取相应用户的记录赋值给 robno
  robno=Read(usermainid(用户名));
  //如果用户没有可用的 mainid 值,就要去调用函数 Robid
抢夺一个值
  If not exist(robno)then
  Robno=F_robid(用户名);
  //判断所抢 mainid 是否存在,如果存在,则可以顺利地更
新
  If exist(robno)then
    begin
      Try
        Begintransac;
          Write(主表);//写主表内容
          Wirte(明细表);//写明细表内容,即主表的从表
  //写其它相关的企业逻辑表的内容
          Write(其它相关的企业逻辑表);
  //如果此资源已用,就删除此用户占用的资源
          Delete(usermainid(用户名));
        Committransac;
      Except
        Rollbacktransac;//如果提交不成功,事务回滚
        Raise;
      End;
    End;
End;
```

当然,本方案有可能出现一种情况,那就是当某个用户产生一个 mainid 后,由于某种原因他不再用本系统了,那产生的这个 mainid 就可能在主表中不存在,导致主表中的 mainid 不连续,这是不允许的。可以采用一种措施来解决这个问题,那就是在用户登录系统时增加一个功能,让用户登录时去检查临时表 usermainid 中的资源的利用情况,如果自己抢占 mainid 已用完并且别人抢占的 mainid 长时间没有用,这时将别人长时间没有用 mainid 占用过来作为己有,就可以解决那些抢占到了资源,而又长期没有用的资源。

4.3 方案分析

本方案是在争用主表中的 mainid 共享资源,且在长事务的环境下提出问题的一种解决方案,采用了抢占共享资源和长事务更新分离的办法,相对一般的解决方法,具有以下特点:

(1)效率较高。因为数据量比较大,原来产主表中新的 mainid 需要从几十万,甚至上百万的数据中查找出最大值,然后再由此产生新的 mainid,新方案是从另一个临时表中进行操作,临时表中只有一条记录,它表示当前已产生主表中 mainid 的最大值,从访问数据表的数据级上低了很多,因此效率较高,典型以牺牲空间换取时间的方法。

(2)并发度更高。由于 DBMS 并发的基于单位是事务[9],原来的只有一个事务,并且是长事务,新方案将其分为两个事务,分解后的事务相对原来的长事务来说要小得多,因此从理论上说整个程序并发度更高。

(3)更新出错代价较小。本方案采用抢占共享资源与更新分离的方法,只有抢占到了主表中的 mainid,才有机会去更新其相关的内容。也就是说,只要能更新,所产生主表中待插入的 mainid 已经是己有,肯定是每个用户所抢的 mainid 值会各不相同,就不像以前那样,有可能多个用户产生了相同的 mainid 值;另外,如果在以后的更新中出现了错误,不会影响已抢占到的主表的 mainid 值,因此新方案出错的代价相对原来方案要小得多。

(4)出现冲突的机会较小。本方案出现超时的情况明显减少了,因为抢占主表待插入的 mainid 值时是从临时表中去抢占的,临时表 maxmainid 数据量很小,且事务很短,并发度就较高,因此出现冲突的机会较小。

(5)从总体上看,主表中的 mainid 是连续的,达到了满意效果,但从局部来说,可能不连续。比如有个用户成功地抢点到一个 mainid 值,但由于后面更新时有个错误而没有成功,而这个用户如果要继续用,下一次肯定可以利用已有的资源,然而如果此用户不用系统了,在短时间看这个值就不连续。但在解决方案中有个补救措施,任何一个用户在登录时都要去检查临时表 usermainid 有没有占用资源并且超过时间没有用的,如果有,这个用户就会利用别人已占的资源,这样可以达到总体上连续的。

5 实际应用

在某高校学生信息系统收费子系统升级中,用户要求一个学生可以多次缴费,一张收据表下有多种明细费用,也就是收据与费用是一对多的关系,同时在学生缴费时伴随着要去更新历年缴费的费用总表,还有就是在缴住宿费时,就要去更新学生相应的住宿信息并且要求收据编号系统统一编号并且必须连续,学校规模较大,数据量大,其数量级在百万以上。这个问题就类似以上分析的问题,实际上就是一个主明细表的更新,还附带其它相应表的更新操作,属于争用公共资源(收据编号)的长事务更新问题。

将基于争用共享资源与大量更新分开的解决方案运用到学生信息系统收费子系统升级中,成功地解决由于学校发展规模增大,数据量增加,客户端不断增多出现的更新错误。

结束语 本文在争用共享资源和长事务更新的条件下,分析了现有的解决方案面临的问题和原因,提出了基于争用共享资源和更新分离的解决方法并给了部分伪代码。本方案运用到一个高校的学生信息系统升级中,成功地解决了随着学校规模增加,导致数据量不断增大、客户端增加时由于争用共享资源长事务更新而出现的更新效率低,并且经常出现抢占共享资源冲突,导致超时不成功的现象。由于争用共享资源且长事务的更新问题在数据库系统中应用非常广泛,本文所提出争用共享资源与长事务更新分离的方法对解决这类问题具有一

(下转第 110 页)

转发、浏览、归档等管理功能，为用户提供实现页面定制功能的应用，可使用户能编辑、定制自己的门户网站页面，能实现其他已有系统有效的关键数据的共享。系统整个功能如图 3 所示。

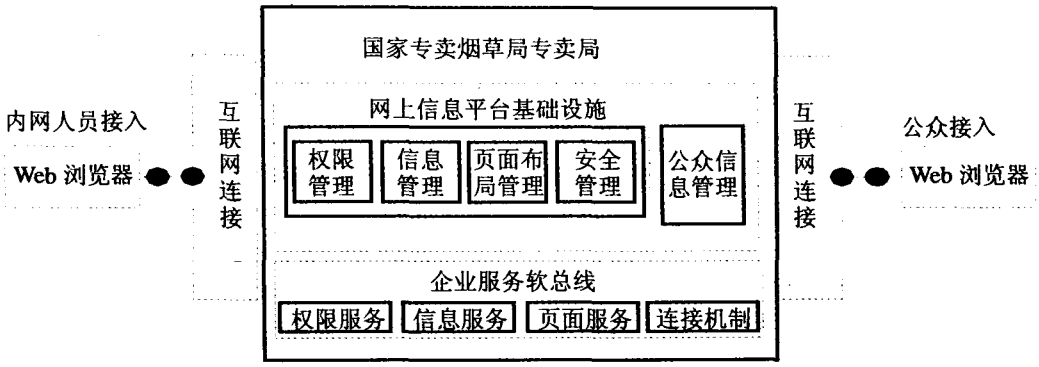

图 3

1）权限管理

为系统中的不同用户分配角色和权限。用于维护信息系统中与访问控制相关的信息，包括数据实体(用户、用户组、权限组(角色)、权限和系统页面组)信息的添加、修改和删除，和数据实体间关联信息的添加与删除。在维护访问控制信息的时候，需要遵循基于角色访问控制模型的角色约束规则。这样降低了模型复杂度，简化了角色层次结构。

2）信息管理

信息管理实现网站内容的更新与维护，提供在后台输入、查询、修改、删除各新闻类别和专题中的具体信息的功能，选择本信息是否出现在栏目的首页、系统的首页等一系列完善的信息管理功能。具体包括以下功能：增添、修改、删除各栏目信息(包括文字与图片)的功能。信息类信息的管理，分新增，发布/取消发布，转发，浏览，归档等功能。

3）页面布局管理

布局管理：对布局进行增删改，布局是表现将一个页面矩形分割为多个区域，以及各个区域的大小和位置。

模块管理：模块是指布局中的每一个布局单元对应哪种信息类型或是要实现哪种功能，采用何种方式显示。

4）安全管理

根据系统信息对用户访问进行验证，通过用户名和所访问的页面信息对资源访问进行验证。通过 CA 保证可靠性、系统安全性。

结论 本文提出的解决策略立足于系统自身对信息整合平台的功能要求，结合内部的实际情况，以面向对象的思想和方法、WebService 技术，以及企业服务软总线等技术进行了系统设计，该设计基本解决了开放性、灵活性、可移植性等问题，满足了专卖司对信息整合平台的需求，解决了现有系统的界面集成、单点登录、物理分布等整合问题，整个系统可根据项目的需要可进一步完善和扩充。同时，该平台的实现，有利于提高专卖司业务管理水平，促进管理现代化，有效降低成本，加快技术进步，增强市场竞争力，并为未来专卖系统信息化打下坚实的基础。

参 考 文 献

1 刘罡. 烟草专卖司信息整合平台研究及应用:[重庆大学硕士论文]. 2005.4
2 邵维忠，杨芙清. 面向对象的系统分析. 清华大学出版社，2003
3 烟草行业信息化建设统一技术平台要求. 国家烟草专卖局，2003
4 周书，李洁，金海滨，等. 系统集成与项目管理[M]. 北京：科学出版社，2004

(上接第 74 页)

定的参考价值。

参 考 文 献

1 郑阿奇. SQL Server 实用教程[M]. 北京：电子工业出版社，2005，1：323～328
2 王能斌. 数据库系统原理[M]. 北京：电子工业出版社，2000，1：86～94，140～162
3 BarghoutiN S, Kaiser G E. Concurrency control in advanced database application[J]. ACM Computing Surveys, 1991, 23(3): 269
4 Bhargava B. Concurrency Control in Database Systems[J]. IEEE Transactions on Knowledge and data engineering, 1999, 11(1)
5 罗惠琼，李伟岸. 基于长事务处理的分布式框架设计思路[J]. 计算机应用，2004.12
6 汪锦岭，金蓓弘，李京. 一种基于强可有序化标准的长事务调度算法[C]. 计算机研究与发展，2005.1355～1361
7 李德军. 在多层数据库结构中利用事务处理保证数据的一致性[J]. 计算机系统应用，2006.3
8 许中卫，周勇. 在 Delphi 中用事务保持数据库一致性和完整性[J]. 微机发展，2000.6
9 王珊，陈红. 数据库系统原理教程[M]. 北京：清华大学出版社，2004，11：161～179