第 34 卷 第 16 期 计 算 机 工 程 2008 年 8 月
Vol.34 No.16 Computer Engineering August 2008
·软件技术与数据库· 文章编号：1000—3428(2008)16—0060—03 文献标识码：A 中图分类号：TP311.13

基于关联事务的移动数据库冲突处理算法

张晓丹，何 锐，牛建伟
(北京航空航天大学计算机学院，北京 100083)

摘 要：移动数据库系统由于自身的特点采用乐观复制机制。该文引入关联事务的概念，提出关联事务划分算法(UTDA)及冲突处理算法(CRA)。UTDA 算法将移动终端在本地提交的移动事务划分成关联事务，把关联事务作为数据同步和冲突处理的基本粒度。实验结果表明，UTDA 算法满足事务执行的原子性和串行性，提交时间比传统事务提交时间减少了 2/3，为移动数据库系统的冲突处理提供了可行的解决方案。

关键词：移动数据库；冲突处理；关联事务

Conflict Reconciliation Algorithm Based on Union-transaction in Mobile Database

ZHANG Xiao-dan, HE Rui, NIU Jian-wei
(Department of Computer Science, Beijing University of Aeronautics and Astronautics, Beijing 100083)

【Abstract】 Mobile database system adopts optimistic replication mechanism because of its characteristics. This article introduces the concept of union-transaction, presents Union-Transaction Division Algorithm(UTDA) and Conflict Reconciliation Algorithm(CRA). UTDA algorithm divides mobile transactions in mobile terminal into union-transactions, makes union-transaction as basic granularity of data synchronization and conflict reconciliation. Experimental results show that UTDA algorithm satisfies atomic and serializability of transaction execution. The cost of commit time falls by two-third approximately. It provides a kind of feasible scheme for conflict reconciliation of mobile database system.

【Key words】 mobile database; conflict reconciliation; union-transaction

1 概述

一个理想的移动数据库系统应该达到 4 个目标：可用性与可伸缩性，移动性，串行性和收敛性[1]。这 4 个目标决定了理想的移动数据库系统应该是一种乐观复制方式，即系统允许移动终端在网络断接的情况下根据本地缓存执行事务(即移动事务)操作，造成系统短暂的不一致，重新连接时进行数据的同步处理，使系统重新收敛于一致性状态。上述过程称为数据同步过程，其中需要解决的关键问题是乐观复制所导致的冲突处理。

冲突处理问题一直是移动数据库领域的重点研究问题。文献[1]提到的两级复制机制要求移动终端提交的移动事务在基节点重做，增加了固定主机的负担，浪费系统资源。文献[2]提到的多版本方法需要保存数据的多个版本，增加存储空间。文献[3]以事务为单位的冲突算法将事务的串行性要求放宽，但是需要特定的冲突处理函数和代价函数。BAYOU系统需要人工依据系统的具体特点来定制冲突处理函数，影响系统的通用性和适应性[4]。

分析以上几种策略可知，合理解决冲突处理问题需要选择合适的同步粒度，冲突处理算法尽可能减少与系统本身的关联。为了克服目前移动数据库系统冲突处理存在的缺陷，在关联事务概念的基础上提出关联事务的划分算法(Union-Transaction Division Algorithm, UTDA)和冲突处理算法(Conflict Reconciliation Algorithm, CRA)。

2 UTDA 关联事务划分算法

2.1 同步粒度

同步粒度是移动数据库数据同步的基本单位。根据现有的研究可以将同步粒度归纳为 3 类：元组，操作，事务。不同的同步粒度有各自的优点和缺点，下面分别介绍。

选择元组作为同步粒度是大多数商用移动数据库的做法。这种方法快速简单，移动终端甚至不需要自己维护事务执行，但是该方法依赖于语义，并且以数据为同步粒度，破坏了事务的原子性。

操作级同步是以单个数据库操作作为同步粒度。这种方式下移动终端需要保存执行的 SQL 语句，不需要在固定主机重新生成，减轻了同步服务器的负载。但在冲突处理过程中，一个 SQL 语句可能涉及到多个元组，增加冲突检测的难度。

最后一种同步粒度是事务，选择事务为同步粒度要求移动终端的冲突处理能力增强。这种同步粒度满足原子性，但是回滚次数较多，影响系统执行效率。

本文选择事务作为同步粒度并在此基础上进行改进，并且引入关联事务[5]的概念。目前移动终端的处理能力不断提高，在移动终端提交本地事务可以减轻固定主机的负担。通过对关联事务的操作结果进行合并，不需要保存中间结果，节省存储空间。终端只需要上载关联事务操作结果和相应的时间戳(或版本值)就可以完成数据同步，所以网络中只需传输较少的数据包。以关联事务为同步粒度的冲突处理策略确保事务执行的正确性、串行性和原子性。

2.2 相关定义

定义 1(事务读集) 事务 T 的读集 $ReadSet(T)$ 为其读过的

作者简介：张晓丹(1983－)，女，硕士，主研方向：移动计算，移动数据库；何 锐，博士；牛建伟，副教授、博士

收稿日期：2007-08-27 **E-mail：**zhangxiaodan83@163.com

数据对象的集合。

定义 2(事务写集) 事务 T 的写集 $WriteSet(T)$ 为其修改过的数据对象的集合。

定义 3(事务结果集) 事务 T 的结果集 $ResultSet(T)$ 为 T 写集中各数据元素对应的值的集合。

定义 4(关联事务) 事务的关联是指存在 2 个或者 2 个以上事务重复操作某一个或者多个相同的数据对象。如果出现这种情况，则把这些相关事务称为关联事务。2 个事务 T_1 和 T_2 是关联事务，需要满足下面 2 个条件：

(1) $WriteSet(T_1)\cap WriteSet(T_2)\neq\varnothing$ (2 个事务的写集相交)

(2) $ReadSet(T_2)\cap WriteSet(T_1)\neq\varnothing$ (T_2 的读集与 T_1 的写集相交)[5]

记为关联事务 $UT_1=\{T_1,T_2\}$，由此可以类推到多个事务之间的关联。

假设有 n 个串行执行的事务 T_m $(m=1,2,\cdots,n,n\geq 2)$。其中，任意 2 个事务 T_i 和 T_j $(i,j=1,2,\cdots,n,i<j)$，满足(1)、(2)这 2 个条件，则 n 个事务 T_m $(m=1,2,\cdots,n,n\geq 2)$ 关联。多个事务关联就是要求其中任意 2 个事务都要有关联关系，记为 $\{T_1,T_2,\cdots,T_n\}$。2 个事务的关联是多个事务关联的特殊情况。

定义 5(关联事务读集) 关联事务 UT，包括事务 $T_k(k=1,2,\cdots,n)$。其读集 $UnionReadSet(UT)=ReadSet(T_1)\cup ReadSet(T_2)\cup\cdots\cup ReadSet(T_n)$。

定义 6(关联事务写集) 关联事务 UT，包括事务 $T_k(k=1,2,\cdots,n)$。其写集 $UnionWriteSet(UT)=WriteSet(T_1)\cup WriteSet(T_2)\cup\cdots\cup WriteSet(T_n)$。

定义 7(关联事务结果集) 关联事务 UT，包括事务 $T_k(k=1,2,\cdots,n)$。结果集 $UnionResultSet(UT)=ResultSet(T_1)\cup ResultSet(T_2)\cup\cdots\cup ResultSet(T_n)$。

定义 8(n 元关联事务) 对于关联事务 UT，如果 UT 中包括 n 个事务，则称关联事务 UT 为 n 元关联事务。

2.3 关联事务划分算法

UTDA 算法的思想是首先根据事务之间的关联性找到二元关联事务，去掉不满足串行性要求的事务集合，在剩下的集合中根据关联性继续合并，直到不能合并出关联事务。最后在剩余的集合中找到数目最少的，且可以覆盖所有事务的关联事务集合。

上面提到的寻找覆盖所有事务的最小闭包问题为集合覆盖问题，其相应的判定问题推广到NP完全的顶点覆盖问题，因而也是NP难度的[6]。同时考虑到得到最优解的时间复杂度和空间开销问题，本文选择一种比较简单的集合覆盖算法，关联事务仅合并到二元关联事务。其时间复杂度为 $\theta(n^2)$。具体算法描述如下：

关联事务划分算法(UTDA)：

涉及到的变量：

tranArray[] //初始事务-输入
tranNum //事务个数
CorreTran[] //所有关联事务-输出
indexArray[] //标记每个事务是否成为关联事务
tranPtr //事务指针
nextPtr //事务指针

算法步骤：

(1)tranPtr 指向第 1 个事务
(2)while tranPtr 不为空
(3)nextPtr 指向 tranPtr 下一个事务
(4)while nextPtr 不为空
(5)if 2 个事务有关联关系
(6)then 将 2 个事务保存在关联事务中；将 indexArray 中事务对应的标识设置
(7)tranPtr 指向 tranPtr 的下一个事务；nextPtr 指向 tranPtr 的下一个事务
(8)else nextPtr 指向 nextPtr 的下一个事务
(9)endif
(10)endwhile
(11)endwhile
(12)for 遍历 indexArray[]
(13)if indexArray 没有标识
(14)then 将该事务添加到关联事务中；设置标识
(15)endif
(16)endfor
(17)for 遍历所有关联事务
(18)得到该关联事务的关联读集，关联写集，相应的关联结果集
(19)endfor

关联事务和普通事务相比，减少了空间开销。因为关联事务可以合并相关联的操作(步骤(18)将具有相同写集的事务结果进行合并)，所以不需要保存事务提交的中间结果，但是提交效果相同，节省时间开销。如果本次操作修改了前面操作涉及到的某个数据，用本次操作的结果覆盖前面操作的结果，前提条件是后一个操作读到的是正确的数据。最终数据库的更改通过提交结果集数据来实现。

2.4 正确性

关联事务以事务为基本同步粒度并在此基础上进行修改，它具有事务粒度的特点，所以满足原子性。关联事务将相同数据的操作结果合并，省略中间结果的保存，最终结果仍然保证事务执行的正确性。例如：

update *salary* = 1 234 where *id* = 11
update *salary* = 4 264 where *id* = 11
update *salary* = 5 678 where *id* = 11

对于本例只需要保存 *salary* = 5 678， *salary* = 1 234 和 4 264 为中间结果，不需要保存。

至于事务的串行性，只需要证明即使对事务进行合并，关联事务仍然满足串行性。假设存在事务 T_i 和 T_j 是关联事务，假设不满足串行性原则，则存在事务 T_k 满足 $i<k<j$ 并且 T_k 和 T_i 是关联事务，从算法步骤(6)和步骤(7)可以看出，如果 T_k 和 T_i 是关联事务，保存该关联事务后，T_i 指向下一个事务，T_k 指向 T_i 的下一个事务，不存在 T_i 和 T_j 是关联事务的可能。2 个事务指针在以后的判断中有可能指向 T_j $(i<j)$，但是不会指向 T_i。所以关联事务仍然满足串行性。

3 冲突处理算法

关联事务在移动终端提交，只需要提交操作结果，移动终端将关联事务的读集、写集和结果集上载给同步服务器。同步服务器对接收到的关联事务进行冲突检测。如果组成关联事务的某个事务提交失败，则执行回滚操作。通常来说整个关联事务都需要回滚。这样导致本来可以提交的事务也回滚了，降低了系统效率。CRA 算法的思想是当某个关联事务提交失败，则需要回滚部分事务，从出错事务开始回滚。

例如关联事务 UT 包括 $\{T_1,T_2,T_3\}$。其中，

T_1： update *salary* = 3 252; update *name* = bobby;

T_2： update *name* = sunny; update *age* = 56;

T_3： update *salary* = 637 3; update *age* = 25

如果 *name* 元组出错，则 T_1 需要回滚，与此相关的 T_2 和 T_3 都需要回滚。如果 *age* 元组出错，则事务 T_2 和 T_3 回滚， T_1 不需要回滚。但是实际上 *salary* 和 *name* 元组都没有出错。所以不需要全部回滚。

CRA 算法的思想为：同步服务器首先查找接收到的关联事务读集中每一项的版本值(或时间戳)，如果符合版本值要求则提交该事务，否则将该项添加到出错集合中。将出错集合发送给移动终端，移动终端根据出错读集查找本地关联事务集合，回滚相关操作。

冲突检测算法：

涉及的结构：

```
struct UT   //关联事务
{
 union ReadSet;   //关联读集
 union WriteSet;   //关联写集
 union ResultSet;   //关联结果集
};
```

涉及到的变量：

UTSet //关联事务集合，包含多个移动终端上载的关联事务集合

false ReadSet   //包含出错读集合

算法实现：

同步服务器操作：

```
for 每个关联事务
   for 关联事务中 union ReadSet 的每一项
      if 有一项不满足时间戳要求
        then 将这一项添加到 false ReadSet; 将对应的数据从 union ResultSet 中去掉; break;
      endif
   endfor
endfor
for 每个关联事务
将union ResultSet写入数据库;
endfor
向每个移动终端发送出错读集 false ReadSet;
```

移动终端操作：

移动终端接收到出错读集 false ReadSet;

寻找与 false ReadSet 相关的本地操作，执行回滚操作;

同步服务器中经过冲突检测可以全局提交的事务，同样可以采用 UTDA 算法，得到关联事务结果集写入中心数据库，节省空间和时间开销。

4 模拟实验与性能比较

4.1 实验说明

实验的目的是分析比较 UTDA 算法和以普通事务为同步粒度的冲突处理算法(CTA)的时间和空间开销。UTDA 算法按照关联事务顺序操作，CTA 算法按照事务形成顺序处理。测试数据包括事务操作和提交数据库的时间，提交事务个数和本地保存所有 SQL 操作结果的空间个数。提交事务的个数影响移动终端上载同步服务器的数据包个数，如果一个事务可以封装为一个数据包，那么提交的事务个数越多，网络传输的数据包个数就越多。测试的事务个数分别为 100, 200, 300, 400, 500，每个事务包括 2 个操作。

4.2 实验结果

实验测试数据随机生成，为了方便测试，暂不考虑出错事务，且操作类型全部为更新操作。实验包括 3 组，第 1 组测试本地提交时间，测试结果如图 1 所示。可以看出 UTDA 算法明显优于 CTA 算法。测试事务个数为 500 时，UTDA 算法提交时间大约是 CTA 算法提交时间的 1/3。并且随着事务增多，特征更加明显。图 2 显示 2 种算法提交事务个数的比较结果。UTDA 算法提交的事务个数大概是 CTA 算法的一半(≈55%)。因为 UTDA 算法根据事务的关联性将事务合并，合并后事务个数明显减少。CTA 算法采用传统的事务执行方法，提交的事务个数为初始化输入的事务个数。图 3 是 2 种算法执行时空间开销的比较结果。CTA 算法需要保存每个事务执行的结果，占用大量的存储空间，而 UTDA 算法只需要保存关联事务执行的最后结果，中间数据对最终结果没有影响。UTDA 算法节省了 27%左右的存储空间。

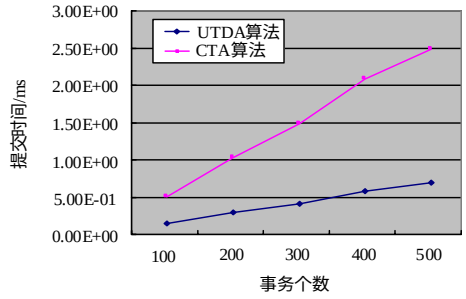

图 1　本地提交时间比较

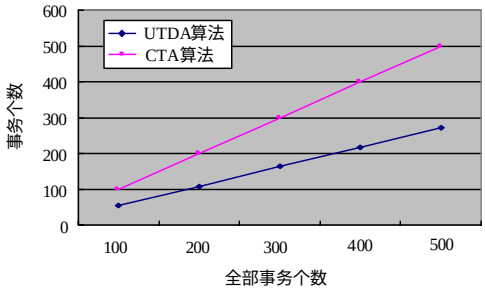

图 2　提交的事务个数比较

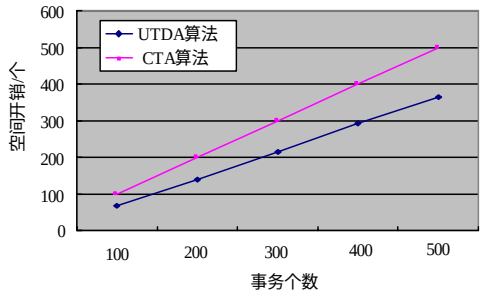

图 3　本地空间开销比较

从上面的实验数据可知，UTDA 算法在空间和时间开销上都优于 CTA 算法。冲突处理算法 CRA 以 UTDA 算法为基础，采用回滚部分操作的方式，降低回滚率。实验结果表明 UTDA 算法和 CRA 算法具有良好性能和应用价值。

5 结束语

本文引入关联事务的概念，将关联事务作为移动数据库系统数据同步的基本粒度。提出 UTDA 算法，该算法满足事务执行的原子性、正确性和串行性。关联事务结果集的合并降低了系统的时间和空间开销。冲突处理算法 CRA 只回滚出错操作，降低事务回滚次数。关联事务的划分影响整个冲突处理过程的时间开销，本文提出的 UTDA 算法本身不是最优算法，希望在今后的研究中找到更好的方法。

(下转第 65 页)