复制的移动数据库系统事务级同步处理策略*

丁治明[1], 孟小峰[2], 王 珊[1,2]

[1](中国科学院 计算技术研究所,北京 100080);
[2](中国人民大学 数据与知识工程研究所,北京 100872)
E-mail: dingzhiming@263.net; {xfmeng,suang}@public.bta.net.cn
http://www.ict.ac.cn

摘要: 同步处理技术是保持复制的移动数据库系统一致性的一项关键技术,但现有的事务级同步处理算法存在着一定的局限性.为了克服这些缺陷,并增强其实用性,提出了一种新的移动数据库同步处理模型——基于双时间印的事务级同步(DTSTLS)模型.DTSTLS 模型采用了一种三级复制体系结构,系统可以直接使用通用的数据库产品作为其数据库服务器,因此具有良好的可扩充性.作为一种异步的多主副本复制方法,DTSTLS 模型允许移动计算机在断连的情况下存取本地副本,从而造成系统短暂的不一致,重新连接时进行冲突检测及同步处理,使系统重新收敛于一致性的状态.此外,通过一种独特的时间印处理策略,DTSTLS 模型减少了通信代价,并降低了资源消耗.实验结果表明,DTSTLS 模型提高了移动数据库系统的资源利用效率,保证了事务调度的可串行性和数据库的一致性.

关 键 词: 移动计算;数据库;事务处理;复制
中图法分类号: TP311 文献标识码: A

数据复制技术是提高移动数据库系统可用性和可靠性的一项关键技术.一个理想的移动复制机制应该达到 4 个目标,即可用性与可伸缩性、移动性、可串行性、收敛性[1].这 4 个目标决定了理想的移动数据复制应该是一种异步的多主副本复制,即系统允许移动计算机在断连的情况下在本地副本上执行事务操作(称这类事务为移动事务),从而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的状态.上述过程称为同步处理过程,其中需要解决的关键问题是异步复制所导致的冲突处理问题.

在移动数据库系统复制方面,人们已经进行了大量的研究,并提出了许多模型与算法,如两级复制算法[1]、容错型定额同意方法[2]、主动复制机制[3]等.然而,这些工作大部分均没有讨论冲突处理问题.某些工作(如 BAYOU[4]系统)虽然研究了冲突处理问题,但需要人工依据系统的具体特点来定制冲突处理函数,因此影响了系统的通用性和适应性.文献[5]提出了一种有效的移动复制冲突处理模型——多版本冲突消解模型.该模型在数据库服务器上管理数据对象的多个版本,并提供了数据快照一级的事务隔离性.根据文献[6]的结论,这一隔离级别不能完全保证事务调度的可串行性.

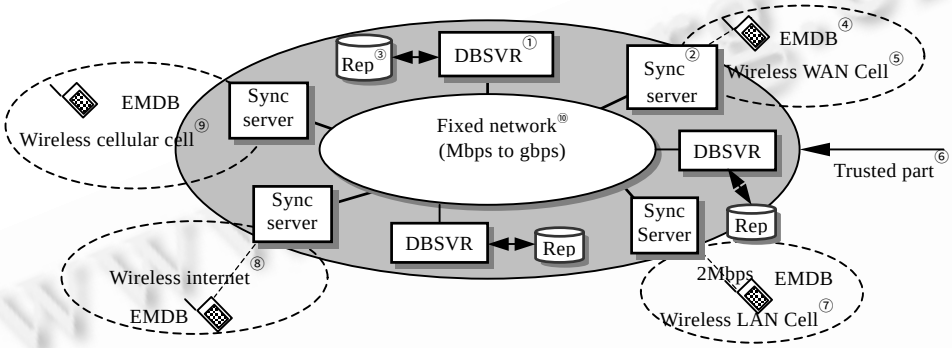
此外,由于复制的移动数据库系统通常需要保存完整的移动事务日志,且在同步处理过程中需要传递大量的数据,因此在实际应用中受到了移动计算环境诸多因素的限制.同时,系统的可扩充性还要求移动数据库系统应能方便地容纳其他主流数据库产品.为了克服目前移动数据库系统同步处理算法中所存在的缺陷,并增强其实用性,本文提出了一种基于双时间印的事务级同步(double-timestamp transaction-level synchronization,简称 DTSTLS)机制,并给出了具体的实现算法.

* 收稿日期: 2001-04-20; 修改日期: 2001-09-18
基金项目: 国家自然科学基金资助项目(60073014);国家 863 高科技发展计划资助项目(863-306-ZD12-12-1)
作者简介: 丁治明(1966－),男,湖北荆州人,博士生,高级工程师,主要研究领域为数据库与知识库系统,移动计算;孟小峰(1964－),男,河北邯郸人,博士,副教授,主要研究领域为数据库与知识库系统,移动计算,web 信息管理;王珊(1944－),女,江苏无锡人,教授,博士生导师,主要研究领域为数据库与知识库系统,移动数据库,数据仓库技术.

本文第 1 节阐述 DTSTLS 同步处理模型的基本工作原理.第 2 节给出 DTSTLS 模型的时间印处理策略.第 3 节描述 DTSTLS 同步处理算法.第 4 节对 DTSTLS 模型的性能进行实验分析并给出相关结论.

1 DTSTLS 同步处理模型

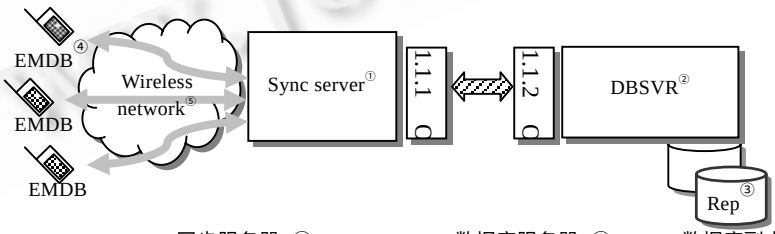
移动数据库系统三级复制模式可用如图 1 所示的结构图表示.在该系统中,可信部分由固定网络以及网络上的固定主机组成.其中固定主机又可分为两类,一类是不带无线通信接口的数据库服务器(DBSVR),它可以是任意的大型数据库系统,如 Oracle, Sybase, DB2, SQL Server 等,每个 DBSVR 上保存数据库的一个完整的副本,另一类是带有无线通信接口的同步服务器(sync server),它的作用是进行分布式事务处理、控制 EMDB 和 DBSVR 之间的数据交换并支持一个无线网络单元.移动计算机(MC)上保存数据库的一部分副本,并通过 EMDB 对本地数据进行管理.通过同步服务器上的 ODBC 接口,EMDB 可以与 DBSVR 进行数据交换.



①Database server(数据库服务器),②Synchronization server(同步服务器),③Replica(数据库副本),
④Embedded mobile database(嵌入式移动数据库系统),⑤无线广域网,⑥可信部分,⑦无线局域网,
⑧无线 Internet 网,⑨无线 Cellular 单元,⑩固定网络.

Fig.1 Architecture of the replicated mobile database system
图 1 复制的移动数据库系统结构

为了讨论方便,假设系统中所有的更新类事务均由 EMDB 产生并通过 Sync Server 进行全局提交,Sync Server 通过写全锁协议(ROWA)来保证各 DBSVR 之间的强一致性,此外,为了保证各 DBSVR 时钟的一致性,可以在固定主机层维护一个全局时钟[7,8],当各固定主机上的数据对象被修改时,新的时间印将取自该全局时钟.由于各固定主机保持强一致性且具有统一的时钟,因此可以它们视为一个逻辑实体(在后续讨论中,该逻辑实体仍以 DBSVR 表示).各个 EMDB 之间以及 EMDB 与 DBSVR 之间保持弱一致性,需要一定的机制对它们进行控制,因此仍然需要将它们表示为多个逻辑实体.DTSTLS 移动数据库同步处理模型如图 2 所示.



①Synchronization server(同步服务器),②Database Server(数据库服务器),③Replica(数据库副本),
④Embedded mobile database(嵌入式移动数据库),⑤无线网络.

Fig.2 DTSTLS synchronization model
图 2 DTSTLS 同步处理模型

在 DTSTLS 模型中,DBSVR 上的事务满足 ACID 性质,并通过两段锁协议保证事务调度的可串行性,但 EMDB 上的事务提交时仅处于本地提交状态,需要在 DBSVR 上验证通过后方能全局提交,将其所做的数据修

改写入 DBSVR.

上述移动数据库系统中的每个元组具有两个时间印:主时间印和副时间印.其中,主时间印是 DBSVR 修改元组时所加的时间标记,副时间印是 EMDB 修改元组时所加的事务标记(见第 2 节).EMDB 共有 3 种状态,即一致性状态、积累状态、消解状态,并依次在这 3 种状态之间进行转换.

(1) 一致性状态

EMDB 与 DBSVR 进行同步后,双方均已根据对方新近所做的修改刷新了本地数据,因此对应的元组在 EMDB 和 DBSVR 上具有相同的值,此时 EMDB 处于一致性状态.

(2) 积累状态

EMDB 在两次同步处理之间的状态为积累状态.在该状态下,EMDB 与 DBSVR 断接并在本地副本上执行移动事务.EMDB 所提交的移动事务的有关信息存放在移动事务日志中.移动事务日志记录各个非只读型移动事务的标识、事务操作、以及各操作的读映象集(为了方便说明,假设 EMDB 上各移动事务串行执行,且事务操作以 SQL 语句的形式表示).

定义 **1**(事务操作的读集). 对于事务 T,定义 $OP(T,i)$ 为 T 的第 i 个操作,同时定义 $OP(T,i)$的读集 $ReadSet(T,i)$为 $OP(T,i)$执行时读到的所有元组的标识(以"表名·主码值"的形式表示)的集合.

定义 **2**(事务操作的读映象集). 对于事务 T,设 Ψ为 $ReadSet(T,i)$中的任一元素,且 $PTS(\Psi)$,$STS(\Psi)$分别为 $OP(T,i)$执行时所看到的 Ψ 的主时间印和副时间印,则 T 的第 i 个操作 $OP(T,i)$的读映像集定义为:$RImage(T,i)=\{(\Psi,\zeta,\delta)|\ \Psi\in ReadSet(T,i)\wedge \zeta=PTS(\Psi)\wedge \delta=STS(\Psi)\}$.

定义 **3**(事务的读集). 设事务 T 含有 n 个操作,则 T 的读集定义为

$$RSet(T)=\bigcup_{i=1}^{n} ReadSet(T,\ i).$$

在移动计算环境中,移动计算机的资源非常有限.在 EMDB 的移动事务日志中存储元组的时间印而不是存储实际的数据值是为了最大限度地提高资源的利用效率.EMDB 移动事务日志的结构见第 2 节.

(3) 消解状态

当 EMDB 重新与 DBSVR 连接时,启动同步处理过程(为了方便说明,设同步处理时 EMDB 上没有其他移动事务),此时它处于消解状态.同步过程结束后,EMDB 与 DBSVR 达成一致,EMDB 又重新回到一致性的状态.

2 DTSTLS 模型的时间印处理策略

在 DTSTLS 模型中,DBSVR 上的元组始终具有主时间印,而副时间印为可选项.当 EMDB 处于一致性状态时,其元组仅含有主时间印且该主时间印与 DBSVR 上对应元组的主时间印相同(如图 3(a)所示).在积累状态和消解状态中,系统将按照规则 1~规则 4 对时间印进行处理.下面是讨论中将用到的符号:

PTS(X, $Site$):元组 X 在场地 $Site$ 上的副本中的主时间印,

STS(X, $Site$):元组 X 在场地 $Site$ 上的副本中的副时间印,

MCID(MC):移动计算机 MC 的标识,

MTID(MT):移动事务 MT 的事务标识,

CurrTime($Site$):场地 $Site$ 上的当前系统时间.

规则 1. 移动计算机 MC 启动移动事务 MT 时,按照如下方法生成事务标识:

$$\text{MTID}(MT)=\text{MCID}(MC)\cdot\text{CurrTime}(MC).$$

规则 2. EMDB 上的移动事务 MT 在本地副本上修改元组 X 时,将其时间印设置为:PTS(X, $EMDB$)保持不变,

$$\text{STS}(X,\ EMDB)=\text{MTID}(MT).$$

规则 3. 在同步处理的上载过程中,当 DBSVR 上的元组 X 被移动事务 MT 的基事务(见第 3 节)修改时,X 的时间印设置为:

$$\text{PTS}(X,\ DBSVR)=\text{CurrTime}(DBSVR),$$
$$\text{STS}(X,\ DBSVR)=\text{MTID}(MT).$$

规则 4. 在同步处理的下载过程中,EMDB 上被刷新的元组 X 的时间印设置为:

$$PTS(X, EMDB)= PTS(X, DBSVR),$$

$$STS(X, EMDB)= <null>.$$

在 DTSTLS 模型中,各数据对象的主时间印均取自 DBSVR(即固定主机层的全局时钟),EMDB 并不对数据的主时间印进行修改(见规则 2、规则 3 及定义 5 中时间印的比较方法),因此 EMDB 的时钟无须与 DBSVR 的时钟保持一致,另一方面,EMDB 在修改数据对象时仅更改其副时间印(副时间印上带有移动计算机的标识,见规则 1、规则 2),且副时间印的比较只有发生在同一个 EMDB 所修改的数据之间时才有意义(否则必然不相等),因此系统也无须保持不同 EMDB 之间时钟的一致性.

设一致性状态下 EMDB 上的数据库表 Tbl 的状态如图 3(a)所示,在积累状态下 EMDB 执行了 3 个移动事务.图 3(b)和图 4 分别给出了移动事务执行完毕时数据库的状态和 EMDB 移动事务日志的结构.

Tbl

Name	Acc	PTS	STS
Joe	3500	ts1	null
Susan	4500	ts2	null
Mike	800	ts3	null
Bob	2000	ts4	null

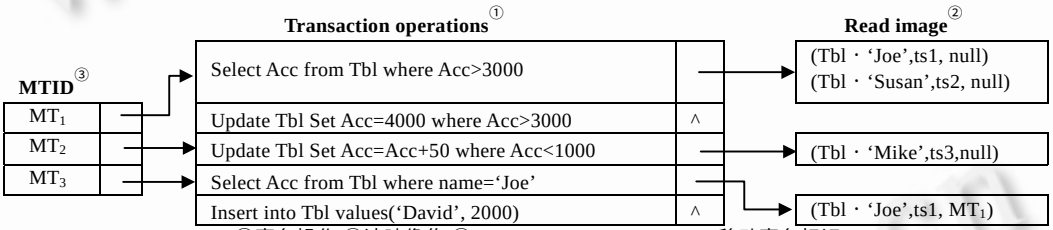
(a) Initial state
(a)初始状态

Tbl

Name	Acc	PTS	STS
Joe	4000	ts1	MT_1
Susan	4000	ts2	MT_1
Mike	1300	ts3	MT_2
Bob	2000	ts4	null
David	2000	null	MT_3

(b) Final state with mobile transactions completed
(b)移动事务执行完毕时的状态

Fig.3 Two different states of the sample database at EMDB
图 3 EMDB 数据库的状态



①事务操作,②读映像集,③Mobile transaction identifier(移动事务标识).

Fig.4 Structure of the log file at EMDB
图 4 EMDB 移动事务日志的结构

3 DTSTLS 同步处理算法

DTSTLS 同步处理算法由两个独立执行的子过程,即上载子过程和下载子过程组成.它们分别独立地在 Sync Server 上串行执行.在上载过程中,EMDB 将移动事务日志作为同步请求(synchronization request,简称 SR)发送给 Sync Server 并存入同步请求队列,随后可以断开连接.Sync Server 依次处理 SR 中的各个移动事务.设 SR 中含有 m 个移动事务,记为 $MT_1,MT_2,\ldots,MT_m$.当处理移动事务 MT_i $(1\leq i\leq m)$时,Sync Server 将首先在 DBSVR 上启动一个对应的基事务 BT_i,BT_i 含有与 MT_i 完全相同的操作序列.在执行 BT_i 的过程中,Sync Server 需要检测两类复制冲突,即读集冲突和执行时冲突.

定义 **4**. 对于事务 T,设 Ψ 为 $ReadSet(T,i)$ 中的任一元素,定义 $\xi(\Psi, T, j)$ 为事务 T 的第 j 步操作执行时所读到的 Ψ 的值.

定义 **5**(读集冲突). 对于任一移动事务 MT_i,设其含有 n 个操作,且其对应的基事务为 BT_i,如果下列条件之一成立则 DBSVR 检测到读集冲突:

(1) $(\exists j \in [1, n]) \wedge (ReadSet(MT_i, j) \neq ReadSet(BT_i, j))$

(2) $(\exists j \in [1, n]) \wedge (\exists\ \Psi \in (ReadSet(MT_i, j) \cap ReadSet(BT_i, j))) \wedge\ (\xi(\Psi, MT_i, j) \neq \xi(\Psi, BT_i, j))$.

读集冲突是指如下两种情况:① BT_i 的某个操作的读集与 SR 中记录的 MT_i 对应操作的读集不一致,② 读

集中对应元组的值不相等.其中第 2 种情况可通过时间印的比较来进行检测:设 X 为 MT_i 读集中的任意一个元组(其时间印记录在同步请求 SR 中),如果 X 的副时间印为空,则只需比较该元组与 DBSVR 上对应元组的主时间印,如果 X 的副时间印不为空,则只需比较对应元组的副时间印.可以证明,如果对应元组的时间印匹配,则它们的数据值必然相同,反之,如果对应元组的时间印不匹配,则可以认定当 EMDB 处于积累状态时,X 在 DBSVR 上的副本已被修改,或者 MT_i 读到了验证失败的移动事务所产生的数据(限于篇幅,证明从略).

定义 **6(**执行时冲突**).** 在执行 BT_i 各操作的过程中,如果某个语句未能正常执行且返回了错误代码,则检测到执行时冲突.

许多因素可以导致执行时冲突,如违反数据库的主码或唯一性约束,试图修改已删除的数据对象等.

定理 **1.** 对于移动事务 MT_i $(1\leq i\leq m)$,如果在执行其对应的基事务 BT_i 时,没有检测到读集冲突和执行时冲突,则 BT_i 在 DBSVR 上全局提交不会破坏数据库的一致性,否则 BT_i 不能全局提交.

证明:若题设条件成立,则:① 由于 BT_i 和 MT_i 的操作完全相同,且不存在读集冲突(即各操作所读数据也完全相同),因此 BT_i 保持了与 MT_i 相同的事务语义,即 BT_i 不破坏数据库的一致性,② 由于不存在执行时冲突,因此 BT_i 的各个操作在提交前均已正确执行,③ 由于 DBSVR 采用两段锁协议进行并发控制,因此其上的任一事务 T 必与 BT_i 满足可串行性,综合①~③,事务 BT_i 可以提交且不破坏 DBSVR 数据库的一致性.

如果题设条件不满足,则表明事务 BT_i 不能保持 MT_i 的原有语义,应该予以撤消. □

由于 EMDB 上的移动事务在修改数据对象时,以其事务标识作为数据对象的副时间印,因此可以利用这一特点提前排除必然会被回滚的事务.此外,对于需删除的数据,Sync Server 将为之打上特殊的删除标记,而不真正删除.这样被删除的数据对象可以被其他的 EMDB 下载以刷新其本地副本.上载算法见算法 1.

算法 **1**. DTSTLS 上载算法

```
INPUT: SR   /* 同步请求 */
1.  CancelSet=φ;   /* 初始化 */
2.  For i=1 to | SR | do   /* 逐个处理 SR 中的事务,| SR |为 SR 中的事务个数 */
3.    If (∃MTID ∈ CancelSet) ∧ (∃X ∈ RSet(MT_i)) ∧ (STS(X, EMDB)=MTID)
4.    Then   CancelSet = CancelSet ∪ {MT_i};   /* 直接撤消 MT_i */
5.    Else    Issue(ODBC, 'BeginTrans');   /* 启动 BT_i */
6.          j=1; pass=true;
7.          While (j<= | MT_i | ) and (pass==true) do   /* | MT_i | 为 MT_i 的操作个数 */
8.            If (MOP_j 为 Delete 操作) then   /* MOP_j 为 MT_i 的第 j 个操作 */
9.                将 MOP_j 改写为设置删除标记的语句;
10.           Endif;
11.           OP_j =MOP_j;   /* OP_j 为 BT_i 的第 j 个操作 */
12.           Issue(ODBC, OP_j ), 并对 OP_j 所修改元组的时间印按规则 3 进行处理,
13.           If (OP_j 的执行结果≠SUCCESS)
14.             Then pass=false;   /* 检测到执行时冲突 */
15.           Endif,
16.           If (ReadSet(OP_j) ≠ ReadSet(MOP_j )) ∨   /* OP_j 和 MOP_j 读集不相等 */
17.           ((∃ Ψ∈(ReadSet(OP_j ) ⋂ ReadSet(MOP_j))) ∧ (Timestmp( Ψ,OP_j)≠Timestmp( Ψ,MOP_j)))
18.                 /* OP_j 和 MOP_j 读集中对应元组的时间印不匹配,时间印的比较方法见定义 5,*/
19.             Then pass=false;   /* 检测到读集冲突 */
20.           Endif;
21.           j=j+1;
22.       EndWhile;
23.       If (pass==true) then Issue(ODBC, 'CommitTrans' );
24.                       Else Issue(ODBC, 'RollBack' );
25.       Endif;
```

```
26.  Endif;
27. Endfor;
28. 将 CancelSet 传送至 OB.
```

在下载过程中,Sync Server 首先根据 EMDB 上次同步的时间对 DBSVR 上的数据库进行扫描,读出所有自上次同步之后 DBSVR 修改过的数据并传送至输出缓冲区(output buffer,简称 OB),同时 Sync Server 还要读出 CancelSet 中的移动事务所修改或删除的元组,以撤消这些事务对 EMDB 数据库副本所做的修改.EMDB 可以异步地从 OB 取回下载的数据并刷新本地副本.下载算法见算法 2.

算法 **2**. DTSTLS 下载算法

```
INPUT: T_LastSync;  /* EMDB 与 DBSVR 上次同步时下载子过程结束的时间 */
       SyncTable;  /* 需要下载的表 */
       CancelSet;  /* 需撤消的移动事务的集合 */
       SR;  /* 同步请求 */
1.  Issue(ODBC, 'BeginTrans');
2.  For all Tablename ∈ SyncTable do  /* 读取自上次同步后修改过的数据 */
3.      Issue(ODBC, 'Select * from Tablename where PTS>=T_LastSync';  /* PTS 为元组的主时间印 */
4.      将返回值传送至 OB;
5.  EndFor;
6.  For all MT ∈ CancelSet do  /* 读取需回滚的移动事务修改过的数据 */
7.      For 所有 MT 中的 Update/Delete 语句 OP_UD
8.          将 OP_UD 改写为相应的 Select 语句 OP_S;
9.          Issue(ODBC, OP_S);
10.         将返回值传送至 OB;
11.     EndFor;
12. EndFor;
13. 对下载元组的时间印按规则 4 进行处理,
14. 将固定主机层全局时钟的当前值存入日志文件,作为下次同步的 T_LastSync,
15. Issue(ODBC, 'CommitTrans').
```

4 性能分析及结论

4.1 实验模型及参数说明

为了分析 DTSTLS 模型在移动计算环境下的性能,我们以近期研制的嵌入式移动数据库系统 KingBase Lite 2.0 为基础,通过模拟实验的方法进行了详细的实验.在实验系统中,DBSVR 和 EMDB 上分别保存数据库的一个完整副本.每一轮实验均从一致性状态开始,经过一段时间的断接操作,然后进入同步处理阶段.整个实验系统的主要参数见表 1.

4.2 实验结果分析及结论

我们选择 3 项主要指标,即移动事务撤消率、移动事务日志的数据量以及同步过程中的通信量,来对 DTSTLS 模型的性能进行分析.其中移动事务撤消率与断接时间 μ 之间的关系如图 5 所示.

由图 5 可以看出,从总体上讲,移动事务的撤消率随着 μ 的增长而增长.此外,它还随着 P_u 的增长或 P_r 的下降而增长.这是因为在 DTSTLS 模型中,发生冲突的可能性(包括读集冲突和执行时冲突)与 EMDB 断接期间 DBSVR 所更新的数据总量成正比.

为了比较 DTSTLS 时间印策略对存储空间及通信数据量的影响,我们采用了不加时间印的方法(在图中记为 ValueLog)作为实验的参照对象,实验结果如图 6 和图 7 所示.

Table 1 Main parameters of the simulation experiments

表 **1** 模拟实验中的主要参数

Parameter①	Value②	Meaning③
DBSize	2000 tuples	Number of tuples in the sample database④
TupleSize	256 bytes	Tuple size of the database⑤
TransSize	4	Average number of operations per transaction⑥
P_r	0.25 ~ 0.75	Proportion of read-only transactions at DBSVR⑦
P_u	0.25 ~ 0.75	Proportion of update operations in update transactions at DBSVR⑧
λ_f	20 / s	Transaction arrival rate at DBSVR⑨
λ_m	1 / s	Transaction arrival rate at EMDB⑩
μ	1 ~ 15 s	Disconnection time⑪

①参数,②值,③含义,④实验数据库中的元组数,⑤每个元组的字节数,⑥每个事务中的平均操作数,⑦DBSVR 上只读型事务所占的比例,⑧在 DBSVR 更新类事务中,更新操作所占的比例,⑨DBSVR 上的事务到达率,⑩EMDB 上的事务到达率,⑪断接时间.

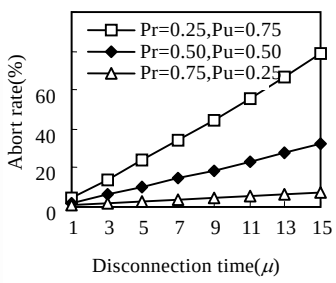


Fig.5 Mobile transaction abort rate

图 5 移动事务撤销率

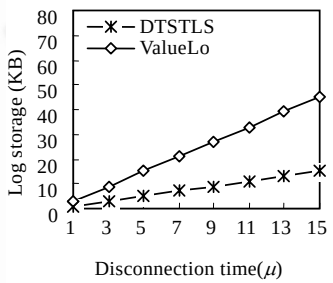


Fig.6 Storage consumption of log file

图 6 事务日志的数据量

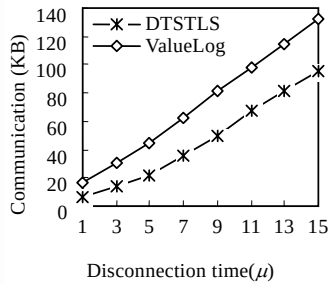


Fig.7 Communication cost

图 7 同步过程中的通信量

从上述实验结果可以看出,DTSTLS 模型有效地降低了移动计算机对存储空间和通信带宽的消耗,提高了资源的利用效率.可以推断,随着 TupleSize 的增加,DTSTLS 时间印策略对资源利用效率的优化效果也将逐渐变得明显.

综合上述分析,通过双时间印策略以及事务级冲突处理算法,DTSTLS 模型为移动数据库系统三级复制模式(即移动计算机 $\xleftrightarrow{\text{通信接口}}$ 同步服务器 $\xleftrightarrow{\text{ODBC}}$ 数据库服务器)提供了一种可行的解决方案.

致谢 感谢中国人民大学信息学院硕士研究生李凌伟在模拟实验中所做的工作及提出的宝贵建议.

References:

[1] Gray, J., Helland, P., O'Neil, P. E., *et al*. The dangers of replication and a solution. SIGMOG Record, 1996,25(2):173~182.

[2] Byun, S., Moon, S. Resilient data management for replicated mobile database systems. Data and Knowledge Engineering, 1999, 29(1):43~55.

[3] Wu, S., Chang, Y. An active replication scheme for mobile data management. In: Chen, A., Lochovsky, F., eds. Proceedings of the 6th International Conference on Database Systems for Advanced Applications. Los Alamitors, CA: IEEE Computer Society Press, 1999. 143~150.

[4] Terry, D., Theimer, M., Petersen, K., *et al*. Managing update conflicts in Bayou, a weakly connected replicated storage system. Operating Systems Review, 1995,29(5):172~183.

[5] Phatak, S. H., Badrinath, B. R. Multiversion reconciliation for mobile databases. In: Papazoglou, M., ed. Proceedings of the 15th International Conference on Data Engineering. Los Alamitors, CA: IEEE Computer Society Press, 1999. 582~589.

[6] Berenson, H., Bernstein, P., Gray, J., *et al*. A critique of ANSI SQL isolation levels. SIGMOG Record, 1995,24(2):1~10.

[7] Zhou, Long-xiang. Implementation Techniques of Distributed Database Management Systems. Beijing: Science Press, 1998 (in Chinese).

[8] Lamport, L. Time, clocks and the ordering of events in distributed systems. Communications of the ACM, 1978,21(7):558~565.

附中文参考文献:

[7] 周龙骧.分布式数据库管理系统实现技术.北京:科学出版社,1998.

A Novel Transactional Synchronization Scheme for Replicated Mobile Database Systems*

DING Zhi-ming[1], MENG Xiao-feng[2], WANG Shan[1,2]

[1](*Institute of Computing Technology, The Chinese Academy of Sciences, Beijing* 100080, *China*);

[2](*Institute of Data and Knowledge Engineering, Renmin University of China, Beijing* 100872, *China*)

E-mail: dingzhiming@263.net; {xfmeng,suang}@public.bta.net.cn

http://www.ict.ac.cn

Abstract: Synchronization is one of the key technologies in maintaining the consistency of replicated mobile database systems. A lot of research has been focused on transactional synchronization schemes in recent years and many models and algorithms have been proposed. However, existing transactional synchronization schemes have some limitations. In order to solve these problems, a new synchronization scheme, double-timestamp transaction-level synchronization (DTSTLS) model, is put forward in this paper. DTSTLS model utilizes a three-tire architecture. Therefore existing database products can be used as its database server, which improves the flexibility and extensibility of the system. As an asynchronous group replication method, DTSTLS model allows mobile users to access local replicas of the database and to locally submit mobile transactions when the system is disconnected. The locally committed transactions are sent to the synchronization server for conflict reconciliation and synchronization when the system is reconnected. Mobile transactions that have executed conflicting operations will be aborted, and only those mobile transactions that do not violate the consistency of the database system can be globally committed. In this way, the consistency of the database system is ensured. Besides, through a novel timestamp strategy, DTSTLS model cuts down the resource consumption and reduces the communication costs. Detailed simulation experiments are carried out, and the results show that the proposed methodology not only ensures the serializability and consistency of the mobile database system, but also reduces storage consumption and communication cost.

Key words: mobile computing; database; transaction processing; replication; synchronization

---

* Received April 20, 2001; accepted September 18, 2001

Supported by the National Natural Science Foundation of China under Grant No.60073014; the National High Technology Development 863 Program of China under Grant No.863-306-ZD12-12-1