第29卷 第8期 计 算 机 学 报 Vol. 29 No. 8
2006年8月 CHINESE JOURNAL OF COMPUTERS Aug. 2006

核心化多级安全数据库系统未决提交事务日志写出依赖研究

徐 震 张 敏
(中国科学院软件研究所信息安全国家重点实验室 北京 100080)

摘 要 核心化体系结构的多级安全数据库系统中不同级别事务由该级别DBMS实例处理,DBMS实例自行维护事务日志缓存.事务处理过程中高级事务可能读取已提交的低级事务数据,如果低级事务提交日志记录尚未写入持久存储,而高级事务已提交并且提交日志记录写入持久存储后系统崩溃,恢复后系统将进入不一致的状态.为解决上述问题引入一个可信的日志协调实体维护全局的未决提交事务间的依赖关系,并协调各个DBMS实例的提交日志记录写出操作,保证被依赖的提交日志记录先于依赖它们的日志记录写入持久存储.文中还给出了方案的实现算法,并证明了算法的正确性,通过分析论证了方案的实用性.

关键词 核心化体系结构;多级安全数据库系统;未决提交;依赖图
中图法分类号 TP309

Research on Dependency of Flushing of Pending Commit Transaction Log in Kernelized Multilevel Secure Database System

XU Zhen ZHANG Min
(*State Key Laboratory of Information Security, Institute of Software, Chinese Academy of Sciences, Beijing* 100080)

Abstract In a kernelized DBMS, transactions are processed by their corresponding DBMS instances and those instances maintain log storage and log caches of their own. During the execution of transactions, a transaction having high security level may read data item which is generated by low security level transaction. Suppose when the commit log record of low level transaction is still in cache and the high level transaction has committed and the commit log has been flushed into durable storage, the system will reach an inconsistent state after restarting. In order to solve the problem, a trusted log coordinator is introduced to maintain the global dependency of pending commit transactions' commit log record coordinate the log writing of different DBMS instances to guarantee that the depended commit log records are written out before the depending ones. Algorithms of the scheme are given and their correctness is proven. Finally, the effective of the scheme is demonstrated via analysis.

Keywords kernelized architecture; multi-level secure database system; pending commit; dependency graph

---

收稿日期:2006-03-06;修改稿收到日期:2006-05-31.本课题得到国家自然科学基金(60025205,60273027)、国家"八六三"高技术研究发展计划项目基金(2004AA147070,2002AA141080)资助.徐 震,男,1976年生,博士,助理研究员,主要研究方向为访问控制、系统安全与可信计算.E-mail: xuzhen@is.iscas.ac.cn.张 敏,女,1975年生,博士研究生,助理研究员,主要研究方向为数据库安全与安全评价.

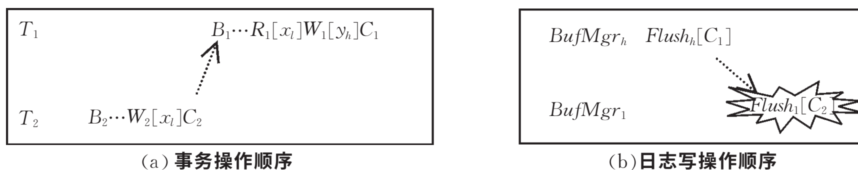
1 引　言

对于多级安全数据库系统的设计而言，确定其体系结构至关重要，因为它在很大程度上决定了系统的性质和实现代价. 国内外在这个领域已经进行了一定的研究，并且提出了若干方案. 其中有四类体系结构受到了广泛的重视：核心化(kernelized)体系结构、完整性锁(integrity lock)体系结构、数据复制(replicated)体系结构以及可信主体(trusted subject)体系结构[1,2].

采用核心化体系结构的多级安全数据库系统中不同级别的事务分别由相应级别的 DBMS 实例提供服务. 采用这种体系结构的安全数据库系统可以充分利用安全操作系统提供的安全功能. 另外，还可以直接利用现有的数据库系统，并且采用的数据库系统不必是强制 TCB 的一部分. 因此，这种方式可以保证 TCB 规模尽量小，从而可以用尽量低的开发和评估代价实现高级别的安全保障[3,4].

核心化体系结构的系统中存在多个 DBMS 实例，并且这些实例都有一定的自治性. 因此，从全局而言需要一个实体来协调不同实例中的多个日志子系统. 日志子系统主要进行事务操作的备份与恢复. 在核心化体系结构的配置下，各个 DBMS 实例自行维护日志的内存缓存.

然而，在 LOIS 安全数据库系统[5]的研究过程中，我们发现核心化的安全数据库系统结构为事务提交日志记录写出带来了新问题，如图 1 所示. 低级事务 T_2 修改低级数据 x_l 后提交，而高级事务 T_1 操作过程中读取了 x_l 并提交(图 1(a)). 然而，两个事务的日志分别由各自的日志管理器维护，如果事务 T_1 的提交日志记录成功写入持久存储，而 T_2 的 DBMS 实例崩溃，则恢复后系统实际上进入了不一致状态.



(a) 事务操作顺序　(b)日志写操作顺序

图 1　未决事务提交日志依赖问题

该问题主要由 4 个因素引发：(1)不同安全级别事务间存在关联性；(2) DBMS 实例的日志子系统具有自治性；(3)事务的提交实际上存在两种状态——提交日志写入缓存和写入持久存储；(4)系统中存在多种类型崩溃的可能. 为此，需要引入机制对 DBMS 实例的日志子系统进行协调，保证事务间的关联性与事务提交相一致，从而避免上述问题.

在本文中我们将提交日志仅写入日志缓冲区的事务称为未决提交事务，而将提交日志写入持久存储的事务称为提交事务. 在存在节点或进程故障的情况下，系统的一致性无法得到保障. 本文将上面的问题称为未决提交事务日志写出依赖问题. 为解决该问题，我们在系统中引入了一个日志协调实体来协调各个实例的日志缓存管理模块的提交日志记录写出和恢复. 未决提交事务日志的写出依赖关系通过依赖图来刻画，依赖图由日志协调实体创建和维护. 日志协调实体根据依赖图控制各日志管理模块的提交日志记录写出，保证被依赖的未决事务提交记录先于依赖事务写出，本文给出了相关的主要算法描述，并证明了算法可以确保事务系统不会发生上述问题.

本文第 2 节主要讨论了相关工作；第 3 节将给出安全模型、日志子系统抽象结构和系统故障模型；第 4 节给出问题解决方案的算法、正确性证明以及算法的实例. 第 5 节对方案的实现进行了分析；最后在第 6 节对全文的工作进行总结，并给出下一步的研究思路.

2 相关工作

核心化数据库系统体系结构提出后[6]引起了研究者大量关注. 文献[7]提出了该领域研究的挑战并给出了已有的解决方案. 文献[5]对包括核心化数据库系统在内的安全数据库研究作了全面的综述，并给出了作者在 LOIS 安全数据库系统方面的成果. 然而，核心化数据库系统恢复机制研究远不如并发控制那样多，很大原因在于一些研究者认为恢复机制可以在每个安全级别独立进行[8]. 文献[9]指出，新事务在开始执行前必须保证它所支配的安全级别全部完成恢复，并给出了一个基于 UNDO 日志的方案，文献[10]的原形中采用了上述方案. 然而，上述工作主要在数据复制体系结构的上下文中进行，并

且在这种环境下关注的主要是安全提交问题.

实际上,核心化数据库系统中各个 DBMS 实例相对自治,因此导致其中事务的提交和恢复具有了一定的分布式提交和恢复的特点. 分布式提交和恢复方面已经有了广泛而深入的研究,包括两阶段提交[11]及其优化方案、一阶段提交[12]和混合提交方案[13]等,其评价指标主要有消息数、磁盘写、延迟和非阻塞. 然而,这些方案在核心化数据库系统中并不适用,因为在这些提交协议中每个事务提交将导致至少一次日志缓存写出. 核心化数据库系统仍然是一种集中式数据库,每个事务一次以上的日志缓存写出操作将严重影响系统的性能[14],通过可信实体协调日志缓存的写出完全可以同时保证完整性和系统性能.

3 系统结构与基础定义

3.1 安全模型

本文的数据库系统的安全模型采用经典的 BLP 模型[15],包括一个数据(客体)集合 D、一个操纵数据集合中数据的事务(主体)集合 T 以及一个由安全级别组成的格结构 S. 其中,S 中的元素间存在偏序的支配关系——$\geq$. 如果 $s_i \geq s_j$,则称 s_i 支配 s_j. 如果 $s_i \geq s_j$ 且 $i \neq j$,则称 s_i 严格支配 s_j,表示为 $s_i > s_j$. D 中每个数据元素以及 T 中元素均被授予固定的安全级别. 级别函数 $L: D \cup T \rightarrow S$ 给出数据元素或事务的安全级别.

事务 T_i 为了访问数据 x,需要满足以下两个条件:

(1) T_i 可以读访问 x,仅当 $L(T_i) \geq L(x)$.

(2) T_i 可以写访问 x,仅当 $L(x) = L(T_i)$.

考虑到完整性问题,这里对 *-property 进行了更严格的限制——仅允许写同级数据.

3.2 系统事务模型

与上述安全模型相对应,本文中论及的数据库系统采用了多级事务模型. 系统支持并发的事务处理,每个事务拥有一个安全级别,它可以访问的数据受到了安全模型的限制. 事务是一个操作序列(不考虑子事务、嵌套事务等复杂事务模型),操作包括 $r[x]$, $w[x]$, *commit* 和 *abort*,此外还要满足以下条件:

(1) 事务操作序列的最后一个操作为 *commit* 或 *abort*;

(2) *commit* 或 *abort* 不能同时出现,并且事务中仅存一个 *commit* 或 *abort*;

(3) 对事务 T_i 操作中任意 $r[x]$, $L(T_i) \geq L(x)$;

(4) 对事务 T_i 操作中任意 $w[x]$, $L(x) = L(T_i)$.

而单级事务是一个蜕化的多级事务,也即只读写访问同级数据.

3.3 日志子系统的结构

本文中的多级安全数据库系统采用了核心化的结构,这里的抽象系统结构仅涉及全局日志协调实体和 DBMS 实例中的日志子系统(如图 2 所示).

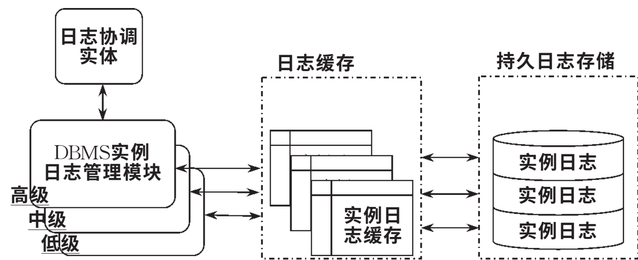


图 2 日志子系统抽象结构

整个日志子系统主要包括运行实体(日志协调实体、DBMS 实例日志管理模块)和存储实体(日志缓存、持久存储):

(1) 日志协调实体. 它用于维护未决事务提交日志写出的依赖关系,保证被依赖的未决事务提交日志先行写入持久存储. 由于需要与各级别 DBMS 通信,日志协调实体是一个可信实体.

(2) DBMS 实例日志管理模块. 系统中每个 DBMS 实例的日志管理模块自行维护本级别日志缓存,并根据需要将日志缓存写入持久存储(日志缓存和持久存储的安全级别与 DBMS 实例的安全级别相同),提交日志的写出操作需要日志协调实体来协调.

(3) 日志缓存. 不同级别日志记录写入不同级别日志缓存,DBMS 实例的日志管理模块根据需要将日志记录写入持久存储. 日志记录写出有四种情况:日志缓存满、显式日志写出命令、同级依赖事务日志记录写出和高级依赖事务提交日志记录写出. 日志缓存是非持久存储,发生实例崩溃或系统崩溃时其中的数据会丢失.

(4) 持久日志存储. 不同级别日志记录写入不同级别日志缓存,写操作由 DBMS 实例的日志管理模块执行. 持久日志存储中的数据在实例崩溃或系统崩溃时不会丢失.

3.4 系统故障模型

日志协调模块与各个 DBMS 实例运行在不同进程空间中,因而存在故障的独立性. 由于事务故障在实例中可以自行处理,故障模型主要包括实例崩

溃、协调进程崩溃和系统崩溃.

(1) DBMS 实例崩溃. DBMS 在实例崩溃情况下，实例的日志管理器所维护的日志缓存将丢失，其中保存的提交事务日志未能写入持久存储，为此该事务将失败，同时需要事务协调模块协调各个实例保证依赖于该事务的最终状态为失败.

(2) 协调进程崩溃. 协调进程（日志协调实体）发生故障后未决事务提交依赖关系将无法确定，此时各实例中的日志缓存应被丢弃，结果导致全系统故障，恢复机制与全系统一致.

(3) 全系统崩溃. 所有 DBMS 实例的日志缓存和协调模块的状态丢失，系统重启，在协调进程的控制下进行系统恢复.

本文中假定系统中存在机制保证 DBMS 实例崩溃导致全系统崩溃，因此文中只需讨论全系统故障的情形.

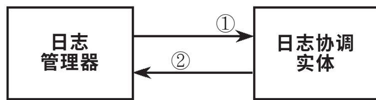
4　提交日志记录写出方案

本小节描述提交日志记录写出方案，主要描述日志协调实体和 DBMS 实例的日志管理器对未决提交事务日志写出依赖问题的处理.

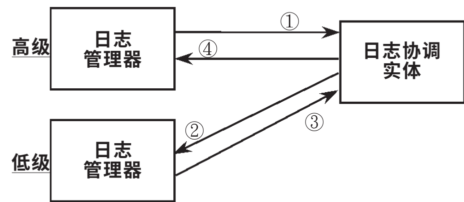
4.1　处理流程

图 3 是提交日志写出方案的图示. 其中，图 3(a)描述依赖关系的维护流程. 日志管理器接受来自事务管理器提交请求，生成提交日志记录，事务管理器的请求中还包括指定事务读访问数据的生成事务集合. 在消息①中日志管理器告知日志协调实体新增的未决提交事务和所有可能的依赖关系. 日志协调实体处理完毕后将通过②来确认.

图 3(b)描述日志的写出处理流程. 日志管理器接收到日志写出请求或日志缓存满需要写出日志记录，为避免未决提交事务日志写出依赖问题，需要日志协调实体的帮助. 在消息①中日志管理器发出提交协调请求. 日志协调实体根据维护的依赖关系计算所有其它实例的日志管理需要先行写出的提交日志，然后根据被支配的关系拓扑有序地自下而上（保证所有被依赖的日志记录首先处理）将所有被依赖的日志记录写出. 在图 3(b)所示的例子中，仅有一个低级实例中的提交日志记录被依赖，通过消息②请求写出相应日志记录，消息③是确认. 最后，日志协调实体通过消息④确认所有被依赖的日志记录全部写出.



(a) 依赖关系维护流程



(b)日志写出处理流程

图 3　处理流程图示

4.2　日志管理器

这里只讨论日志管理器在处理未决提交事务日志写出依赖问题方面的机制. 从交互的角度看，它接受其它实体的请求，同时也会请求其它实体进行处理. 接受的请求包括：(1)事务管理器将提交日志记录和该事务读访问低级数据的生成事务集合同时通过接口发至日志管理器；(2)实例可能请求日志管理器写出缓存数据；(3)日志协调实体执行协调过程中会要求日志管理器将特定日志记录写出. 对外的请求包括：(1)告知日志协调实体有新的未决事务加入，同时发送该事务读访问低级数据的生成事务集合；(2)请求日志协调实体协调写出提交日志记录. 下面给出日志管理器的算法.

算法 1.　$LogManager_s$( ).

/ * s 是日志管理器的安全级别 * /

Cobegin

Thread *CoordingnatedLogOp*( )

Repeat

　ReceiveDFlush(*Coordinator*, *lsn*)

　　/ * 接收协调者的日志写请求 * /

　/ * 日志协调实体考虑彻底，T_i 提交日志记录前的提交日志记录不需额外处理 * /

　Flush(*lsn*) / * 将 T_i 提交日志记录及之前的记录写入持久存储 * /

　Ack(*Coordinator*, OK) / * 告知日志协调者已成功写出 * /

Forever

End thread

…

Thread *LogOpReceiver*( ) / * 日志操作接收与处理 * /

Repeat

　ReceiveLogOp(*DB_Instance*$_s$, *op*, *para*)

```
    Case op Do
    CommitLog: /* 成功完成操作前该事务所作修改不
                 可见 */
      lsn := NewLsn( ) /* 获取提交日志序列号 */
      logRec :=
          GenerateLogRecord(lsn, para.tid, commit, null)
      If 缓存空间不够
        LFlush(lsn-1) /* 算法 2 专门讨论函数的实现 */
      AddLogBuffer(logRec) /* 日志记录加入缓存 */
      /* 告知日志协调实体新未决事务加入,para.rset 是
         事务读访问的全部低级数据的生成事务集合 */
      Send(Coordinator, NewPendingNode, para.tid,
           lsn, para.rset)
    FlushBuffer:
      LFlush(para.lsn) /* 在日志协调实体的控制下将
                 指定位置前的日志写入持久存储 */
    Other:
      ...
    End Case
  Forever
  End Thread
  ...
  Coend
  End LogManager_s
```

算法 2. *LFlush(lsn).*

```
clsn := lsn
found := false
/* 找到最新的提交日志记录 */
While clsn 不是日志缓存最后一个记录
  If clsn 对应日志记录是提交日志记录
    found := true
    break
  clsn := clsn-1
End while
If found = True
  /* 请求日志协调实体控制将被依赖未决事务提交
     日志写出,并等待完成 */
  Send(Coordinator, FlushingCommitLog,
       GetTid(clsn), clsn, null)
Flush(lsn)
End LFlush
```

算法 2 取代了原有系统中的日志记录写出,它的主要功能是查找日志缓存中最新(lsn 最大)的提交日志记录并请求日志协调实体写出所有被依赖的未决事务的提交日志记录. 日志协调实体可以根据这个 lsn 计算本实例将写出的所有提交日志记录和它们依赖的低级未决事务. 两个算法中用到的函数接口功能在表 1 中给出说明,文中不再详细描述.

表 1 算法 1、2 中用到的函数说明

函数	说明
ReceiveDFlush(FROM, lsn)	接收来自 *FROM* 的日志写出请求,参数为 *lsn*
Flush(lsn)	将序列号为 *lsn* 的日志记录写入持久存储
Ack(FROM, STATUS)	告知 *FROM* 处理结果 *STATUS*
ReceiveLogOp(FROM, op, para)	接收 *FROM* 的日志处理操作,*op* 为操作类型,*para* 为参数
NewLsn()	获取下一个日志序列号
GenerateLogRecord(lsn, tid, op, data)	生成日志记录,*lsn* 为日志序列号,*tid* 为事务标识,*op* 为日志操作,*data* 为日志内容
AddLogBuffer(logRec)	将 *logRec* 写出日志缓存
Send(TO, op, tid, lsn, para)	发送请求至 *TO*,操作 *op* 包括新未决事务和日志协调请求,*tid* 为事务标识,*lsn* 为日志序列号,*para* 在新未决事务请求中表示可能的依赖事务集

4.3 日志协调实体

依赖图(dependent graph)用来精确刻画未决提交事务间提交日志记录写出的依赖关系,它的端点是系统特定时刻所有未决提交事务的集合,而有向边是表明高级事务提交日志写出对低级事务提交日志记录的依赖关系.

定义 1. DG 是依赖图集合,任意 $dg \in DG$ 是有向图,可以表示为一个二元组 $\langle V, E\rangle$,

$dg.V = \{T_i \mid T_i$ 是特定时刻系统中全部未决提交事务的集合$\}$;

$dg.E = \{(T_i, T_j) \mid T_i, T_j \in V(dg),\ L(T_i) > L(T_j)$ 且 T_i 提交日志记录写出依赖于 $T_j\}$.

定理 1. 依赖图是无环图.

证明. 假定 $dg' \in DG$ 是有环图,其中任意一个环为$(T_i, \cdots, T_j, T_k, T_j)$,其中 T_i 是安全级最高的事务,则存在$(T_k, T_i) \in DG$,所以 $L(T_k) > L(T_i)$,与假定矛盾. 证毕.

DBMS 实例的日志管理器自行维护实例的日志记录,每个日志记录有一个唯一的序列号(LSN). 在执行写出操作时实际上也存在依赖关系——序列号大的日志记录依赖于小的,换句话说就是序列号小的日志记录要先行写出. 为此需要维护事务与其提交日志记录映射关系的数据结构. 这里引入了实例未决事务集合和系统未决事务集合.

定义 2. 实例未决事务集是事务标识及其提交日志序列号构成的二元组(tid, lsn)的集合，集合中所有事务的安全级别相同，表示为 PTS. 系统未决事务集是一个系统中所有安全级别的实例未决事务集，表示为 $SPTS$. 对于 $spts \in SPTS$，用 $spts_s$ 表示 $spts$ 中安全级别为 s 的实例未决事务集.

另外，我们扩展级别函数 $L: D \cup T \cup PTS \rightarrow S$ 能够给出数据元素、事务和同级未决事务集的安全级别. 日志协调实体维护一个依赖图和一个系统未决事务集，基于这两个数据结构判断依赖关系并协调日志写操作.

算法 3. $Coordinator(\ )$.

```
/* 协调日志写出操作 */
创建和初始化依赖图 DepG
创建和初始化系统未决事务集 Spts
Cobegin
Thread DepGraph( )
Repeat
  Receive(LogManager_s, op, tid, lsn, rset)
              /* 接受来自各实例日志管理器的请求 */
  Case op Do
  NewPendingNode:
    If tid not in DepG.V /* 在依赖图中增加新节点 */
      DepG.V := DepG.V ∪ {tid}
      For each dtid in rset /* tid 读访问的全部低级数
                        据的生成事务组成的集合 */
      /* 在依赖图中增加依赖关系 */
        If dtid in DepG.V /* 未决提交依赖 */
          DepG.E := DepG.E ∪ {<tid, dtid>}
      End For
    /* 在系统未决事务集中增加事务 */
    Spts_s := Spts_s ∪ {(tid, lsn)}
  FlushingCommitLog:
    GFlush(tid) /* 依据依赖图协调各级实例的日志
                写出操作，具体的算法 4 中给出 */
  End Case
Forever
...
Coend
End Coordinator
```

算法 4. $GFlush(tid)$.

```
创建 ISpts 并初始化为空 /* ISpts 类型为 SPTS，用来
                        记录中间处理结果 */
/* 对 Spts 中所有安全级别被 L(tid) 支配的实例未决
   事务集根据支配(>)关系进行拓扑排序，生成安全
   级别数组 level[n] */
n := TopoSort(Spts, L(tid), level[], '>')
/* 计算所有同级依赖的事务 */
For each ltid in Spts_L(tid)
  If LSN(ltid) < LSN(tid)
    ISpts_L(tid) := ISpts_L(tid) ∪ (ltid, LSN(ltid))
End For
/* 计算所有跨级别直接依赖 */
For i := 1 to n Do
  For each (ltid, lsn) in ISpts_level[i]
    For each (ltid, dtid) in DepSet.E
      ISpts_L(dtid) := ISpts_L(dtid) ∪ (dtid, LSN(dtid))
                              /* 增加依赖未决事务 */
      For each (ktid, klsn) in ISpts_L(dtid)
                              /* 增加实例未决事务 */
        If klsn < LSN(dtid)
          ISpts_L(dtid) := ISpts_L(dtid) ∪ (ktid, LSN(ktid))
      End For
    End For
  End For
End For
/* 对 ISpts 中所有不为空的 pts 的安全级别根据被支
   配关系(<)进行拓扑排序生成数组 level[n] */
n := TopoSort(ISpts, MinLevel(ISpts), level[], '<')
For i := 1 to n Do
  /* 发送写出消息并等待完成，发送最大的 lsn 即可 */
  SendDFlush(LogManager_level[i],
              MaxLSN(ISpts_level[i]))
  For each tid in ISpts_level[i] /* 清理数据结构 */
    RemoveNode(DepG, tid)
                      /* 删除图中节点和关联的边 */
    Spts_level[i] := Spts_level[i] - {(tid, LSN(tid))}
                      /* 删除系统未决事务集中的事务 */
  End For
End For
End GFlush
```

由于采用算法 1、算法 2，我们扩展级别函数 $L: D \cup T \cup PTS \rightarrow S$ 能够给出数据元素、事务和同级未决事务集的安全级别. 日志协调实体维护一个依赖图和一个系统未决事务集，基于这两个数据结构判断依赖关系并协调日志写操作.

算法 3、算法 4 描述的方案写出日志的过程中，事务依赖的事务集被首先写入持久存储，所以恢复就可以采用 DBMS 原有实例的恢复机制进行，这里不再赘述.

表 2 算法 3、4 中用到的函数接口

函数	说明
$Receive(FROM, op, tid, lsn, para)$	接收来自 $FROM$ 的操作请求，op 包括新未决事务加入和提交日志记录写出协调操作，tid 为事务标识，lsn 为日志序列号，$para$ 在 op 为新未决事务加入时是读访问依赖事务集
$TopoSort(Spts, l, level[\,], rel)$	对 $Spts$ 中所有安全级别被 l 支配的实例未决事务集，根据关系 rel（'>'或'<'）进行拓扑排序，l 根据关系 rel 分别为安全级上界和下界，生成安全级别数组 $level[\,]$，返回值为数组的大小
$SendDFlush(TO, lsn)$	向 TO 发送写出 lsn 前的日志记录的请求，并等待完成
$MaxLSN(pts)$	获得实例未决事务集中最大的 lsn
$LSN(tid)$	获得 tid 对应的提交日志记录序列号
$RemoveNode(DepG, tid)$	输出 $DepG$ 中 tid 对应的节点和所有 tid 发出和至 tid 的边

4.3 子系统性质

前面的算法描述了提交日志记录的写出方案，需要日志协调实体和各 DBMS 实例的日志管理器协作完成. 下面，证明该方案可以解决未决提交事务日志写出依赖问题.

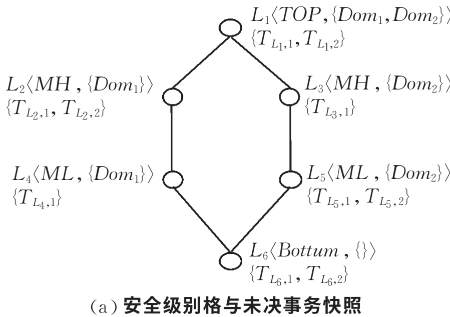
定理 2. 采用本文提交日志写出方案的日志子系统中不存在未决提交事务日志写出依赖问题.

证明. 首先，我们扩展级别函数 $L: D\cup T\cup PTS\rightarrow S$ 能够给出数据元素、事务和同级未决事务集的安全级别. 日志协调实体维护一个依赖图和一个系统未决事务集，基于这两个数据结构判断依赖关系并协调日志写操作.

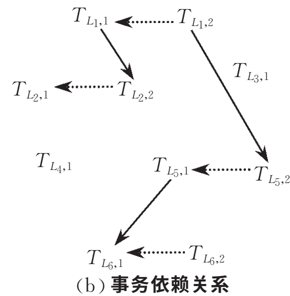
算法 3 中 $DepG$ 精确维护了高级未决提交事务的提交日志对低级未决提交事务的提交日志的依赖关系. 其次，算法 2 中 $ISpt$ 的计算结果是事务 tid 的所有依赖的事务集合. 对 $L(tid)$ 支配的安全级别拓扑排序，根据这个顺序计算未决依赖关系不会遗漏，而同级依赖关系不会遗漏. 此外，写出操作按照 $ISpt$ 中安全级别被支配顺序进行，保证了高级未决事务依赖的低级未决事日志记录首先写出. 同时，在同级写出过程中日志序列号小的记录先写出，保证了同级依赖关系. 最后，在发生系统崩溃时，各级别独立恢复，因为各级别持久日志存储中不存在未决依赖情况，所以系统恢复后能够保证一致性. 证毕.

4.4 实 例

第 3 节给出了方案的主要算法，这里通过一个实例说明算法 $4GFlush(tid)$ 的计算过程. 图 4(a)是系统安全级别的配置和未决事务快照. 其中，系统的安全级别格中有 4 个等级（$TOP, MH, ML, Bottum$），两个范畴（Dom_1, Dom_2），安全级别共有 6 个（$L_1\sim L_6$），每个级别下分别标注了处于未决提交的事务. 图 4 的依赖关系计算实例(b)是(a)中事务的依赖关系，虚线表示同级实例依赖，实线表示未决提交依赖. $LogManager_{L_1}$ 决定写出事务 $T_{L_1,2}$ 的提交日志记录，计算如下：



(a) 安全级别格与未决事务快照



(b) 事务依赖关系

图 4 依赖关系计算实例

1. 初始化计算出 $ISpts_{L_1}=\{T_{L_1,1}, T_{L_1,2}\}$.

2. 对安全级别根据支配关系拓扑排序得出 $level[\,]=\{L_1, L_2, L_4, L_3, L_5, L_6\}$.

3. 根据 $level[\,]$ 循环计算 $ISpts$，下面列出 6 步的计算结果：

① $\{L_1\{T_{L_1,1}, T_{L_1,2}\}, L_2\{T_{L_2,1}, T_{L_2,2}\}, L_3\{\}, L_4\{\}, L_5\{T_{L_5,1}, T_{L_5,2}\}, L_6\{\}\}$；

② $\{L_1\{T_{L_1,1}, T_{L_1,2}\}, L_2\{T_{L_2,1}, T_{L_2,2}\}, L_3\{\}, L_4\{\}, L_5\{T_{L_5,1}, T_{L_5,2}\}, L_6\{\}\}$；

③ $\{L_1\{T_{L_1,1}, T_{L_1,2}\}, L_2\{T_{L_2,1}, T_{L_2,2}\}, L_3\{\}, L_4\{\}, L_5\{T_{L_5,1}, T_{L_5,2}\}, L_6\{\}\}$；

④ $\{L_1\{T_{L_1,1}, T_{L_1,2}\}, L_2\{T_{L_2,1}, T_{L_2,2}\}, L_3\{\}, L_4\{\}, L_5\{T_{L_5,1}, T_{L_5,2}\}, L_6\{\}\}$；

⑤ $\{L_1\{T_{L_1,1}, T_{L_1,2}\}, L_2\{T_{L_2,1}, T_{L_2,2}\}, L_3\{\}, L_4\{\}, L_5\{T_{L_5,1}, T_{L_5,2}\}, L_6\{T_{L_6,1}\}\}$；

⑥ $\{L_1\{T_{L_1,1}, T_{L_1,2}\}, L_2\{T_{L_2,1}, T_{L_2,2}\}, L_3\{\}, L_4\{\}, L_5\{T_{L_5,1}, T_{L_5,2}\}, L_6\{T_{L_6,1}\}\}$.

4. 对安全级别根据被支配关系拓扑排序得出 $level[\,]=$

$\{L_6, L_4, L_2, L_5, L_3, L_1\}$.

5. 根据 $level[\ ]$ 写出提交日志记录，顺序为 $(T_{L_6,1}, T_{L_2,2}, T_{L_5,2}, T_{L_1,2})$.

6. 写出结束后 $Spts = \{L_1\{\}, L2\{\}, L_3\{T_{L_3,1}\}, L_4\{T_{L_4,1}\}, L_6\{T_{L_6,2}\}\}$.

5 分　析

上文中已经对未决提交事务日志写出依赖问题的解决方案进行了详细描述，这一小节主要分析引入该解决方案对系统的影响.

首先，在系统结构方面需要引入一个可信的日志协调实体，并对日志管理器和事务管理器进行少量修改. 在核心化体系结构的 DBMS 中，一般需要一个可信实体进行全局的协调和同步，日志协调实体可以作为它的子模块实现. 日志协调实体实现比较简单不至于过多增加可信实体的代码量，而造成安全评估方面的过高代价.

其次，引入日志协调实体后会带来一定的性能损失. 因为在存在提交依赖的情况下，日志写出操作将请求日志协调实体将所有被依赖的日志记录现行写出，这将影响到若干相关 DBMS 实例的处理，从而一定程度上降低整体性能. 但是需要说明的是该机制将会影响到进行低级数据读，同时有些操作的事务. 而在实际处理中只读事务和同级读写事务占了很大一部分. 所以，综合考虑该机制的引入对系统的性能影响是可以接受的.

此外，对系统最微妙的影响在于，该机制可能被用于由高安级到低级的隐通道. 根据非干扰原理[16]，为了保证信息流安全，高级的操作不应影响低级状态. 而在本文的写出方案中，高级数据库系统实例的提交操作可能导致低级别实例的日志写操作. 这样恶意高级用户就可以和低级用户串谋构造隐通道，实现非法信息流. 下面，我们通过一个案例定性地描述一个隐通道构造方案.

(1) 低级用户通过实验分别估计数据库实例无磁盘写操作和有磁盘写操作两种情况下的响应时间. 高级用户通过试验分析或其它方式得到高级 DBMS 实例日志缓存大小和修改操作日志记录大小.

(2) 高级用户事务做一定量修改操作导致日志缓存将满，少量操作即将导致日志写出.

(3) 低级用户修改少量数据并提交事务(写磁盘概率很低，日志系统采用缓存机制).

(4) 高级事务读低级用户事务的数据，执行写操作并提交事务，再通过少量操作导致上述提交日志写入持久存储，该操作将导致强制低级用户的事务提交日志记录写出.

(5) 这样低级用户就可以通过分析响应时间获取高级用户的信息.

然而，在实际应用中这种隐通道的构造较为困难，带宽也比较低. 因为，高级用户传递信息需要做足够多的操作保证缓冲区接近满，同时操作还会受到并发事务的影响. 我们可以通过审计、干扰等方法来进一步限制类似的隐通道的带宽. 从实际系统设计开发而言，采用综合的方案一般更为简单，从而代价更低.

6 结　论

在对核心化体系结构的多级安全数据库系统的研究过程中，作者发现其中存在未决提交事务日志写出依赖问题. 本文给出该问题的定义，分析了它的产生原因，并提出了解决方案. 方案的主要思想是在体系结构中引入一个可信的日志协调实体，维护全局的未决提交事务依赖关系，并协调各个 DBMS 实例的提交日志记录写出操作，保证被依赖的提交日志记录先于依赖的写入持久存储，从而避免发生不一致的状态. 给出了方案的实现算法，并通过实例进一步解释了复杂算法. 证明了算法可以解决本文提出的问题. 通过分析，我们发现引入相应的机制需要付出一定的代价，包括开发代价、性能损失代价和隐通道的引入，但总体而言这些代价是可接受的.

为了深入进行研究，我们还需要展开两项工作：(1)需要通过进一步的实验测定引入机制对系统性能的影响，确定影响性能的主要因素，并改进算法以减少对系统性能的影响；(2)为了满足安全保障级别的要求，设计一个能够防止类似隐通道问题的方案.

参　考　文　献

1 Notargiacomo L.. Architectures for MLS database management systems. In: Abrams M. D., Jajodia S., Podell H. J. eds. Information Security: An Integrated Collection of Essays. Los Alamitos, California, USA, 1994, 439～459

2 Alturi V., Jajodia S., Ceorge B.. Multilevel Secure Transaction Processing. Boston/Dordrecht/London: Kluwer Academic Publishers, 2000

3 Denning D. E., Lunt T. F., Schell R. R., Shockley W. R.. A multilevel relational data model. In: Proceedings of the IEEE

Symposium on Security and Privacy, Oakland, 1987, 220～234
4 Denning D. E., Lunt T. F., Schell R. R., Shockley W. R., Heckman M.. The seaview security model. In: Proceedings of the IEEE Symposium on Security and Privacy, Washington DC, 1988, 218～233
5 Xu Zhen. Research on multi-policy secure database system [Ph. D. dissertation]. Institute of Software, Chinese Academy of Sciences, Beijing, 2004(in Chinese)
(徐 震. 支持多安全策略数据库管理系统研究[博士学位论文]. 中国科学院软件研究所, 北京, 2004)
6 Air Force Studies Board. Multilevel Data Management Security. Washington DC: National Research Council. National Academy Press, 1983
7 Atluri V., Jajodia S., Bertino E.. Transaction processing in multilevel secure databases with kernelized architecture: Challenges and solutions. IEEE Transactions on Knowledge and Data Engineering, 1997, 9(5): 697～708
8 Atluri V., Jajodia S., Mccollum C., Mukkamala R.. Multilevel secure transaction processing: Status and prospects. In: Samarati P., Sandhu R. eds. Database Security: Status and Prospects X. London: Chapman & Hall, 1997, 79～98
9 Kang I. E., Keefe T. F.. On transaction processing for multilevel_secure replicated databases. In: Proceedings of the 2th European Symposium on Research in Computer Security, Toulouse, France, 1992, 329～347
10 Kang I. E., Keefe T. F.. Transaction management for multilevel secure replicated databases. Journal of Computer Security, 1995, 3(2): 115～145
11 Gray J.. Notes on database operating systems. In: Bayer G. S. R., Graham R. M. eds. Operating Systems, An Advanced Course. Berlin: Springer-Verlag, 1978, 393～481
12 Abdallah M., Guerraoui R., Pucheral P.. One-phase commit: Does it make sense? In: Proceedings of the 1998 International Conference on Parallel and Distributed Systems, CA, 1998, 182～192
13 Al-Houmaily Y. J., Chrysanthis P. K.. 1-2PC: The one-two phase atomic commit protocol. In: Proceedings of the 2004 ACM Symposium on Applied Computing, Nicosia, Cyprus, 2004, 684～691
14 Gray J., Reuter A.. Transaction Processing: Concepts and Techniques. San Mateo, California: Morgan KaufMann, 1993
15 Bell D. E., Lapadula L. J.. Secure computer systems: Unified exposition and multics interpretation. The Mitre Corp, Burlington Read, Bedford, MA 01730, USA: Technical Report: MTR-2997, 1976
16 Goguen J. A., Meseguer J.. Security policies and security models. In: Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, Los Alamitos, CA, 1982, 11～20

XU Zhen, born in 1976, Ph. D., assistent professor. His research interests include access control system security and trusted computing.

ZHANG Min, born in 1975, Ph. D. candidate, assistent professor. His research interests include datebase security and security evaluation.

Background

This paper is a part of the research work of LOIS secure DBMS(Supported by the National Natural Science Foundation of China under grant Nos. 60025205, 60273027 and the National High Technology Research and Development Program (863 Program) of China under Grant No. 2004AA147070, 2002AA141080) which is a kernelized DBMS. LOIS SDBMS support multi-level transaction processing. However there exists the problem of dependency of flushing of pending commit transaction log, which has not been seriously studied by now. This is partly because some researchers believe recovery can be carried on in each DBMS instances independently. Kang I. E. *et al*. present a scheme based on UNDO log in 1992, in the Kang's paper a new transaction can begin only when all DBMS instances finish recovery. And later they provide the prototype which implemented the scheme. However the above work concentrated on atomic commit problem. In kernelized DBMS, DBMS instances are relatively autonomic, which made its transaction processing like that of distributed systems in a certain degree. Such field has been extensively studies and several schemes exist such as 2PC, 1PC, 1-2PC. However, such schemes all brought about a lot of disk write which may lead to prohibitive performance penalty to the system.

So, in this paper, the authors presents a trusted log coordinator to maintain the global dependency of pending commit transactions' commit log record of which is still cache and coordinate the log writing of different DBMS instances to guarantee that the depended commit log records are written out before the depending ones. In this way, the problem is solved with acceptable covert channel. However, in order to satisfy the requirements of higher security level, such covert channel should be eliminated, which is also further work.