文章编号: 1001-2486 (2000) 01-0007-04

一种实用有效的恢复策略WAL_P*

王意洁, 胡守仁

(国防科技大学并行与分布处理国家重点实验室, 湖南 长沙 410073)

摘 要: 从面向对象数据库的体系结构和事务模型的角度出发, 提出了一种实用有效的恢复策略 WAL_P, 该策略实际应用于自行研制的面向对象数据库系统 KDOODB 中, 为事务管理的有效实现奠定了基础。

关键词: 面向对象数据库; 恢复; 嵌套事务; 页

中图分类号: TP311.132.4 文献标识码: A

A Practical and Efficient Recovery Strategy WAL_P

WANG Yi-jie, HU Shou-ren

(National Key Laboratory for Parallel & Distributed Processing, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: In view of the system structure and transaction model of OODB, a practical and efficient recovery strategy — WAL_P is proposed, which is implemented in the object-oriented database system KDOODB, and provide powerful support for the efficient implementation of the transaction management.

Key words: object-oriented database; recovery; nested transaction; page

嵌套事务模型[1,2]是一种复杂度较高、灵活性较大、应用领域较广的事务模型。嵌套事务中包含着具有开始点和结束点的其他事务, 这些嵌入的事务称为子事务。嵌套事务的深度可以是任意的。不包含于任何事务中的根事务称为顶层事务 (top-level transaction), 拥有子事务的事务称为父事务, 事务的祖先—后代关系是父子关系的传递闭包。嵌套事务的行为遵循以下规则:

• 提交规则: 只有父事务才能访问子事务提交之后的结果。子事务的最终提交 (如将结果释放到外界), 仅当本身已局部提交且所有的祖先事务 (包括根事务) 均最终提交。

• 回退规则: 如果任何一级上的 (子) 事务回退, 则它的所有子事务均回退。此规定带来的直接后果为, 如果根事务回退, 所有的事务全部回退, 不管它们是否实施了局部的提交。

• 可见性规则: 子事务一旦提交, 父事务就可观察到它作出的所有改变。父事务在子事务开始前作出的改变对子事务是可见的。子事务并发运行时, 变化不向兄弟事务显示; 否则, 子事务一旦提交, 其变化向兄弟事务显示。

目前, 既采用页服务器结构又支持嵌套事务模型的 OODB 系统还为数不多, 而且它们大多没有充分考虑事务管理的实现效率问题, 尤其是恢复的有效实现。顺应 OODB 的发展趋势, 我们自行研制的面向对象数据库系统 KDOODB[3,4], 采用页服务器结构, 同时, 遵照 ODMG-93 国际标准提供对嵌套事务模型的支持。另外, 针对嵌套事务模型的特点, 我们提出了一种采用先写日志协议 (write ahead log protocol)[5] 的基于页的恢复策略 WAL_P, 它实际应用于 KDOODB 系统中, 为 KDOODB 系统的有效实现奠定了基础。

1 WAL_P 的数据结构

1.1 日志序号

* 收稿日期: 1999-06-29
基金项目: 国家自然科学基金资助项目 (69903011); 国家部委项目资助 (15.4.1)
作者简介: 王意洁 (1971-), 女, 助理研究员, 博士。

基于页的日志序号是指: 每个页有一个唯一的日志序号, 这个日志序号将页的状态改变与日志记录联系起来。在系统重启时, 通过比较日志记录和页上的日志序号就可以判断更新操作的结果是否存入磁盘数据库, 并进一步决定是否进行 redo 操作和 undo 操作:

• 对于圆满事务的 redo 操作而言, 如果页上的日志序号小于日志记录的日志序号, 则说明对该页的更新并没有存入数据库, 那么, 需要进行 redo 操作;

• 对于夭折事务的 undo 操作而言, 如果页上的日志序号大于等于日志记录的日志序号, 则说明对该页的更新已经存入数据库, 那么, 需要进行 undo 操作。

1.2 日志记录

日志文件由一系列记录构成, 日志记录主要包括以下几个域:

• LSN (日志序号): 唯一标识一个记录, 它是单调递增的;

• Type (类型): 标识记录的类型;

• Subtype (子类型): 标识记录的子类型;

• Tid (事务标识): 标识该记录所属的事务;

• Pid (页标识): 标识被更新的页, 在 Update 和 CLR 类型的记录中出现;

• Last_LSN: 指向属于同一事务的前一个记录, 对于顶层事务的 Start 类型的记录, 它为空;

• RedoNext_LSN: 一方面, 它为有效地进行选择性的 redo 操作提供支持; 另一方面, 它出现在 CLR 类型的记录中, 提供对 undo 操作幂等性 (idempotence) 的支持;

• UndoNext_LSN: 只在 CLR 类型的记录中出现, 指向事务卷回时需处理的下一个记录;

• Length: 标识该记录的长度;

• Data: 包括 redo 和 undo 操作所需的数据 (如: 前像和后像等)。

1.3 事务表

事务表是事务管理的基础, 在事务执行过程中, 事务表是被动态更新的。在设置检查点时, 事务表将被写入日志文件, 等到系统重启时用于确定圆满事务和夭折事务。事务表包括以下几项:

Tid: 事务标识; Last_LSN: 记录事务的最后一个日志记录的序号; State: 标识事务的当前状态。

1.4 脏页表 (dirty page table)

脏页表管理已经在缓存中被更新但尚未存入磁盘数据库的页 (即: 脏页)。每当某页被存入磁盘数据库时, 就从脏页表中删除与其对应的项。脏页表包括以下两项:

Pid: 页标识; Dirty_LSN: 记录该页被调入缓存后第一次被更新的逻辑时间。

1.5 页的结构

每页除了一个页标识 Pid 外都有一个 LSN 域, 用于记录该页最后一次被更新的逻辑时间 (即: 与最后一次更新对应的日志记录的序号), 页的更新必然伴随着其 LSN 域的更新。

2 事务的正常执行

2.1 更新操作

更新操作的执行将引起一个日志记录的产生以及页中 LSN 域的更新。一个事务的日志记录是通过后向指针链接的, 后向链接是通过记录的 Last_LSN 域实现的, 它的值是根据事务表的 Last_LSN 域来设置的。

2.2 补偿日志记录

大多数采用先写日志协议的系统都会实现补偿日志记录, 原因是: (1) 有利于恢复管理器记录卷回过程中的 undo 操作; (2) 为介质恢复提供支持; (3) 保证恢复处理的幂等性。每当撤销 (undo) 一个更新操作时, 就为其产生一个补偿日志记录, 并将页中的 LSN 域更新为补偿日志记录的序号。这样, 页的 LSN 域始终是递增的, 且通过比较 LSN 就可确定 undo 操作的结果是否存入磁盘数据库。

补偿日志记录的创建与一般的日志记录相似。由于不需对其进行 undo 操作, 所以它的信息主要用于 redo 操作。它的 UndoNext_LSN 域的值与被它补偿的记录的 Last_LSN 域的值相同, 即指向事务卷

回时需要处理的下一个记录。它的 RedoNext_LSN 域指向被补偿的记录。

2.3 事务的全部卷回

事务夭折将引起事务的全部卷回, 即: 根据日志自后向前卷回所有的更新操作, 并且每卷回一个更新操作就产生一个补偿日志记录, 直到遇到该事务的 Start 类型日志记录, 产生一个 Abort 类型日志记录, 然后结束卷回过程。

2.4 事务的部分卷回

WAL_P 支持事务的部分卷回, 即首先在事务中设置若干保存点, 然后可以规定事务卷回到指定的保存点。事务部分卷回的实现提高了事务管理机制的灵活性和有效性。部分卷回与全部卷回的唯一区别是: 前者的卷回过程到指定的保存点终止, 后者的卷回过程到 Start 类型的日志记录终止。在部分卷回之后, 事务可以继续进行正常的处理并提交。在部分卷回存在的前提下, 事务的卷回过程中就会遇到两种日志记录: 非 CLR 记录和 CLR 记录。若遇到非 CLR 记录, 那么通过它的 Last_LSN 域得到下一个要处理的记录; 若遇到 CLR 记录, 那么通过 UndoNext_LSN 域得到下一个要处理的记录, 这样就跳过了已撤销的记录, 提高了恢复的效率。

2.5 事务的提交和夭折

KDOODB 系统采用两段锁协议进行并发控制。根据嵌套事务行为规则, 事务提交处理过程是:

• 若该事务有父事务:

(1) 利用文献 [3] 中提出的事务标识分配策略可以很容易地找到其父事务;

(2) 产生一个 Prepare 类型的日志记录, 并在事务表中设置它的状态为 pre-commit, 更新其 Last_LSN 域为 Prepare 记录的 LSN;

(3) 更新父事务日志——将当前事务的日志链接到父事务的日志上;

(4) 更新父事务状态——在事务表中, 根据当前事务 Last_LSN 域更新父事务的 Last_LSN 域值;

(5) 将当前事务拥有的锁转交给父事务。

• 若该事务没有父事务:

(1) 确保它的日志记录已全部写入日志文件;

(2) 将一个 Prepare 类型的日志记录写入日志文件, 并在事务表中设置它的状态为 pre-commit, 更新其 Last_LSN 域为 Prepare 记录的 LSN;

(3) 将一个 Commit 类型的日志记录写入日志文件, 释放它拥有的锁, 并从事务表中删除它和它所有 pre-commit 状态的子事务。

事务夭折的处理过程是:

(1) 事务全部卷回;

(2) 释放它拥有的锁, 并从事务表中删除它和它所有 pre-commit 状态的子事务。

3 系统重启

当发生系统故障时, 需要进行系统重启以使数据库恢复到一致状态, 从而保证事务的原子性 (atomicity) 和持久性 (durability)。基于 WAL_P 的系统重启可以分为三个阶段 (如图 1): 分析阶段、undo 阶段和 redo 阶段, 其目的是撤销夭折事务的更新操作 (保证原子性) 和重做 (redo) 圆满事务的更新操作 (保证持久性)。许多恢复策略在处理系统故障时都或多或少地进行了不必要的 undo 和 redo 操作, 这严重影响了恢复的效率。WAL_P 充分考虑了恢复的效率问题, 在系统重启时进行了有选择的必要的 undo 和 redo 操作。

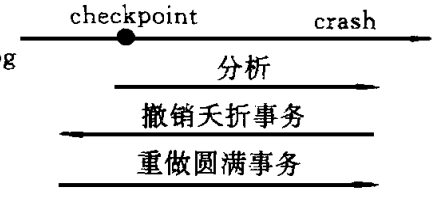

图 1 基于WAL_P的系统重启

Fig 1 System restart based on WAL_P

3.1 分析阶段

分析阶段是为下面两个阶段作准备的, 它从磁盘的指定位置取出最后一个检查点的 LSN, 从该检查点取出信息来初始化事务表和脏页表; 然后, 从该检查点开始依次读取每个日志记录并进行处理,

直到日志文件的结尾。

3.2 undo 阶段

undo 阶段的任务主要是: 1) 撤销夭折事务; 2) 找出需要重做的更新操作, 为 redo 阶段作准备。实际上, 没有必要将夭折事务的所有更新操作都撤销, 也没有必要将圆满事务的所有更新操作都重做, 应该根据它们对数据库状态的影响情况有选择地进行。简言之, 就是从 Last_record 开始自后向前依次读取每个记录, 并进行判断和相应的处理: 如果该记录对应于夭折事务的更新操作, 且更新结果已存入磁盘数据库, 则进行 undo 操作, 并产生一个 CLR 记录; 如果该记录对应于圆满事务的更新操作, 且更新结果未存入磁盘数据库, 则将其加入前向链, 为 redo 阶段作准备; 另外, 如果是 Start 记录, 则从事务表中删除与该事务对应的项。在 undo 阶段建立前向链的主要目的是提高 redo 阶段的效率。

对系统故障恢复的较高要求是: 即使在系统重启时反复发生故障, 数据库仍然能恢复到相同的一致状态。为了达到这个要求, 需要恢复策略具有幂等性。

为了保证 undo 阶段的幂等性, 需要严格区分夭折事务的各种更新操作并采取不同的处理方式:

(1) 更新操作的结果未存入磁盘数据库

——不作任何处理, 因为更新操作未对数据库产生影响;

(2) 更新操作的结果已存入磁盘数据库, 且没有相应的补偿日志记录

——撤销更新操作, 并产生一个相应的补偿日志记录;

(3) 更新操作的结果已存入磁盘数据库, 且有相应的补偿日志记录, 但其对应的 undo 操作的结果为存入磁盘数据库 (即: 更新操作并未真正卷回)

——撤销更新操作, 并将页的 LSN 域设置为已有的补偿日志记录的 LSN;

(4) 更新操作及其 undo 操作的结果均已存入磁盘数据库 (即: 更新操作真正卷回)

——不作任何处理, 因为更新操作对数据库的影响已被消除。

3.3 redo 阶段

redo 阶段依据前两个阶段的结果选择性地进行某些 redo 操作, 从而保证事务的持久性。如前所述, 它的准备工作已由 undo 阶段完成 (建立前向链), 它只需获取前向链的头, 然后依次重做前向链中的每一个操作。由于前向链是存于日志缓冲区中的, 所以 redo 阶段不必访问日志文件, 从而节约了大量的时间开销。

4 结论

WAL_P 以先写日志协议为基础, 以页作为恢复粒度, 能够有效地处理各种故障。它具有一定的灵活性, 对日志类型和缓存的管理策略都没有严格的限制; 它在嵌套事务的正常执行过程中进行了周密细致的准备工作, 从而使嵌套事务的恢复准确有效; 它比较充分地考虑了恢复的效率问题, 在各个阶段的处理工作都体现了“效率至上”的准则; 它也有较好的可扩展性, 能够很容易地将其改为以对象为粒度的恢复策略, 从而为基于对象的事务管理提供支持。

面向对象数据库系统 KDOODB 采用页服务器结构, 符合 ODMG-93 国际标准, 支持嵌套事务模型。WAL_P 在 KDOODB 系统中的实际应用表明它是一种实用有效的恢复策略。

参考文献:

[1] 王意洁, 王勇军, 胡守仁. 面向对象数据库管理系统中的事务管理 [J]. 计算机科学, 1996, 23 (6): 59-62.

[2] Rick Cattell, et al. The Object Database Standard: ODMG-93 [S]. Morgan Kanfmann, 1994.

[3] 王意洁. 面向对象数据库的并行查询处理与事务管理 [D]. 长沙: 国防科技大学研究生院, 1998.

[4] 王意洁, 钟武, 章文嵩, 胡守仁. 一个基于 Win32 平台的面向对象数据库系统 [J]. 计算机科学, 1999, 26 (2): 30-34.

[5] Gray J N, Reuter A. Transaction Processing: Concepts and Techniques [M]. Morgan Kaufmann Publishers, USA, 1993.