数据库 并发控制 (6)

计算机科学 1994Vol. 21№. 3

26-30 TP311.13

高级数据库系统中基于语义的并发控制

孙建伶 何志均
(浙江大学人工智能研究所 杭州 310027)

摘 要 This paper surveys the semantic-based concurrency control for advanced database systems, which aimed to eliminate the limitation of conventional concurrency control and to meet the requirements of advanced database applications. Covered in the paper are those approaches that using transaction semantic, that using abstract data type semantic, and that supporting for cooperation. The current situation and the future directions of the semantic-based concurrency control are also analysized.

关键词 Database System, Concurrency Control, Semantic.

一、数据库的一致性和传统并发控制技术

维护数据库的一致性是数据库管理系统的根本任务之一。传统数据库系统维护一致性的途径是事务。事务的本质特征是其原子性，它既是逻辑上的原子单位(一段相对完整任务的操作序列)，又是并发控制的原子单位(各事务的运行互不干扰)，也是恢复的原子单位(一个事务的运行或者成功，或者不对数据库产生任何影响)[1]。

传统数据库对事务的调度基于可串行化原则，只有等价于一个串行调度的调度才是一个可以保证数据库一致性的调度，结果是并发事务被一个接一个地串行运行。

数据库系统并发控制的任务就是在保证数据库系统一致性的前提下，提高系统的并行性能；减少系统的额外开销。传统并发控制技术可分成三大类：

1. 悲观方法 其基本前提是假设数据访问的冲突频率较高。悲观并发控制使一个产生冲突的事务处于等待状态，只有当冲突消解后处于等待状态的事务才可以继续运行。实现悲观并发控制的一般方法是锁机制，二阶段锁协议(2PL)[1]是保证事务运行可串行性的充分条件。但是在没有任何额外语义(除了抽象的读写操作)可供利用的情况下，2PL又是保证事务运行可串行化的必要条件。

基本 2PL 无法避免死锁的产生，消除死锁的手段是放弃事务。为了避免死锁产生时的连锁放弃(cascade abort)，可以采用严格2PL，即事务拥有的锁只有在事务提交(或放弃)后才释放。显然，这是以降低并发度为代价的。

锁机制的其它变种有：如果预先知道数据的存取次序满足某种偏序关系，则利用树协议[1]可以产生非 2PL 的调度，以提高并行性；如果数据有多种粒度，则利用多粒度锁协议[1]可以减少上锁的数目，以减少系统代价。

悲观并发控制还可以利用时间戳排序法[1]实现，这种方法可以避免死锁，却以事务的放弃为代价。

以上方法是以事务的等待和(或)放弃作为保证可串行性的手段，而基于多版本[1]的方法通过保留数据的旧版本来避免事务的等待(只读事务不用等待)，减少事务的放弃(只读事务不会放弃，读/写事务可能放弃)。多版本方法通常与锁机制或时间戳方法结合在一起使用。多版本的缺点是系统代价较高，更为严重的是它用事务的回退(rollback)替代等待，作为解决冲突的根本手段，代价高昂。

孙建伶 博士，主攻数据库、人工智能和软件工程。

2. 乐观方法　其基本前提是假设数据访问的冲突频率较低。乐观并发控制[1]允许事务先不理会冲突而运行，当事务提交时系统才进行有效性验证。系统在事务开始运行时赋予每个事务一个时间戳，事务在其生存期内对数据库应该有一个一致的视图。在事务提交时，系统根据时间戳检验这种数据库的一致性视图是否在事务开始和结束这段时间里改变了，如果没有改变，事务提交，否则，事务放弃。

3. 自适应并发控制　自适应并发控制方法是根据事务的冲突频率选用悲观或乐观方法。文[2]采用事务的长短作为计算事务冲突频率的依据。文[3]采用数据项的访问频率作为计算事务冲突频率的依据。但是这种方法的系统代价十分高昂，可能抵消潜在的好处。事实上，真正使用自适应并发控制的 DBMS 未见报告。

综上所述，传统并发控制可以总结为：以数据的竞争性使用为假设，以原子化和可串行性为原理，以等待和放弃为基本手段。

二、新的并发控制需求

高级数据库应用，如 CAD、CASE，对并发控制提出了新的需求[4]：

1. 支持长事务　这种事务通常持续几小时乃至几天。长事务意味着事务间冲突和死锁概率的增加[5]。以等待和放弃为基本手段的传统并发控制技术无法满足长事务的需求。

2. 支持用户控制　在 CAD 等应用领域中，用户的工作通常是交互、非确定的。并发控制机制应该为用户提供丰富的控制手段，如开始一个事务、交互地在事务中运行操作、动态重构事务、随时提交或放弃事务等。事务的非确定性意味着并发控制机制除非实际运行事务并检验它的结果，否则它无法确定事务的运行是否会破坏数据库的一致性。这可能导致下面的情形：用户很有兴致地工作了一段时间，等到后来准备提交事务时才发现工作中有操作会破坏数据库的一致性。用户当然不希望全部放弃所做的工作，而希望能显式地取消某些操作的结果以满足数据库一致性的要求。

3. 支持协作过程　在 CAD 等设计环境中，多个用户分工合作完成一项大的设计任务。这种协作过程以合作生产数据资源为特征，它不同于传统数据库资源的竞争性使用。协作既是不同于传统数据库的新的需求，也为并发控制提供了新的途径。

数据库的传统并发控制技术无法满足以上新的并发控制需求，根本原因在于没有利用应用层的语义。DBMS 将数据的访问抽象为读、写两种操作，把事务看成是由读、写操作组成的原子序列，通过串行化方法来保证并发事务的正确性。但是可串行性降低了事务的并发度，给应用带来了过多的约束。事实上，将调度的一致性等同于事务的可串行性是一个极大的限制。如果 DBMS 能利用更多的应用层语义，将应用层定义的一致性约束条件作为维护数据库一致性的依据，那么就可能冲破可串行性带来的桎梏，构造出非串行化的但却是一致的调度，从根本上提高事务的并发度。近几年数据库并发控制的研究基本上围绕着应用层语义的利用，这就是本文所讨论的基于语义的并发控制。它一般分成两大类：基于事务语义的并发控制和基于抽象数据类型语义的并发控制。支持协作过程的并发控制也属于一种基于语义的并发控制，但由于它专门针对工程设计过程，故把它单独列出。

三、基于事务语义的并发控制[4]

在这种途径中，事务的并发性质按照事务的语义和事务操作数据的模式来定义，又可以分为以下两类：

1. 对可串行化技术的扩展。它在保持可串行化的前提下利用事务的语义，当无事务语义可资利用时又回归到传统的方法。属于这一类的并发控制技术有：

(1)利它锁(Altruistic locking)[6]。是对基本两阶段锁协议的扩展。它利用了事务的数据存取方式的信息来决定可以释放事务的哪些资源,以便马上让其它事务使用。比起基本2PL,它允许事务有条件地释放某些它不再需要的资源,因而具有比基本2PL更高的并发度。利它锁协议需要假设事务是编程方式的,因而不适合用户交互控制的场合。

(2)快照验证[7]。是对乐观并发控制(也称验证技术)的扩展。基本的验证技术对冲突的定义太弱,快照验证技术进一步将冲突划分为严重冲突和非严重冲突。严重冲突需要事务的重新运行,而非严重冲突则不需要事务的重新执行。

(3)嵌套事务[8]。是对平面结构事务的扩展,它利用了事务自身的嵌套结构,以开发事务内部的并发性,并将故障局部化。

(4)多层事务[9,10]。是对单层事务的扩展。传统事务是单层的,由基本的读、写操作序列组成。多层事务中操作分层:一个层次上的每个操作由它下一层的操作序列构成,多层事务不仅可以提高并发度,而且也是并发控制模块化的有效途径。

多层事务与嵌套事务的区别主要有二点:(a)多层事务通常是DBMS实现上的考虑,它有预先确定的层数,而嵌套事务是用户组织事务的考虑,通常没有预先确定的层数。(b)多层事务有抽象的概念,层数越高,操作的抽象性越强,而嵌套事务则未必有抽象概念。

2. 对可串行化的放松。在利用事务语义的同时不坚持可串行化原则,因而具有更高的并发度。Garcia-Molina提出的利用语义知识的并发控制[11]就是这一类的代表。Garcia-Molina定义了四种类型的事务语义:(1)事务类型;(2)每一类事务的相容事务类集合;(3)事务分划成小的步骤;(4)事务中某些操作步的补偿步。前三类语义是陈述性的,后一类语义是过程式的。这四种语义知识定义了事务之间的交织。利用这些语义,Garcia-Molina定义了与传统原子性相对的概念,即语义原子性。一个事务被认为是语义原子性的,是指它的所有操作步都运行了或者它的所有操作步最终跟随着各自的补偿步。Farrag对Garcia-Molina的工作做了精化[12],用事务断点概念代替事务相容概念,具有比Garcia-Molina更精细的事务交织模型。

其它属于这一类的并发控制技术还有[4]:·Sagas;·冲突谓词正确性;·事务动态重构等。

四、基于抽象数据类型语义的并发控制[13]

抽象数据类型用其上定义的操作来表征自身,这些操作用来存取抽象数据类型的实例。基于抽象数据类型语义的并发控制按照类型及其上的操作的语义来定义抽象数据类型上的并发性质。比如,队列对象,在其上定义的remove操作(从队列头移去一个元素)和append操作(从队列尾增加一个元素)是不冲突的(但在传统并发控制技术看来是冲突的)。

在这种途径中,并发事务的交织由抽象数据类型上定义的操作冲突关系所约束,我们说操作对(O1,O2)处在类型的冲突关系中,是指O1和O2不能并发执行。操作之间冲突关系的定义方式通常有以下三种:

1. 可交换性(commutativity)[14]。基于可交换性的冲突关系意味着如果两个操作不可交换(即包含这两个操作的序列的运行结果和这两个操作的次序有关),则这两个操作冲突。可交换性又可细分为前向可交换性和后向可交换性,这两者的区分很微妙,依赖于恢复算法[13]。

2. 序列依赖[15]。基于序列依赖的冲突关系意味着如果一个操作O1依赖于另一操作O2(即如果在一个操作序列中O2出现在O1的前面会使O1失效),则操作O1与O2冲突。

基于序列依赖的冲突关系比基于可交换性的冲突关系要弱,具有更高的潜在并行度。

3. 可恢复性[16]。假如用 state(O,S)表示一个对象在 S 状态时运行方法 O 后产生的新状态;return(O,S)表示一个对象在 S 状态下运行方法 O 的返回值,则可恢复性定义为:考虑对象的两个操作 O1 和 O2,O1 在状态 S 下的运行紧跟着 O2 的运行。那么操作 O2 对于操作 O1 是相对可恢复的,当且仅当对这个对象的所有操作 S 满足:

return (O2,state(O1,S))=return(O2,S)

可交换性意味着可恢复性,但可恢复性并不意味着可交换性。利用可恢复性使得不可交换的操作也能一起运行,可减少事务等待时间。

五、支持协作过程的并发控制

协作过程中对数据的操作方式与传统商用数据处理方式很不一样,后者体现为数据的频繁(事务短而多)竞争性(冲突频率高)共享;前者体现为数据资源的协作性生产,事务长而少,各协作参加者对共有数据的访问有某种局部性。这也是应用层语义的一种。

协作过程并发控制需要为协作者对数据库的局部占有及协作者之间的信息交流提供强有力支持。已经提出了很多种支持协作过程的并发控制策略,按照对协作的支持程度可以分为以下两大类:

1. 支持设计者之间的协调。在这种模型中,设计者之间分工完成一个大的设计项目。在大多数时间里,各设计者独立工作于各自负责的子项目中,但有时需要发生交互以综合各自的工作。因此需要有一些规则以协调多个设计者对项目数据库的并发访问,保证各设计之间的工作不会相互干扰。支持协调的并发控制一般都是基于版本/配置、检入/检出机制的。对象版本不仅是设计过程本身所要求的,而且也是支持协调的有效手段。各设计者从共有数据库中检出一些对象,到其私有数据库中,他们工作在不同的版本上,协调机构或者对并发的检入/检出施加一些限制,或者在对象检入时负责版本的融合。按协调的策略可以分成悲观协调与乐观协调二大类。

(1)悲观协调。在这种模型中,设计项目数据库由公有数据库和私有数据库组成。公有数据库被所有设计者所共享,而每个私有数据库由一个设计者拥有。每个设计者在他的私有数据库中开始一个长事务。当长事务需要访问公有数据库中的对象时,它以某种访问方式(如读、写、删除)检出对象。公有数据库检查其访问方式是否与其它长事务对同一个对象的访问方式冲突,若冲突则写上通知设计者。

悲观协调和传统锁机制有些类似,即都假设对象上的访问冲突频率较高,不同之处是悲观协调允许获准检出的数据可以在较长时间里保存在私有数据库中,而且,检出不获准时长事务并不进入等待,而是马上通知设计者,以便视情况采取不同的对策。

(2)乐观协调。与悲观协调策略不同,乐观协调允许在同一对象上进行相互冲突的访问。对象的操作结果生成新的对象版本。协调机制提供将同一对象不同版本融合在一起的手段。Sun 公司的网络软件环境(NSE)[17]就是采用乐观协调法的典型例子。

2. 支持紧密协作。在这种模型中,设计人员分工更细、相互之间的信息交流更频繁、数据访问的冲突频率更高。

支持紧密协作的并发控制策略种类繁多,而且有些技术相互交叉,有些区别起来很微妙。比如有:

(1)通知(Notification)。它允许用户要求数据库提供通知服务,即当某些设计者感兴趣的事件(如对象删除、对象检入、对象检出等)发生时设计者能得到通知,以便了解设计进程,并为交互式冲突消除提供辅助。

(2)成组范型(The Group Paradigm)[18]。将设计者分成许多小组,小组成员之间通过局部的并发控制达到紧密协作;而各小组之间通过全局的并发控制得到协调。

支持紧密协作的其它并发控制策略还

有[4]：(3)面向小组的 CAD 事务；(4)协作 CAD 事务；(5)事务组；(6)参加者事务。

以上所论及的支持协作的并发控制策略相对成熟，得到最广泛实现的是基于版本/配置和检入/检出的悲观协调控制策略，这主要是因为这种策略十分简捷；乐观协调法的多版本融合是一个难点，但如果不是将两个版本融合为一个版本，而是将不同版本融合进同一版本树中，则实现起来要容易些。支持紧密协作的并发控制几乎没有在 OODB 中得到完整的实现，这主要缘于其对设计组织方式的过分假定以及策略本身的繁琐和实现上的困难。

六、评论

并发控制机制可利用的应用层的语义是极为丰富的。包括对象上操作的冲突关系、事务的嵌套、层次结构、事务之间的交织、事务分类、事务的可中断点、事务对数据的访问模式，事务之间的协作关系，等等。

基于事务语义的并发控制和基于抽象数据类型语义的并发控制分别利用应用层语义的一个方面，前者利用了应用程序中数据操作及事务的语义，后者利用了模式定义中类型及操作方法的语义。基于事务语义的并发控制有一个中心控制机构，并且增加新的语义比较方便；基于抽象数据类型语义的并发控制易于实现并发控制的分散化，增加新的语义需要改动数据模式，不够方便。

面向对象数据库系统从原理上说可以包容这两种类型的并发控制。特别是，基于抽象数据类型语义的并发控制很好地反映了数据库的面向对象特征，是数据库的面向对象特征对并发控制的影响的体现。

然而，应用层语义为并发度的提高所带来的潜力目前还远远不能发挥出来。应用层的语义颇有些类似人工智能中的知识，存在着知识获取和知识运用效率这两个问题。无论是事务的语义，还是抽象数据类型的语义，都是不容易抽取和把握的。这对应用系统的设计是一个极大的障碍。相比而言，传统的以读、写为基本操作集，以可串行化为基本原理的并发控制，虽然其系统性能(吞吐率)在新的应用领域中较低，但用户使用起来极为方便(可以只显式调用事务的开始命令和结束命令而不需其它的额外操作)。另外，语义的运用效率也是一个严重问题，弄不好，系统的代价甚至可能超过运用语义所带来的好处。

目前，大多数基于语义的并发控制还仅仅是概念上、理论上的结果，有些又十分复杂以至无法使用。而且，这种途径主要针对长事务的并发控制，对用户控制和协作过程的支持很弱。

在所有的基于语义的并发控制中，目前只有嵌套事务和多层事务得到实现。如 ObjectStore[19]实现了嵌套事务、DASDBS[20]实现了多层事务。这两种事务都是结构上比较简明的。大多数 OODBMS 的并发控制基本上还是基于传统的原理和方法，同时为了支持 CAD 等设计环境中的应用，这些 OODBMS 又提供了某些机制以支持协作过程的事务管理，主要是基于检入/检出的并发协调技术。

基于语义的并发控制今后将特别强调应用层并发语义的易于把握和易于使用；在保持理论严密性的同时将特别强调高效的实现技术。

七、小结

维护数据库的一致性是数据库管理系统的根本目标之一。传统数据库系统的事务管理以原子性和可串行化为特征，不能满足新的应用系统对并发控制的需求；为克服传统并发控制缺陷而提出的基于语义的并发控制为满足新的需求创造了潜在的可能性，但目前由于其复杂性和不成熟性而未能在 OODBMS 中得到实现。只有基于检入/检出机制的并发控制技术能较好地支持协作设计过程，在 OODBMS 中得到了比较多的实现。灵活、高效、实用的并发控制技术仍然是一个有待探寻的目标。(参考文献共 20 篇略)