基于SDML的存储块的封锁策略

钱 进 洪晓光

(山东大学 计算机科学与技术学院 济南250061)

摘 要 XML是当今流行的数据存储方式。传统的XML存储方式,是以流式文件的方式存储的。这样的方式有其优点,也有致命的缺点。我们基于SDML的存储方式,是以存储XML文档的结构为目的的存储方式,克服了流式存储方式访问上的缺点。并且在对SDML存储方式的修改的基础上,这篇论文提出了对XML文档的各种粒度的封锁,以适应对高度并发性的要求。

关键词 SDML,封锁,包含超链接的数据块,封锁粒度

The Locking Method Based on SDML

QIAN Jin HONG Xiao-Guang

(School of Computer Science & Techenology,Shandong University,Jinan 250061)

Abstract Nowadays,XML is the prevails way of storation. Traditional method of XML storation stores XML documents in flowing type way. Such a way has its advantages,there is a deadly shortcoming too. The stroe way we used based on SDML is by way of storing the structure of XML file as the memory of the purpose. This store way gets over the shortcoming of the flowing files storation. Even more,in this paper,we present a method based on modified SDML for various granularity locking to adapt applications.

Keywords SDML,Locking,Block locking,Locking granularity

1 研究背景

XML是当今流行的数据存储方式。在传统的XML存储中,数据是以顺序、平面的方式存储的。我们力求设计一种方式,以层次式的方式存储XML数据。我们的研究是在SDML的存储平台上进行的(SDML是我们开发实现的一种可以以层次式的方式存储XML文档的存储系统,系统存储细节我们将在后面描述),在这种存储策略下,可以支持较高的并发性以及较细的封锁粒度,并且对于字符数据中包含超链接的XML文档,SDML存储系统有特殊的策略,我们将在后面论述。封锁是实现并发控制的一个非常重要的技术。在关系数据库中,人们已经做了很多的研究,形成了较完善的封锁策略。在传统的XML数据库中,为了支持并发性操作,只能对整个的XML文档进行加锁。这样的加锁方式,封锁的粒度比较粗,在有较高的并发性要求的系统中,是不符合要求的。

·关系数据库中的封锁(引自《数据库系统概论》萨师煊,王珊)

所谓封锁就是事务T在对某个数据对象(例如在关系数据库中的表、记录;XML数据库中的元素、字符数据)操作之前,先向系统发出请求,对其加锁。加锁后事务T就对该数据对象有了一定的控制,在事务T释放它的锁之前,其他的事务不能更新此数据对象。

①排他锁又称为写锁。若事务T对数据对象A加上X锁,则只允许T读取和修改A,其他的事务都不能再对A加任何类型的锁,直到T释放A上的锁。这就保证了其他事务在T释放A上的锁之前,不能再读取和修改A。排他锁保证被修改的数据块没被其他的事务访问。

②共享锁又称为读锁。若事务T对数据对象A加上S锁,则事务T可以读取A但不能修改A,其他事务只能再对A加S锁,而不能加X锁,直到T释放A上的S锁。这就保证了其他事务可以读A,但在T释放A上的S锁之前不能对A做任何修改。这种模式的锁保证数据在读取其间不会因为其他事务操作而变为脏页。

因此这两种锁模式,是不相容的。由于两种模式的锁的不相容性,我们在有些情况下,需要对已经取得S锁的数据进一步加X锁。因此有些系统引入了升级锁(U锁)。

③升级锁。若事务T已经取得了数据对象A上

钱 进 硕士研究生,主要研究方向:数据库应用。

的S锁，并且要求进一步要求对A加上X锁，此时T可以向系统请求对A加U锁。当T取得了A上的U锁，在没有其他的事务对A加锁的情况下，可以把U锁直接升级为X锁，无需进行解锁工作。

意向锁的含义是如果对一个节点加意向锁，则说明该节点的下层节点正在被加锁；对任一个节点加锁时，必须对它的上层节点加意向锁。

（一）意向排他锁（IX锁）。如果对一个数据对象加IX锁，表示它的后裔节点拟加X锁。

（二）意向共享锁（IS锁）。如果对一个数据对象加IS锁，表示它的后裔节点拟加S锁。

（三）意向升级锁（IU锁）。如果对一个数据对象加IU锁，表示它的后裔节点已经取得了共享锁，并拟升级为X锁。

（四）共享意向排他锁（SIX锁）。如果对一个数据对象加SIX锁，表示对它加S锁，再加IX锁，即SIX=S+IX。

在引入了上述封锁机制后，我们的封锁机制就完整了，完整的封锁相容性表如下。

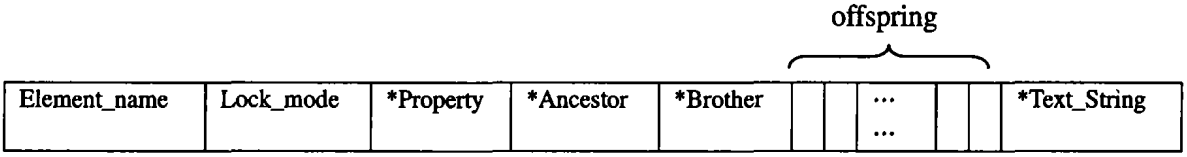
T1 \ T2	S锁	X锁	U锁	IS锁	IX锁	IU锁	ISX锁	未加锁
S锁	相容	不相容	不相容	相容	不相容	不相容	不相容	相容
X锁	不相容	不相容	不相容	不相容	不相容	不相容	不相容	相容
U锁	相容	不相容	不相容	相容	不相容	不相容	不相容	相容
IS锁	相容	不相容	不相容	相容	不相容	不相容	不相容	相容
IX锁	不相容	不相容	不相容	不相容	不相容	不相容	不相容	相容
IU锁	相容	不相容	不相容	相容	不相容	不相容	不相容	相容
ISX锁	不相容	不相容	不相容	不相容	不相容	不相容	不相容	相容
未加锁	相容	相容	相容	相容	相容	相容	相容	相容

2 基于SDML的存储块的封锁策略

2.1 SDML存储策略

为了适应即将提出的封锁策略，我们将在SDML存储系统的基础上作出一定修改，以适应应用。我们可以设计一个存储块的布局方式，以指针的方式来实现对XML数据的层次式的存储。

1. *标签的存储* 在我们的存储方法中，我们为每一个元素标签都建立一个元素头信息块，其中存储元素的标签、封锁模式、属性、直接子元素、祖先元素、兄弟元素以及当前元素结束标签之前的字符数据（若其结束标签之前的不是字符数据，而是另一个标签，则此指针为空）。此种存储方式类似于树存储策略中的孩子兄弟存储方法。为了适应对数据块的封锁机制，我们在每个数据块的头部设置一个封锁模式字段（1个字节），可以提供高达255种封锁方式。



Element_name元素名称，Lock_mode封锁模式（类同于存储块的封锁模式，可以提供255种封锁方式），*Property指向属性字段的指针，*Ancestor指向祖先元素的指针，*Brother指向兄弟元素的指针，*Text_String指向此元素结束标签前的字符数据的指针（若结束标签之前没有字符数据，则此指针为空），Offspring用来存放当前元素所有的直接子元素的标签以及指向相应子元素的指针和指向此子元素直接前驱字符数据的指针（若没有直接前驱字符数据，则此指针为空）。

2. *字符数据的存储* 可以采用关系数据库实现中变长纪录的存储策略。只是在头信息中附加封锁模式字段。

3. *属性的存储* 类似于字符数据的存储。同样需要附加封锁模式字段。

4. *附加块* 当字符数据中包含超链接的时候，我们可以为包含这段字符数据的元素标签设置一个附加块，用来存储所有的出现在字符数据中的超链接。为了实现并发性封锁，在每个附加块的头部设置一个字节封锁模式字段，可以提供255种封锁状态。

·根据需求可以对上述的存储策略进行修改

根据现存的XML文档的普遍特征，一个包含多段字符数据的元素是不常见的。因此，对标签头的存储中，可以根据需要去掉指向子元素标签的前驱文字的指针。一般情况，一个元素只有一段字符数据，因此可以只在标签头中设置一个字符数据指针，用来指向文字。这样可以减少很多的存储空间。

·存储容量分析

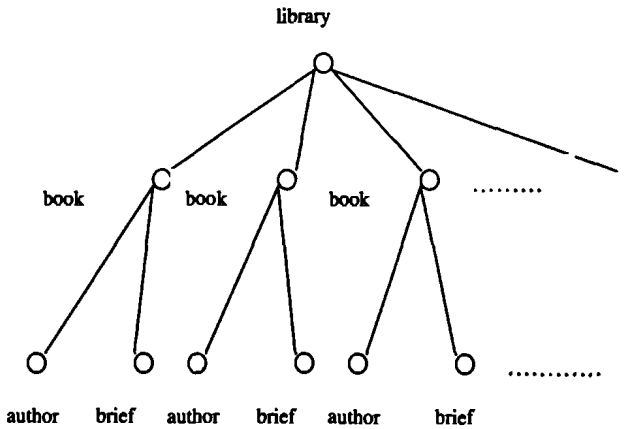
假设指针占用8字节，元素名字最多30个英文及数字字符，则元素名字字段占用30个字节。每一个子

元素共占用8＋8＋30＝46个字节，若不考虑块首信息所占的容量，则我们希望46n＋30＋1＋8＋8＋8＋8＜＝4096(假设每个存储块的容量为4KB)，则n＜＝86。因此采用这种存储方式，每个元素至少可以包含80个子元素，这个数量在一般的 XML 文档中是不多见的。当子元素的数量超过我们所提供的界值的时候，我们可以采用益处块的策略，当采用溢出块后，其总容量在现存的 XML 文档中，就不可能越界了。

当采用这种存储方式的时候，作为整个 XML 文档的元素的元素首部信息可以被常驻内存，因为每一个存储块就可以索引一个完整的 XML 文档，在数据库系统中，有4MB 的信息块常驻内存识别允许的，因此可以有4000多个文档的元素首部可以常驻内存，可节省大量的 I/O 操作。若采用修改后的数据头，则38n＋30＋1＋8＋8＋8＜＝4096，则n＜＝100，可以存储100个子元素。例如对下述的 XML 文档：

```
〈library〉
    〈book title＝"DataBase Processing" borrow＝"NO"〉
        〈author〉David M. Kroenke〈/author〉
        〈brief〉……http：//prenhall. com…〈/brief〉
    〈/book〉
    〈book title＝" ASP. NET Website Programming Program" borrow＝"NO"〉
        〈author〉Marco Bellinaso〈/author〉
        〈author〉Kevin Hoffman〈/author〉
        〈brief〉……http：//www. p2p. wrox. com…〈/brief〉
    〈/book〉
    〈book title＝"Developing XML Solutions" borrow＝"NO"〉
        〈author〉Jake Sturm〈/author〉
        〈brief〉……http：//www. mspress. microsoft. com…
        〈/brief〉
    〈/book〉
    ……
〈/ library〉
```

这个 XML 文档对应的逻辑结构如下图。



2.2 适用于 XML 文档封锁

在对 XML 元素进行封锁的时候，我们实际上是对当前标签的封锁，因此，涉及到对其子标签的封锁机制。因此，在不同的需求下，我们提供两种封锁方式：全封锁以及局部封锁。

(1)全封锁：在对标签进行封锁的同时，加锁管理器同时对其全部子元素的标签进行加锁，而且封锁对应的字符串数据。这种封锁方式适应传统的数据访问。

(2)局部封锁：在封锁的时候，只对当前标签进行封锁，不对其子元素进行封锁。这种封锁方式适应于用户要求修改当前标签的属性的需求，这样可以减少封锁的系统耗费。

2.3 对关系数据库封锁方式的引用

·对关系数据库中三种封锁的引入

由于关系数据库的封锁机制，已经很完备了，因此，我们可以直接把这三种封锁方式，与上面论述的两种封锁方式相结合，就可以产生比较适应 XML 存储的封锁机制。

但是，当我们引入全局封锁和局部封锁的策略后，对元素的封锁就变得复杂了。原因：

Ⅰ、当我们采用全局封锁的时候，一旦对一个元素及其子元素和字符数据进行加锁，这个元素相应的祖先元素间都会受到访问权限的限制。例如，一旦我们对一个元素进行了全局封锁的读封锁，则其所有的祖先元素都不能进行全局封锁的写封锁。另一个例子，一旦我们对一个元素进行了全局封锁的写封锁，则其所有的祖先元素都不能进行其他形式的封锁。基于以上原因，我们在引入全局封锁的同时，在进行封锁的元素的祖先元素上引入意向锁的机制(意向锁在关系数据空中也有应用，我们只是把其策略引入 XML 数据库的封锁策略)。

Ⅱ、当我们对一个元素进行全局封锁以后，则对其所有的子元素都不能再进行任何模式的局部封锁。但是对已经加了意向锁的元素，可以进行局部封锁。对局部封锁，无需引入意向锁的概念。因为局部封锁是不影响其祖先及子孙元素的，所以意向锁对局部封锁不起作用。局部封锁的各种封锁与传统数据库的封锁是相同的。

2.4 对包含超链接的数据块和属性的封锁

·由于在 SDML 中，我们为每个元素的包含超链接的字符数据存储块，都设置了一个附加块，因此，在封锁的时候，对不同的封锁方式，只需要设置数据块的附加块的块头中的封锁模式字段，就可以完成封锁数据信息中的超链接的工作。

·封锁属性的工作类同于对一般数据块的封锁，只需设置块头的封锁模式字段就可以了。

3 对比分析

3.1 SDML 存储的分析

SDML 是由我们开发实现的一种可以以层次式的方式存储 XML 文档的存储策略，这种策略可以适应多种需要。如果应用于实际，更是可以节省很多的 I/O 操作以及查询的耗费。

·I/O 耗费

传统的顺序式的存储，对于 XML 文档的检索是非常不利的。XML 文档本身是有层次结构的。因此以顺序方式存储，对查询来说，会造成不必要的 I/O 操作，这些操作对 CPU 来说，耗费是巨大的。例如上述例子的分析，如果没有其他辅助文档(DTD 或者 Schema)，为了检索一段字符数据，我们就需要遍历整个文档。如果目标在文档的末端的话，那么整个文档就需要被调进内存，这个操作量对查询来说耗费是巨大的。就算是有辅助文档的帮助，就只是在数据库中搜索我们想要得文档这一个操作，就很令人头疼。在我们的 SDML 中，每个作为文档的元素的标签头都可以被常驻内存，省去了很多 I/O 操作，这样对包含大量文档的数据库来说，会节省很多时间。并且在进行文档内检索的时候，有些我们不关心的分支子树也可以不掉入内存，节省了大量的 I/O 操作。

3.2 封锁粒度分析

·传统的 XML 数据库，由于没有层次式的存储方式，文档都是以顺序文件的形式存储的。因此，对文档的封锁，只能在整个文档的粒度上进行封锁。对于一些要求并发操作的 XML 文档，可能同时会有几个事务对同一个文档的不同元素的属性进行修改。在这种需求下，对整个文档的封锁，显然是不合乎应用的。在我们的方法中提出的局部封锁的策略，就可以很好地解决这个需求。就算在一般的需求下，当用户要求修改属性的时候，传统的 XML 数据库也只能对整个文档进行封锁，而在层次式的存储中，这就意味着要对整棵文档树进行封锁，这样就会牵扯到很多的加锁、封锁以及访问权限的问题。但是在提出了局部封锁的概念后，用户修改元素属性的时候，只需要关心当前标签的改动，无需考虑对其他祖先及子孙的影响，省去了很多的加锁操作。例如，在对上面的例子中的 book 元素的属性 borrow 进行修改的时候，若要对整棵树进行封锁的话，只对标签头进行封锁，就需要至少4次封锁，其中还不包括对兄弟节点影响所产生的封锁。而在局部封锁中，只需要对 book 这个标签头进行封锁，进一步封锁它的属性块，然后进行修改就可以了。这样就省去了大部分的封锁耗费，对于结构复杂的 XML 文档，这种封锁的优越性就更明显了。

总结 在基于 SDML 的存储策略的基础上，我们提出的适应于 XML 文档封锁的策略，可以完全适应于要求高度同步性的系统，并且可以根据不同的需求选择适应应用的封锁粒度，可以大幅度地提高系统的效率。采用这种封锁策略，可以无形中减少很多不必要的 I/O 操作，对建立索引更是有很大的帮助。

该策略对 SDML 的改进还不是很完善。例如，在这篇论文中，我们没有涉及对改进的 SDML 的存储块的插入、删除和更新操作，在以后的工作中，可以在这方面继续研究。

参 考 文 献

1 Garcia-Molina H, Ullman J D, Widom J. Database System Implementation
2 Mohan C. An Efficient Method for Performing Record Deletions and Updates Using Index Scansk
3 Mark Graves. Designing XML Databases
4 萨师煊，王珊. 数据库系统概论. 北京：高等教育出版社

(上接第86页)

监控器的另一个重要功能是启动/关闭 CSP，这是监控器的设计难点之一。我们的解决方法是：让 CSP 作为监控器的一个线程，启动 CSP 就生成一个新的 CSP 线程；关闭 CSP 也就是结束这个线程。由于监控器位于 Web 服务器中，事实上，CSP 将作为 Tomcat 的线程运行。

结束语 随着网络计算日益广泛的使用，人们迫切需要通过可视化的监控器直观、实时地监视网络中资源的动态变化和任务的执行情况，并控制和管理任务的执行。对计算服务提供节点的监视控制是其中重要的一类监控器。本文以兰州大学数学网络计算环境(Mathematics Internet Computing Environment)中 CSP(Computing Service Provider)监控器为例，详细阐述了计算服务提供节点监控器的结构、功能、设计和实现。

参 考 文 献

1 程显华等译，Mark Wutka 著. JSP 和 Servlet 程序设计使用专辑. 机械工业出版社，2002
2 詹建. 面向规范的数学网络计算环境：[硕士论文]. 2003
3 晏子译. MySQL 中文参考手册. http://www.pcsoftware.com.cn
4 李昭智等译，Ivor Horrton 等著. Java 2 编程指南. 电子工业出版社，2003
5 Placek M, Buyya R. G-Monitor: Gridbus Web portal for monitoring and steering application execution on global grids. In: Proc. of the Intl. Workshop on Challenges of Large Applications in Distributed Environments (CLADE 2002). Conjunction with the 12th International Symposium on High Performance Distributed Computing (HPDC 2003), Seattle, USA, June 2003
6 Czajkowski K, Fitzgerald S, Foster I, Kesselman C. Grid Information Services for Distributed Resource Sharing. In: Proc. 10th IEEE Intl. Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, 2001
7 MDS3. http://www.globus.org/mds/
98 DataGrid Information and Monitoring Services Architecture: Design, Requirements and Evaluation Criteria: [Technical Report]. DataGrid, 2002
10 Hawkeye. http://www.cs.wisc.edu/condor/hawkeye