基于相对位置的分布式实时协同编辑乐观锁并发控制算法*)

詹永照　毛启容　宋顺林

(江苏大学计算机与通信学院　镇江212013)

The Concurrency Controlling Algorithm of Optimistic Locking Based on Relative Position in Real-Time Distributed Cooperative Editing System

ZHAN Yong-Zhao　MAO Qi-Rong　SONG Shun-Lin

(School of Computer & Communication Engineering, Jiangsu University, Zhenjiang 212013)

Abstract　Traditional locking mechanism is fit for the concurrency controlling and data consistency maintenance under centralized architecture, but its response is slow, so it can not be used as concurrency controlling strategy for real-time distributed cooperative editing system. Moreover, operation transformation and data consistency maintenance technology are able to assure of quick response and unconstraint, but can't solve the context-specific inconsistency problems. Enlightened by the idea of operation transformation and optimistic locking mechanism, we bring forward the concurrency control algorithm of optimistic locking based on relative position. In this algorithm, the start position of locking region and the position of operation are relative, and they are not transformed into absolute position until operations are sent to cooperative sites. Furthermore, any coeditor can edit in advance before his/her locking is confirmed. If his/her locking is successful, the previous operations go into effect, or else undo these previous operations. We have analyzed the actual applications and can find that the possibility of undoing the previous editing operations because of locking conflict is very little. So this concurrency controlling algorithm has virtues of quick response, nice data consistency maintenance etc.

Keywords　Real-time distributed cooperative editing, Relative position, Optimistic locking mechanism, Concurrency controlling, Consistency

1　引言

实时分布式协同编辑系统是计算机支持的协同工作的典型应用系统，不少学者对其进行了大量的研究，其难点是协作编辑的一致性、实时性和无约束性。因此，并发控制始终是它的研究热点，目前已提出的协同编辑的并发控制算法有：传统的加锁法、tickle 锁[1]、floor 控制[2]、可逆执行[3,4](Undo/Redo)和操作转换[5~7]等。传统的加锁法最突出的优点是设计和实现较为简单，能保证具体上下文的语义完整性，但是在分布式协同编辑中无法保证加锁位置的一致性。tickle 锁方法适用于分节加锁的实时分布式协同编辑的并发控制，然而无法提供多个编辑者对节内的同时编辑[1]。JCE 中的协同编辑采用了 floor 控制方法，每个协作编者只有获得 floor 才能进行编辑操作，较适合讨论式的协作编辑，不适合实时无约束的协作编辑。在可逆操作(Undo/Redo)算法中，操作可以立即执行，且与操作有关的信息将被保留下来，以便在必要的时候取消该操作，这种方法的用户响应性能良好，但这种方法需要定义操作的全程时序，这是相当困难的。操作转换算法能自动地解决冲突而无需手工干预，有良好的实时响应性，能保持收敛性和维护意愿，但是不能保证上下文的语义完整性。受传统的乐观锁机制和操作转换方法的启发，本文提出了基于相对位置的分布式实时协同编辑乐观锁并发控制算法，该算法对加锁位置和操作位置均用相应的相对位置表示，当操作发送到各协作节点时再把它转换成应有的位置。编者在加锁请求获得确认前，可预先进行编辑操作，直至得到确认或否认，确认则预先编辑操作生效，否认则取消预先进行的编辑操作。经实际应用分析表明：该算法具有乐观锁机制和操作转换方法的各自优点。

2　系统要素定义及基于相对位置的乐观锁并发控制思想

2.1　系统要素定义

我们的协同编辑系统是混合式结构，当有用户建立起某个主题的协同编辑时，其他用户可随时加入该主题的协同编辑中。各编辑端的系统采用完全分布式结构，每个编辑端获得共享文档的一个副本，即可实时编辑或阅读。协作成员分为主编、协编和读者3种角色。

系统模型所涉及的术语定义如下：

定义1　主编即为某个主题的协同编辑的创建者，是整个协同编辑过程的协调者。

定义2　协编是某个主题的协同编辑的参与者。

定义3　读者是某个主题的协同编辑的旁观者，不参加编辑工作，只阅读编辑信息。

每位参与编辑的用户，必须持有一把锁。加锁后，即在锁

*)本文得到国家自然科学基金项目(60273040)、教育部科学技术研究重点项目和江苏省高校自然科学基金项目(02KJB520003)的资助。詹永照　教授，博士，主要研究方向为分布式计算、人机交互。毛启容　硕士研究生，主要研究方向为 CSCW。宋顺林　教授，主要研究方向为分布式计算、软件工程。

定范围实施编辑操作。

定义4 锁 lock 为五元组表示：lock(id，s_p，o_l，n_l，t_s)，其中 id 为用户标志，s_p 为开始位置，o_l 为加锁时的锁定范围长度，n_l 为编辑中的锁定范围长度，t_s 为加锁时的时间戳。

定义5 锁表为各编辑者的锁信息线性表：locklist ($lock_0$，$lock_1$，$lock_2$，…，$lock_n$)，n 为编者数，$lock_0$为锁头，锁表按锁位置的升序排序。

定义6 锁头 $lock_0$为记录第一把锁的时间戳，作为以后申请成为第一把锁的时间参考。

定义7 参考锁为申请锁时锁定位置的前一把锁，它是作为锁定位置转换的参考点。

定义8 取消操作集是保留锁申请而未得到确认的过程中本地预先进行的各编辑操作。

定义9 相对位置是指从参考锁的结束位置到欲加锁的起始位置的距离。

定义10 锁的冲突是指存在两把锁的锁定范围出现交叉、重叠或者包含情形。

2.2 基于相对位置的乐观锁并发控制思想

每个协作者欲编辑必须先申请锁定范围。锁定范围由主编节点来仲裁，获得主编节点的认可，即可在锁定范围内任意进行各种编辑操作，并将以锁内偏移位置表示的操作发给其它节点，其它节点依次实施其操作。当编者申请加锁时，首先扫描本地的锁表，看有没有冲突。如果有，则申请失败，否则向主编节点发送加锁请求。锁的申请信息里包括参考锁的时间戳。在申请锁的过程中，本地编辑操作可预先进行，并保留这些操作作为取消操作集。主编节点基于申请锁的相对位置判断是否有别的节点与该节点申请的锁位置范围冲突，或者该锁的参考锁的时间戳是否与锁表里面的一致，不一致即参考锁过时。参考锁过时意味着申请加锁者在申请过程中，原来持参考锁的编者已释放该锁，会造成加锁时无参考锁参考来转换加锁位置或转换的位置与申请者的原意不符。如果没有冲突并且参考锁没过时，它将确认其加锁，并附上当时的时间戳，发回认可消息，同时将盖上时间戳的加锁信息发给其它节点。如果有冲突或参考锁过时，主编节点将发回否定加锁信息。当申请锁的节点收到认可消息时，原来的编辑操作生效，否则按取消操作集取消本地的预先编辑操作。申请加锁过程如图1所示。

每个用户节点释放锁时，也将释放锁的信息发给主编节点。主编节点释放该锁，并将锁释放信息发给组内所有节点，每个节点收到该信息后释放该锁。

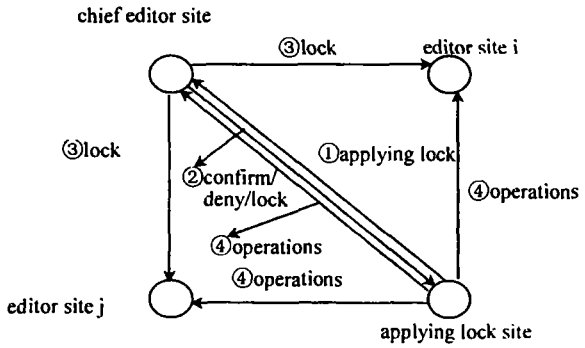

图1 加锁申请与操作

由上述加锁请求过程可看出，如果加锁的位置范围没有与本地的锁表冲突，那么本地的编辑操作仍继续进行，不必等待远程主编节点的应答到来再操作。此外，在协同编辑中，我们还可以引入直观的感知和协调技术。因此，多个协作者同时竞争一个位置，以及锁申请时刻正好参考锁释放的可能性很小，所以发生锁申请不成功而取消操作的可能性也很小。这就提高了整个系统编辑的效率。

3 加锁的处理算法

申请锁节点将含有用户 id、相对位置、长度及参考锁的用户 id、时间戳信息的请求加锁信息 apply_lock (user_id，relactive_pos，lock_len)，ref_lock (ref_user_id，timestamp0)发给主编节点。主编节点在接收到加锁请求消息后，若参考锁未过时，还有可能在其参考锁和加锁的位置之间，已有其它编者加了锁，则必须确定应加锁的位置、做冲突判断及重新调整参考锁。设没有冲突，由于申请节点在申请锁时尚未看到参考锁和加锁的位置之间有人加锁，故其加锁位置可参照其它编者加了锁的位置和加锁时的长度来计算。如图2所示，加锁的长度为 l，参考锁为 $lock_i$，其相对位置为 r，$lock_j$ 和 $lock_{j+1}$间相对位置为 s_j，$lock_j$ 加锁时的长度为 w_j，则必存在 $lock_k$，使得：

$$r \geqslant \sum_{j=i+1}^{k-1}(s_j+w_j)，且\ r+1 \leqslant (\sum_{j=i+1}^{k-1}(s_j+w_j)+s_k)$$

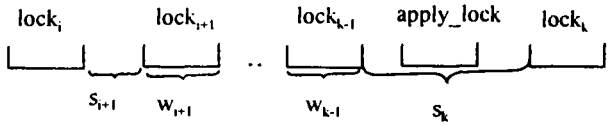

图2 加锁的位置确定

故它应按如下算法执行：

```
按 ref_user_id 查看锁表 locklist，若找到，设找到的锁为 lock_i(id，s_p，o_l，n_l，t_s)；
if (没找到或 lock_i. t_s 大于 timestamp0) 给予申请者否定加锁的回答；
else
  {
   rel_pos00=0;
   rel_pos0=lock_i+1. s_p-lock_i. s_p- lock_i. n_l + lock_i+1. o_l;
  while( rel_pos0≤relactive_pos)
    {
     i=i+1;
     rel_pos00= rel_pos0;
     rel_pos0 = rel_pos0 + lock_i+1 . s_p-lock_i. s_p- lock_i. n_l +
     lock_i+1. o_l;
     }
  if (relactive_pos+ lock_len≤rel_pos0- lock_i+1. o_l )
    { 取时间戳 timestamp;
      给申请锁端发确认加锁消息并附上 timestamp;
      relactive_pos= relactive_pos- rel_pos00;
      在 lock_i 后增一新锁 lock(user_id ，lock_i. s_p + lock_i. n_l +
      relactive_pos，lock_len，lock_len，timestamp);
      lock_i 作为参考锁，将加锁信息 add_lock(user_id，relactive_
      pos，lock_len，timestamp)，ref_lock(lock_i. u)发给其它所有节
      点；}
  else 发否认消息给申请端 ；
  }
  当加锁消息到达其它各端时，这些端按如下算法执行：
  据参考锁的用户 id，查锁表，得 lock_i(id，s_p，o_l，n_l，t_s)；
  if (本端已申请锁未得到确认)
    { 设申请锁为 lock_j(id，s_p，o_l，n_l，t_s)；
      if (lock_j 介于 lock_i 和 add_lock 之间且无冲突)
          加锁位置 s_p = lock_i. s_p + lock_i. n_l + add_
          lock. relactive_pos+lock_j. n_l-lock_j. o_l;
      else {
            if (lock_j 与 add_lock. 冲突) 取消本端的申请锁；
            加锁位置 s_p = lock_i. s_p + lock_i. n_l + add_
            lock. relactive_pos;
      }
   else 加锁位置 s_p = lock_i. s_p + lock_i. n_l + add_
lock. relactive_pos;
  增一新锁 lock(add_lock. user_id ，s_p，add_lock. lock_len，add_
```

lock.lock_len, add_lock.timestamp);

通过加锁过程可以看出:若无加锁范围没有冲突并且参考锁没过时,其他编者进行删除和插入操作时,各编者锁外的信息并没有改变,因此没有造成锁之间的相对位置的变化;若主编节点发现在申请过程中有其他编者在参考锁和欲加锁的范围之间加了锁,主编节点将根据他们共同的一致的信息,进行加锁位置的转换,并重新调整参考锁,不会引起加锁的位置不一致;当加锁消息到达各编辑端时,若有编辑端已在参考锁和加锁的范围之间申请了加锁,并开始编辑,也会按照相应的状况进行加锁位置的转换,保证加锁位置的一致性;若参考锁和加锁的范围之间无其他编者加锁或申请锁,其参考锁和加锁的范围之间显然不变;若有加锁冲突或参考锁过时,系统会自动取消其加锁申请操作,恢复其到加锁申请前状况。故我们的加锁处理方案能保证锁定位置的一致性。

4 编辑操作的位置转换和信息的一致性维护

下面用图3中所示的实时协同编辑对话的情景,来说明乐观锁机制的工作过程。在这个图中有3个协作节点和3个编辑操作:操作O1、O2、O3分别产生于节点1、2、3。假设在所有节点上的初始文档包括以下文本:ABCDEFGHIJK,存在两个基本的编辑操作:(1)Insert(S,P),它表示在位置P插入字符S。(2)Delete(N,P),它表示从位置P删除N个字符。还有一个锁定操作:Lock(P,N),它表示从位置P开始锁定N个字符。假设节点0、1、2分别先执行锁定操作:Lock(0,3),Lock(3,4),Lock(7,4)。3个操作分别为:O1—insert('1',1);O2—insert('3',4);O3— delete(1,7)。各操作位置的转换只需考虑该编辑操作以后的各个锁定位置的转换。插入操作时,应根据插入文本的长度将相应锁位置往前平移;删除操作时,也应根据删除文本的长度将相应锁位置往后平移。而各锁内的编辑操作位置就用锁内的相对位置来表示,其绝对位置为其锁定起始位置与锁内相对位置相结合,因此其绝对位置也得到了相应的正确转换。这样,图3的操作变为:O_1: Insert('1',1);O_2:Insert('3',1);O_3:Delete(1,0)。各节点的执行情况如下:

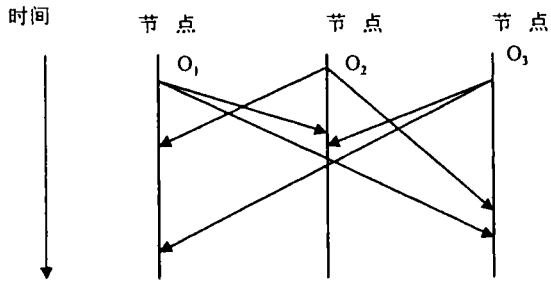

图3 实时协同编辑会话的情景

在节点0,执行了O_1后,因为O_1操作在节点0的锁内插入了一个字符'1',锁表变为:Lock(0,4),Lock(4,4),Lock(8,4),文本状态变为:A1BCDEFGHIJK;当O_2到来时,它在自己的锁内相对位置为1的地方(即字符'D'之后)插入字符'3',这与操作O_2的原来意愿是一致的。当O_2执行完后,锁表变为:Lock(0,4),Lock(4,5),Lock(9,4),文本状态变为:A1BCD3EFGHIJK;当O_3到来时,它在自己的锁内相对位置为0的地方删掉一个字符'H',这与操作O_3原来的意愿也是一致的。在节点0,最后的文本状态为:A1BCD3EFGIJK。

在节点1,首先执行操作O_2,在它自己的锁内相对位置为1的地方插入字符'3'。因为节点0的锁在节点1前面,所以Lock[0,3]不受操作O_2的影响,节点1和节点2的锁定位置要做相应的调整,它们分别变为:Lock[3,5],Lock[8,4],文本状态变为:ABCD3EFGHIJK;当操作O_1:Insert('1',1)到来时,它在与Lock[0,3]相对位置为1的地方插入字符'1',执行操作O_1以后,锁表变为:Lock[0,4],Lock[4,4],Lock[8,4],文本状态变为:A1BCD3EFGHIJK;当操作O_3到来时,它在与Lock[8,4]相对位置为0的地方删除一个字符'H',这与O_3的意愿是一致的。在节点0,最后的文本状态为:A1BCD3EFGIJK。

在节点2,首先执行操作O_3,在它自己的锁内相对位置为0的地方删除字符'H'。因为节点0和节点1的锁在节点2前面,所以Lock[0,3],Lock[7,4]不受操作O_3的影响,只是改变节点2的锁定范围:Lock[7,3],文本状态变为:ABCDEFGIJK;当操作O_2:Insert('3',1)到来时,它在与Lock[3,4]相对位置为1的地方插入字符'3',执行操作O_2以后,锁表变为:Lock[0,4],Lock[4,4],Lock[8,3],文本状态变为:ABCD3EFGIJK;当操作O_1到来时,它在与Lock[0,3]相对位置为1的地方插入一个字符'1',这与O_1的意愿是一致的。在节点0,最后的文本状态为:A1BCD3EFGIJK。

由上面的分析可知:3个节点得到的文本状态是相同的,所以基于相对位置的乐观锁机制能够保证各个节点上得到的文本状态是一致的,从而保证实时协同编辑的共享信息的一致性。

5 算法实现

我们的实时分布式协同编辑系统由一个集中的会话管理器和若干协作节点所组成。会话管理者可以运行在单独的一台服务器上,也可以位于其中的一个协作节点上,其IP地址是公开的,因而成为整个系统的接入点。它主要管理协作者、会话以及会话的日志,并不涉及到执行或发送编辑操作,只与一个用户加入或离开会话时有关。通过会话管理器该系统同时可进行多个实时协同编辑工作。协作节点主要负责与会话管理者的接洽,与其他协作节点交换信息以及编辑信息的界面显示工作,各协作节点是采用完全分布式的结构。协作者的角色分为主编、协编和读者。整个系统采用Java语言实现,其并发控制即采用本文提出的算法实现。

算法中的锁申请信息的结构为:

```
class Tlock_relative{
    String user; //申请该锁的用户名
    String lockformer; //该锁的参考锁所属的用户的用户名
    String locklatter; //该锁的后一个锁所属的用户的用户名
    long timestamp; //该锁申请时参考锁的时间戳
    int distancerela; //锁定起始位置离参考锁的锁定结束位置距离
    int lklength; //初始锁定的长度
}
```

每个协作节点都有一个锁表的副本,它随着各个节点锁定位置的变化和编辑操作的到来而随时更新,在锁表里面,每个节点的锁定位置是使用绝对位置来表示的,其数据结构如下:

```
class Tlock_table {
    String user; // 锁定该位置的用户的用户名
    long timestamp; // 该锁的申请成功时的时间戳
    int lkstart; // 锁的起始位置
    int lkinitlength; // 记录锁锁定后编辑前的锁定长度
    int lklength; // 锁定文本的长度(随用户的操作而变化)
    int curpos; // 用户当前的编辑操作在远程的感知位置
}
```

6 相关工作比较

CES采用了tickle锁方法,只适合分节的实时分布式协

同编辑，无法提供多个编辑者对节内的文本同时编辑[1]。我们的算法除了具有 tickle 锁的功能外，还具有更小范围的实时分布式协同编辑的并发控制与一致性维护能力。

JCE 中的协同编辑采用了 floor 控制方法，每个协作编者只有获得 floor 才能进行编辑操作，较适合讨论式的协作编辑，不适合实时无约束的协作编辑。而我们的算法具有实时性和无约束性。

操作转换算法具有良好的实时响应性、收敛性和无约束性，但是不能保证上下文的语义完整性，该算法的提出者和实现者在后来的实现中也集成了锁机制的并发控制算法[8]。我们的算法在实时响应、收敛和无约束等方面的性能与它相似，此外还能保证具体上下文的语义完整性。

结论 并发控制问题一直是分布式系统研究的重点和难点。本文提出的基于相对位置的乐观锁机制的并发控制策略是结合操作转换的思想，对传统的乐观锁机制及 tickle 锁方法进行了改进。我们已经将该并发控制算法和一致性维护技术成功地运用到了实时分布式协同编辑系统开发中。经原型系统实际运行和多个协作编者操作观察表明：该算法实时响应性和并发性好。该算法经进一步改进和扩充，还可应用到如线性表的分布式共享对象操作的并发控制中。

参 考 文 献

1 Greif I, Seliger R, Weihl W. A case Study of CES: A Distributed Collaborative Editing System Implemented in Argus. IEEE Transaction on software Engineering, 1992, 18(9): 827～839

2 Abde-Wahab H, Kvande B, Kim O, Pavreau J P. An Internet Collaborative Environment for Shared Java Applications. In: proc of 5th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems (FTDCS'97), Los Alamitos: IEEE Computer Society Press, 1997. 112～117

3 Choudhary R, Dewan P. A General Multi-User Undo/Redo Model. In: Marmolin H, Sundblad H and Schmidt K, eds. Proc. of European Conf. on Computer Supported Work, Dordrecht: Kluwer Academic Publishers, 1995. 231～246

4 Sun C Z. Undo Any Operation at Any Time in Group Editor. In: Proc. of 2000 ACM Conf. on Computer Supported Cooperative Work, New York: ACM Press, 191～200

5 Ellis C A, Gibbs S J, Rein G L. Groupware: some issues and experiences. Communications of the ACM, 1991, 34(1): 39～58

6 Sun C Z, Jia X, Zhang Y, Yang Y. A Generation Transformation Scheme for Consistency Maintenance in Real-time Cooperative Editing Systems. In: Proc. of Intl. ACM SIGGROUP Conf. on Supporting Group Work, New York: ACM Press, 1997. 425～434

7 Sun C Z, Ellis C A. Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements. In: Proc. of the ACM Conf. on CSCW, New York: ACM Press, 1998. 59～68

8 Sun C Z, Sosic R. Optional Locking Integrated with Operational Transformation in Distributed Real-time Group Editors. In: Proc. of ACM 18th Symposium on Principles of Distributed Computing, New York: ACM Press, 1999. 43～52

（上接第121页）

形式化的约束规范，这对进一步的机器演证和软件开发将有很重要的作用。

参 考 文 献

1 Sandhu R S, Coyne E J, Feinstein H L, Youman C E. Role-Based Access Control Models. IEEE Computer, 1996, 29(2): 38～47

2 Gavrila S I, Barkley J F. Formal specification for role based access control user/role and role/role relationship management. RBAC '98. In: Proc. of the third ACM workshop on Role-based access control, Fairfax, VA, 1998. 81～90

3 Yan Han, Zhang Hong, LiuFengyu. Modeling and Implementing Active Behavior Control Based on Roles. In: Proc. of Fourth Intl. Workshop on CSCW in Design, Compiegne, France, 1999

4 Booch G, Rumbaugh J, Jacobson I. The Unified Modeling Language User Guide. USA, Addison Wesley, 1999

5 Dewan D, Shen H. Controlling access in multiuser interface, ACM Transactions on Computer-Human Interaction, 1998, 5(1): 34～62

6 Shin M E, Ahn G-J. UML-based Representation of Role-based Access Control. In: Proc. of 5th IEEE Intl. Workshop on Enterprise Security (WETICE 2000), June NIST, MD, 2000

7 Moffett J D. Controb principles and role hierarchies, RBAC '98. In: Proc. of the third ACM workshop on Role-based access control, Fairfax, VA, 1998. 63～69

8 Ahn G-J, Sandhu R S. The RSL99 Language for Role-Based Separation of Duty Constraints. RBAC'99. In: Proc. of the Fourth ACM Workshop on Role-Based Access Control, Fairfax, VA, USA, 1999. 43～54

9 Swift M M, Brundrett P, Dyke C V. Improving the granularity of access control in Windows NT. In: Proc. of the Sixth ACM Symposium on Access control models and technologies. Chantilly, VA USA, 2001. 87～96

10 严悍，张宏，许满武. 基于角色访问控制对象式建模及实现. 计算机学报，2000，23(10)

11 陆钟万著. 面向计算机科学的数理逻辑. 北京：科学出版社，1998

12 丁铁凡，吉逸. 企业外部网的访问控制技术与系统的研究. 计算机集成制造系统，2001，7(1)：7～10