一种高效率的实时协同编辑中的意图保持操作转换算法

蔡维纬[1)] 何发智[1),2)] 吕 晓[1)]

[1)](武汉大学计算机学院 武汉 430072)

[2)](软件工程国家重点实验室 武汉 430072)

摘 要 作为一类高级分布式系统,实时协同编辑系统允许不同地点的用户同时编辑共享文档,具有高响应性和高并发性的特点.操作转换(Operational Transformation,OT)算法能够保留所有用户操作的效果并维护数据的一致性,是协同编辑系统首选的并发控制方法.为了提高远程操作的响应时间,文中提出了一种意图保持的 OT 算法(Merging Operations based Operational Transformation,MOOT).该算法基于这样一个事实,大多数情况下,协同编辑中插入操作的数量明显多于删除操作.因此,MOOT 构造了一种优化的操作历史结构,即删除操作排在插入操作前面,避免算法的计算时间依赖于大多数操作.更进一步,MOOT 在重构过程中移除无效操作,有效的压缩了操作历史的大小.为了验证算法的有效性,在不同插入比例情况下,将 MOOT 算法与当前性能最优的 ABT 算法进行了对比实验.实验结果表明,MOOT 算法具有更高的计算效率,在合理的比例情况下,其计算时间大约是 ABT 算法的计算时间的十分之一.

关键词 实时协同编辑;并发控制;操作转换;社交网络

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2015.02041

An Efficient Preserving Intention Operational Transformation for Real-Time Collaborative Editing

CAI Wei-Wei[1)] HE Fa-Zhi[1),2)] LV Xiao[1)]

[1)](*School of Computer Science, Wuhan University, Wuhan* 430072)

[2)](*State Key Laboratory of Software Engineering, Wuhan* 430072)

Abstract As a kind of advanced distributed systems, real-time collaborative editing systems allow geographically dispersed users to manipulate the shared document simultaneously with the characteristic of high concurrency and responsiveness. Operational Transformation (OT) algorithm, which is able to preserve the operation effects of all collaborators and maintain the data consistency, is the concurrency control method of first choice for collaborative editing systems. To improve the response time of remote operations, this paper proposes an OT algorithm with preserving intention (Merging Operations based Operational Transformation, shorted as MOOT). The idea comes from the fact that the number of insertions is obviously more than that of deletions in collaborative editing environment. The MOOT algorithm constructed an optimized operation history with deletions before insertions so that the computing time did not depend on the majority of operations. Moreover, the algorithm removed useless operations, which effectively compressed the size of the operation history. To confirm the algorithm's effectiveness, this paper conducted experiments comparing the performance of MOOT with the state of the art algorithm (ABT) in

收稿日期:2014-06-16;最终修改稿收到日期:2015-03-15.本课题得到国家自然科学基金(61472289)资助.**蔡维纬**,男,1988 年生,博士研究生,主要研究方向为 CSCW、协同 CAD. E-mail: caiwei0709@gmail.com. **何发智**(通信作者),男,1968 年生,博士,教授,中国计算机学会(CCF)会员,主要研究领域为 CAD 图形图像、协同计算. E-mail: fzhe@whu.edu.cn. **吕 晓**,女,1983 年生,博士研究生,主要研究方向为 CSCW、协同 CAD.

different ratios of insertions. As shown by the experiments, MOOT algorithm is more efficient than ABT algorithm. In a reasonable percentage (80%), the proposed method computes approximately 10 times faster than ABT algorithm.

Keywords real-time collaborative editing; concurrency control; operational transformation; social networks

1 引 言

2013 年图灵奖得主 Leslie Lamport① 的主要成就来源于 1978 年发表的一篇关于分布式系统的论文[1]. 该篇论文是计算机科学史上引用率最高的论文之一(one of the most cited in the history of computer science). 作为分布式系统的进一步发展,实时协同工作系统在继承了 Lamport 事件关系的基础上,所演化出的操作转换(Operational Transformation,OT)算法吸引了相关学者的大量的研究. OT 作为一种乐观并发控制算法,被广泛用于支持纯文本②、Word 和 PowerPoint[2-3]、Spreadsheet[4]、2D 图像[5]、2D 和 3D 图形[6-7]以及 CAD[8-10]的协同编辑工作.

随着移动计算、云计算、大数据的发展,OT 算法的性能逐渐成为关注的焦点. Shao 等人[11]指出实现意图保持且经过严格证明的 ABT 算法(Admissibility-Based Transformation)在计算性能上优于其他的 OT 算法. 基于 ABT 算法框架[12]、支持 String 操作的 ABST[13],异步协同的 ABTS[11]以及支持 anyundo 的 ABTU[14]等算法相继被提出. ABT 算法维护特殊的操作历史 HB(History Buffer),即插入操作在删除操作之前,避免了删除操作对插入操作的影响,很自然地保持了操作对象之间的位置关系. 与已有的 OT 算法相比,ABT 算法不仅能够实现意图保持而且具有更高的计算效率. 但是,ABT 算法的计算时间严重依赖于插入操作的数量.

相关文献表明[15],在协同编辑中插入操作的数量远远多于删除操作,插入操作占总操作数 80% 是一个比较合理(reasonable)的比例. 因此,ABT 算法重构的 HB 结构并非是最合理的. 另一方面,协同编辑过程中存在着许多无效操作对(插入的对象随后又被删除),将它们从 HB 中移除不仅不会影响其他操作,而且能够减少不必要的计算. 基于以上两点考虑,本文提出了一种合并无效操作的 MOOT 算法(Merging Operations Based on Operational Transformation).

本文第 2 节为 OT 相关的研究工作;第 3 节为 OT 的基本模型与算法分析;第 4 节为 MOOT 算法的详细描述;第 5 节为 MOOT 算法的实例分析;第 6 节为实验研究;第 7 节为全文总结.

2 相关工作

Ellis 等人[16]在 1989 年首次提出操作转换的概念,建立了 OT 算法的基本模型. 第一个 OT 算法(dOPT)被成功应用到了协同编辑系统 Grove. 基于 dOPT 算法,杨光信等人[17]提出了面向对象数据模型的并发控制方法.

Ressel 等人[18]发现了 dOPT 的"puzzle"(某些场景下各个站点的结果不一致)并提出转换函数正确性的充分必要条件(TP1 和 TP2). 之后,不断有新的算法被提出,尽管声称能够满足 TP1 和 TP2,但随后都被其他算法证明存在"puzzle". 设计满足 TP2 条件的转换函数是非常困难的. 2014 年 Randolph 等人[19]指出,非全序 OT 算法的转换函数都不能同时满足 TP1 和 TP2 条件. 经过自动机合成的正确转换函数需要为操作添加额外的信息.

大部分 OT 算法通过控制过程(也称作集成过程)摆脱了 TP2 对转换函数的限制. 这类算法不需要转换函数在所有情况下都能获得正确的结果. 控制过程负责调度远程操作与操作历史中的操作进行转换,根据操作的全局顺序来产生唯一的转换路径. SOCT4[20]利用集中式的 sequencer 产生全局有序的操作序列,GOTO[21]结合状态向量和站点优先级来判断操作之间的全序关系,TIBOT[22]定义的全序要求各个站点的逻辑时钟是同步的. 每一个操作在所有站点进行转换的操作序列都是相同的,最终得到的操作历史也是相同的.

控制过程还有另外一个作用,就是处理偏并发操作的集成. 在协同编辑环境下,并发操作有可能

① http://amturing.acm.org/award_winners/lamport_1205376.cfm

② http://www.codingmonkeys.de/subethaedit

产生于不同的文档状态，导致操作对象的位置不具有可比性[23]. 因此，直接调用转换函数可能会导致结果不一致的情况发生. GOT[24]算法补充了基本操作转换的概念，提出了包含转换 IT(Inclusive Transformation)的逆过程排除转换 ET(Exclusive Transformation). ET 能够将不同文档状态下的操作转换成相同状态下的操作. 交换两个操作的执行顺序是先调用 ET 然后调用 IT，这个过程被封装为 $SWAP$ 函数. Suleiman 等人[25]提出了前向转换 FT(Forward Transposition)和后向转换 BT(Backward Transposition)的概念. 通过文献[26]的分析，FT 和 BT 本质上分别等价于 IT 和 $SWAP$. SOCT2[25]的控制过程接收到远程操作后，将站点的操作历史分类排列，因操作在前，并发操作在后，并发操作序列是需要进行转换的. 该排列被大多数 OT 算法用来寻找并发操作集. Sun 等人认为转换函数和控制过程是松散耦合的. 转换函数定义操作之间的互相转换，与具体的应用相关，而控制过程负责调度远程操作与 HB 中的操作进行转换，是独立于具体应用的. 因此，Sun 等人[3,27]将控制过程封装成通用协同引擎 GCE(Generic Collaborative Engine)，并应用到单用户程序向多用户的透明转换.

意图保持是一种更加严格的一致性. Sun 等人[28]最早提出 OT 算法不仅要实现结果一致性，而且要保证最终的结果反应了操作的意图，即意图保持. 然而意图的定义一直是模糊的. Li 等人[29-30]认为意图保持包含了结果一致，并明确的将操作的意图定义成操作作用对象之间的位置关系. 全序的 OT 算法在静默状态(协同会话结束)的结果依赖于操作的全序关系，不一定能够保持操作的意图. TTF[31]利用分布式系统中的墓碑(Tombstone)概念，保留被删除的字符，使得对插入操作的转换具有位置单调递增的性质，能够保持操作对象之间的前后关系. TTF 需要在站点同时维护物理视图和逻辑视图，物理视图保存了逻辑视图中被删除的对象，用户与逻辑视图进行交互，而待转换的操作都是基于物理视图的. 这种方法的空间消耗比较大，并且视图之间的转换也需要耗费大量计算时间. LBT[32]为协同编辑中的所有对象定义全局顺序，但是从偏序推演到全序的过程需要花费大量的计算时间和空间. ABT 在所有站点维护插入在前删除在后的 HB 结构，避免了删除操作对插入操作的影响，简化了意图保持的实现过程.

在实时协同编辑中，可选的乐观并发控制算法还包括CRDT(Commutative Replicated Data Type)[33-35]. CRDT 通过为协同编辑过程中出现的每一个对象定义全局有序并且唯一的标识，实现操作之间的可交换性，最终确保结果一致. 与 CRDT 相比，意图保持的 OT 实现了更强的一致性，结果更接近用户的实际意图. 目前，CRDT 方法还没有得到广泛的实际应用.

文献[36-37]提供了 ABT 算法与其他 OT 算法在集成远程操作时的性能比较. 实验结果表明，ABT 算法是当前效率最高的操作转换算法. 然而，ABT 重构的操作历史是否是最优的，值得商榷.

3　OT 基本模型与算法分析

OT 算法从协同编辑系统中抽象出两个元操作(插入、删除)，操作的对象是线性排列的. OT 算法能够推广到其他具有复杂结构的协同系统[38]. 操作之间的先后关系建立在 Lamport 定义的“happened-before”理论基础之上. 站点执行的操作都被保存在 HB 中. 操作转换的基本思想是本地操作立即执行，远程操作经过转换之后再执行，各个站点按照不同的顺序执行操作之后获得相同的结果.

定义 1.　因果关系. 给定任意两个分别位于站点 i 和站点 j 上的操作 O_a 和 O_b，称 O_a 和 O_b 存在因果关系(记作 $O_a \rightarrow O_b$)，当且仅当 O_a 和 O_b 满足下列 3 个条件之一：(1) $i=j$ 并且操作 O_a 发生在 O_b 之前；(2) $i \neq j$ 并且操作 O_a 在站点 j 的执行先于操作 O_b 的产生；(3) 存在操作 O_x，并且有 $O_a \rightarrow O_x$ 和 $O_x \rightarrow O_b$.

定义 2.　并发关系. 给定任意两个操作 O_a 和 O_b，称 O_a 和 O_b 存在并发关系(记作 $O_a \parallel O_b$)，当且仅当 O_a 和 O_b 既不满足 $O_a \rightarrow O_b$，也不满足 $O_b \rightarrow O_a$.

根据定义 1、2，图 1(a)中操作 o_1 与 o_2 是并发关系，图 1(c)中 o_1 与 o_2 是因果关系. 每个操作都会携带状态向量 $\boldsymbol{SV}$(State Vector)的信息，用来判断操作之间的关系. $\boldsymbol{SV}$ 是一个长度固定的向量，长度等于协同站点的数目，其中每个分量对应该站点已执行操作的数目. 如果操作 o_1 来自站点 i，o_2 来自站点 j，并且 $\boldsymbol{SV}_{o_1}[i] < \boldsymbol{SV}_{o_2}[i]$，则 $o_1 \rightarrow o_2$.

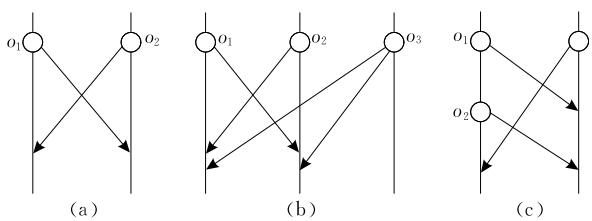


图 1　因果关系与并发关系

为了方便讨论,共享的文档通常被看作是字符的集合,字符串(起始位置为 0).协同编辑中的两个基本操作:$ins(p,c)$表示在位置 p 插入字符 c;$del(p)$表示删除位置 p 的字符.每个站点对应一位用户.图 2 解释了操作转换的基本思想.两个用户共同编辑文档(初始为"*abc*"),站点 1 发起操作 $o_1=ins(2,d)$,站点 2 同时发起操作 $o_2=del(1)$.本地产生的操作都会被立刻执行,然后广播给其他站点.其中,$\boldsymbol{SV}_{o_1}=[0\ \ 0]$、$\boldsymbol{SV}_{o_2}=[0\ \ 0]$,$o_1 \parallel o_2$.当站点 2 接收到站点 1 的插入操作时,由于已经删除了 b,操作 o_1 被转换到位置 $p=1$ 执行.同理,当站点 1 接收到站点 2 的操作时,由于插入的字符 d 并没有移动字符 b 的位置,所以仍然执行 $del(1)$.最终,两个站点都获得了一致的结果.

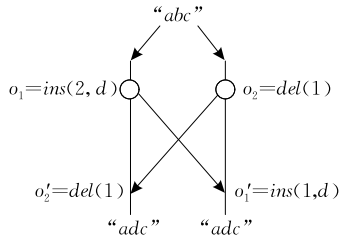


图 2 操作转换的应用实例

Ressel 等人[18]提出了 OT 算法的充分必要条件 TP1 和 TP2,分别对应图 1(a)和图 1(b)的场景.TP1 确保两个并发操作 $o_1 \parallel o_2$,先执行 o_1 再执行 o_2' 的结果等于先执行 o_2 再执行 o_1' 的结果.TP2 确保 3 个并发操作 $o_1 \parallel o_2 \parallel o_3$,$o_3$ 对 $[o_2\ \ o_1']$ 和 $[o_1\ \ o_2']$ 转换之后得到相同的形式,其中 $[o_2\ \ o_1']$ 和 $[o_1\ \ o_2']$ 的执行效果相同.

定义 3. 偏并发关系.当两个并发操作产生于不同的文档状态时,称这两个操作是偏并发关系.

操作的位置 p 都是相对于某个文档状态的,不同文档状态下的 p 不具有可比性.站点产生操作 o 时的文档状态称作 o 的定义状态,也称操作上下文(operation context)[39],记为 $C(o)$.文档状态由从原始状态到当前状态所执行的操作序列来表示.在图 1(c)中,$C(o_3)=[\,]$、$C(o_2)=[o_1]$ 并且 $o_2 \parallel o_3$,因此 o_2 与 o_3 是偏并发的关系.假设 $o_1=ins(2,a)$、$o_2=ins(3,b)$、$o_3=del(2)$,o_3 对 $[o_1\ \ o_2]$ 转换之后得到 $o_3'=del(4)$,o_1 对 o_3 转换之后得到 $o_1'=ins(2,a)$.由于 $C(o_2)\neq C(o_3)$,o_2 对 o_3 直接转换之后得到 $o_2'=ins(2,b)$,$[o_1\ \ o_2\ \ o_3']\neq[o_3\ \ o_1'\ \ o_2']$,造成了不一致的结果.可以看出,偏并发操作之间不能直接进行转换.

定义 4. 等价操作集合.从同一文档状态出发,执行两个不同操作集合之后得到相同的状态,它们被称作等价操作集合.

在接收到所有操作之后,尽管每个站点执行了不同的操作序列,但是满足 TP1 和 TP2 条件的转换函数能够保证所有站点已执行的操作集是等价的.另外,控制过程也会根据已执行的操作构造等价的操作历史.

3.1 转换函数

OT 算法需要定义元操作之间的互相转换,称作包含转换函数 IT.按照已经建立的符号,对于任何一个操作 o,$o.t$ 表示操作的类型(插入、删除),$o.c$ 表示操作的作用字符,$o.p$ 表示操作的位置,$o.id$ 表示产生该操作的站点标识,$o.num$ 表示该操作是站点的第 num 个操作.操作转换只会改变 $o.p$ 的值,其他属性保持不变.

$IT(o_1,o_2)$ 要求 o_1 和 o_2 是定义在相同文档状态下的并发操作,转换之后 $C(o_1')=C(o_2)+[o_2]$.为了处理偏并发操作,OT 算法定义了 IT 的逆过程排除转换函数 ET,目的是将在不同状态下产生的操作转换到相同状态下.函数 1、2 给出了常见的转换函数的具体定义.$\varnothing$ 代表空操作,不产生任何实际效果.

函数 1. $IT(o_1,o_2):o_1'$.

```
1. o1' := o1;
2. IF o1.p > o2.p
3.     IF o2.t = ins
4.         o1'.p := o1.p + 1;
5.     ELSE
6.         o1'.p := o1.p - 1;
7.     ENDIF
8. ELSIF o1.p = o2.p
9.     IF o1.t = o2.t = ins && o1.id > o2.id
10.        o1'.p := o1.p + 1;
11.    ELSIF o1.t = del && o2.t = ins
12.        o1'.p := o1.p + 1;
13.    ELSIF o1.t = o2.t = del
14.        o1' := ∅;
15.    ENDIF
16. ENDIF
17. RETURN o1';
```

函数 2 给出的 ET 的定义并没有包括所有的情况.假如 $o_1=ins(p+1,c)$、$o_2=ins(p,c)$、$o_3=del(p)$,那么 $IT(o_1,o_3)=IT(o_2,o_3)=ins(p,c)$.可

以看出，IT 不是单射函数，所以 ET 并不能满足可逆的要求. 这种情况下，只有在调用 $IT(o_1,o_3)$ 之前记录 o_1 的位置信息，$ET(o_1',o_3)$ 时才会返回 o_1. 关于 ET 和 IT 的可逆性将在 4.3 节进行详细讨论. $SWAP$ 函数通过先调用 $ET(o_1,o_2)$ 得到 o_1'，再调用 $IT(o_2,o_1')$ 得到 o_2'，实现交换操作的执行顺序，$[o_2\ o_1]=[o_1'\ o_2']$.

函数 2.　$ET(o_1,o_2):o_1'$.

1. $o_1':=o_1$;
2. IF $o_1.p>o_2.p$
3. 　IF $o_2.t=ins$
4. 　　$o_1'.p:=o_1.p-1$;
5. 　ELSE
6. 　　$o_1'.p:=o_1.p+1$;
7. 　ENDIF
8. ELSIF $o_1.p=o_2.p$
9. 　IF $o_1.t=o_2.t=del$
10. 　　$o_1'.p:=o_1.p+1$;
11. 　ENDIF
12. ENDIF
13. RETURN o_1';

3.2　控制过程

控制过程负责调度远程操作与站点 HB 中的操作进行转换. 如果远程操作 o 的因操作还未被接收，则将该操作加入到等待队列中，直到发生在 o 之前的所有操作都已接收，o 才会被控制过程集成到 HB 中. 只有操作上下文相同的操作才能通过操作转换获得可交换执行. 因此，控制过程的首要任务是寻找远程操作 o 正确的操作上下文环境. 当站点调用控制过程集成操作 o 时，显然接收站点已经将 o 所有的因操作集成到了 HB 中. 如果能够将 o 的因操作和并发操作区分开来，那么因操作集合就等价于 $C(o)$. 然后，对并发操作逐个进行包含转换，将操作的定义状态递增到当前的文档状态，即可获得该操作在远程站点的执行形式. 具体而言，假如两个站点的操作历史如下：

站点 1.　$H_1=[o_{1a}\ o_{1b}\ o_{1c}]$;

站点 2.　$H_2=[o_{2a}\ o_{1a}'\ o_{2b}\ o_{1b}'\ o_{2c}]$,

站点 2 接收到操作 o_{1c}，$C(o_{1c})=[o_{1a}\ o_{1b}]$，而 $C(o_{2a})=[]$，不能直接调用 $IT(o_{1c},o_{2a})$. 如果站点 2 的 $H_2'=[o_{1a}\ o_{1b}\ o_{2a}\ o_{2b}\ o_{2c}]$，$C(o_{1c})=C(o_{2a})$，$o_{1c}'=IT(o_{1c},o_{2a})$ 并且 $C(o_{1c}')=C(o_{2b})$. 依此类推，o_{1c} 对 $[o_{2a}\ o_{2b}\ o_{2c}]$ 转换之后等同于当前文档状态下产生的操作. 关键的步骤在于构造 H_2 的等价操作集合 H_2'.

Suleiman 等人[25] 设计了 $Separate$ 函数，负责把 HB 重构成因操作在前并发操作在后. 具体做法：从左到右扫描站点操作历史 H 并用 l 记录当前访问的最后一个因操作在 H 中的位置，如果 $H[i]$ 是操作 o 的因操作，则将操作 $H[i]$ 交换到 $H[l+1]$ 所在的位置；如果 $H[i]$ 是 o 的并发操作，则跳过当前位置；遍历结束之后，$H[0:l]$ 即为操作 o 的因操作. $ITSQ$ 函数是对单个操作转换的封装，表示操作 o 对操作序列 sq 进行包含转换. 图 3 描述了 Suleiman 等人的 HB 结构.

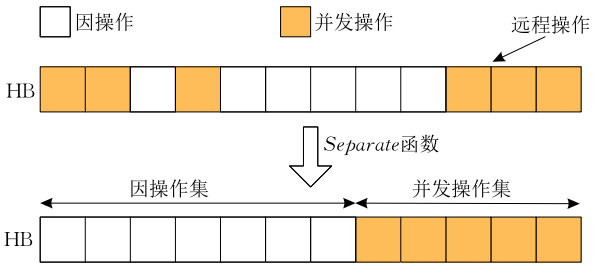


图 3　Suleiman 等人的 HB 结构

函数 3.　$Separate(H,o):l$.

1. $l:=-1$;
2. FOR $i:=0$ up to $|H|-1$ //$|H|$表示 H 中操作的数目
3. 　$o_i:=H[i]$;
4. 　IF $\boldsymbol{SV}_{o_i}[S_{o_i}]<\boldsymbol{SV}_o[S_{o_i}]$ THEN
　　//S_{o_i} 代表 o_i 的站点 id
5. 　　FOR $j:=i$ down to $l+2$ do
6. 　　$SWAP(H[j],H[j-1])$;
7. 　　ENDFOR;
8. 　　$l:=l+1$;
9. 　ENDIF;
10. ENDFOR;
11. RETURN l;

从以上的分析可以看出，OT 算法集成远程操作的性能瓶颈在于 $Separate$ 函数，找出远程操作的并发操作集 H_c 需要耗费 $O(|H|^2)$ 的计算时间.

函数 4.　$ITSQ(o,sq):o'$.

1. $o':=o$;
2. FOR $i:=0$ up to $|sq|-1$
3. 　$o':=IT(o',sq[i])$;
4. 　IF $o'=\varnothing$　//如果 o' 是空操作 $\varnothing$
5. 　　BREAK;
6. 　ENDIF;
7. ENDFOR;
8. RETURN o';

3.3 一致性模型

协同编辑系统的一致性模型定义为下面的 3 个条件：

(1)因果保持. 如果 $o_1 \rightarrow o_2$，则在所有站点 o_1 都在 o_2 之前执行.

(2)结果一致. 当产生的所有操作在每个站点执行之后，所有的站点得到相同的结果.

(3)意图保持[28]. 对于任何一个操作 o，在远程站点执行的效果与该操作在本地站点执行的效果相同，且 o 的执行不改变并发操作的效果.

意图保持能够尽量保证最终的结果符合用户的意愿. 由于 Sun 等人[28]的意图定义比较模糊，Li 等人[29]将操作的意图定义成操作作用对象之间的位置关系. 假如从全局观察所有站点执行的操作，作用对象之间形成了全序的位置关系，遵循这种关系的操作也就意味着操作意图的保持. 图 4 描述了 3 个站点的协同过程. 在图 4(b)～(e)中，有向边的起点所代表的字符在前，终点代表的字符在后. 图 4(a)中，初始位置关系就是初始文档中字符的先后关系，所有站点都是一致的. 在各个站点的本地操作执行之后，y 和 x 之间的位置关系就确定了. 随着远程操作的执行，之前建立的位置关系可能被破坏. 根据图 4(c)能够推导出 y 在 x 前面，而图 4(d)中加入远程操作之后，x 到 y 有了通路，在图中形成了环路，违背了已建立的位置关系. 图 4(e)给出了意图保持的正确结果. 如果随机定义图 4(a)中 3 个操作之间的全局顺序，所有站点都能得到一致的结果. 不同的顺序将产生不同的结果 $axyc$ 或者 $ayxc$. 然而 Sun 等人认为在结果一致的前提下，两种结果都能保持意图.

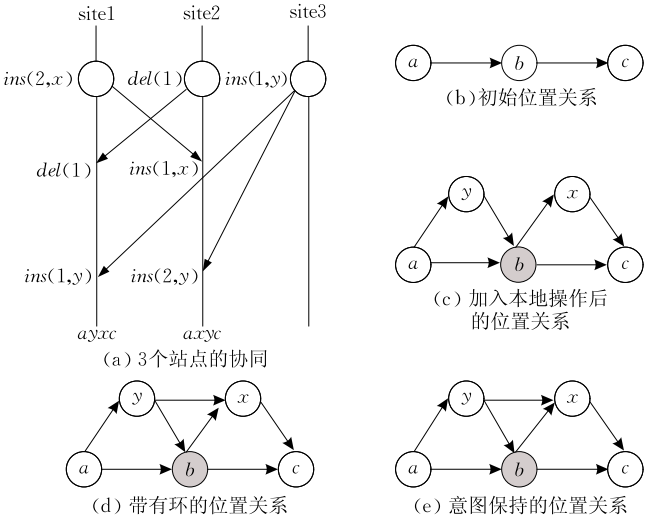


图 4 意图保持的实例

4 MOOT 算法

协同编辑环境中存在大量的无效操作，即插入某个对象之后，这个对象又被随后的操作删除. 这样的一对插入删除操作不会改变文档状态，合并的效果等同于空操作. 如果移除 HB 中的无效操作，能够减少操作历史中的操作数目，从而减少在集成远程操作时所耗费的计算时间. ABT 算法构造了特殊的操作历史结构，获得了更高的计算效率. 然而站点 HB 的等价操作集合并非是唯一的，与 ABT 相比，MOOT 算法提出了一种更优的 HB 结构.

4.1 HB 排列结构

在没有使用全序的 OT 算法中，操作历史中的操作按照站点的执行顺序排列. 在集成远程操作时，首先调用 *Separate* 函数划分出并发操作集 H_c，然后调用 *ITSQ* 函数对 H_c 进行包含转换，总共花费计算时间 $O(|H|^2+|H_c|)$. 非全序 OT 算法的 HB 中的操作是按照操作的全序顺序排列的. 由于并发操作之间并没有自然的先后关系，所以全序的定义依赖于人工干预，并且需要兼容因果关系. 站点可以按照全局顺序接收操作，而这种方式容易造成比较大的延时. 站点也可以按照因果顺序接收，如果违背了全局顺序则利用 Undo/Do/Redo 的模式维护全序的 HB 结构，这种方法需要大量的重复计算，效率非常低.

ABT 算法提出了一种特殊的排列结构，即插入操作在前删除操作在后，记 $H=H_i \cdot H_d$. 为了讨论方便，将这种排列结构称作 IDHB. 在集成本地操作时，产生的操作 o 被立即执行. 然后，调用 *ET* 排除删除操作的影响得到 o'，并广播 o' 到其他远程站点. 同时，为了继续维护 IDHB 这种结构，如果 o 是插入操作，则将 o' 添加到 H_i 中；如果 o 是删除操作，则将 o 添加到 H_d 中. 在集成远程操作 o 时，首先调用 $Separate(o,H_i)$ 将插入操作集 H_i 划分成因操作集 H_{ih} 和并发操作集 H_{ic}. 操作 o 对 H_{ic} 进行包含转换得到 o''，再对 H_d 进行包含转换得到 o'，o' 就是远程操作在本站点的执行形式. 当 o 是删除操作时，将 o' 添加到 H_d 中，时间复杂度为 $O(|H_i|^2+|H_{ic}|+|H_d|)$；当 o 是插入操作时，将 o' 与 H_d 中的每个操作进行交换，最终将 o'' 添加到 H_i 中，耗费计算时间 $O(|H_i|^2+|H_{ic}|+2|H_d|)$. 可以看出 ABT 算法在集成远程操作的计算时间主要取决于插入操作集的大小.

在编辑文档时，插入操作的数量明显多于删除操作的数量. 因此，MOOT 算法构造了删除操作在前、插入操作在后的 DIHB(Delete-Insert HB)结构，使得算法集成远程操作的计算时间取决于较小的操作集.

4.2　意图保持

尽管 HB 可以重构成任何等价操作集合，但并不是所有的结构都能遵循一致性模型. 2014 年 Sun 等人[40]穷举了造成 OT 算法不能满足意图保持的 puzzle，得出的结论是所有的 puzzle 都源自于一组特殊的操作位置关系($o_1=ins(p+1,c_1)$、$o_2=del(p)$、$o_3=ins(p,c_2)$并且 $o_1\|o_2\|o_3$). 因此，有必要分析在这种特殊情况下，DIHB 能否实现意图保持. 图 5 描述了在这种特殊的场景下，DIHB 维护数据一致性的过程.

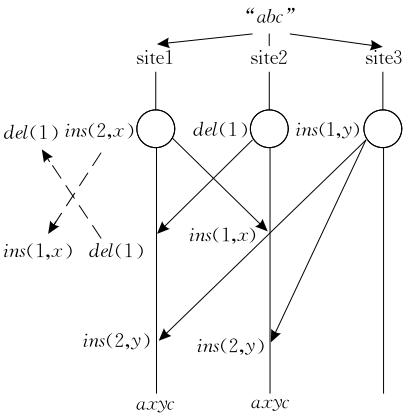


图 5　DIHB 实现 Sun 等人的意图保持

假设在同一个位置插入字符，站点优先级小的操作位置 p 保持不变，优先级大的操作位置 p 加 1 (site1<site3). 图 5 中，站点 1 和站点 2 的操作历史都是相同，最终的结果也是相同的，即满足了 Sun 等人提出的意图保持条件，但不符合 Li 等人提出的操作位置关系的保持. 按照作用关系图，可以判断 y 在 x 前面. 在站点 2，由于删除操作会对插入操作的位置造成影响，导致原本不在相同位置的插入操作需要在同一个位置进行比较，经过站点优先级比较之后，x 出现在了 y 的前面，形成了环路. 因此，要想保持插入对象之间的位置关系，插入操作需要保存在位置 p 之前所执行的删除操作的信息.

利用 Randolph 等人[19]提出的满足 TP1 和 TP2 条件的包含转换函数，可以实现 Li 等人提出的意图保持标准. 插入操作添加属性 nd(number of deleted operations)，记录在位置 p 之前执行的删除操作的数目，相当于保持了插入操作之间的位置关系. 如果两个相同位置的插入操作进行转换，首先比较 nd 的大小，nd 大的操作位置加 1 而 nd 小的操作位置不变，其次再比较站点优先级. 该方法在 TTF[31]算法中已经证明能够保持作用对象之间的位置关系. 在转换过程中，nd 也会根据删除操作而变化. 给定操作 $o_1=ins(p,c,nd,id)$和 $o_2=del(p)$，如果 $o_1.p>o_2.p$，则 $o_1'=IT(o_1,o_2)$并且 $o_1.nd$ 加 1. 在图 6 中，由于 nd 属性的引入，DIHB 结构能够保持作用对象之间的位置关系. 在站点 2，站点 1 的操作 $ins(2,x,0)$和站点 3 的操作 $ins(1,y,0)$对 $del(1)$进行转包含转换之后都变成了在位置 $p=1$ 上的操作，通过比较 nd，可以判断出 y 应该在 x 前面，站点 3 的操作在站点 2 按照原始形式执行.

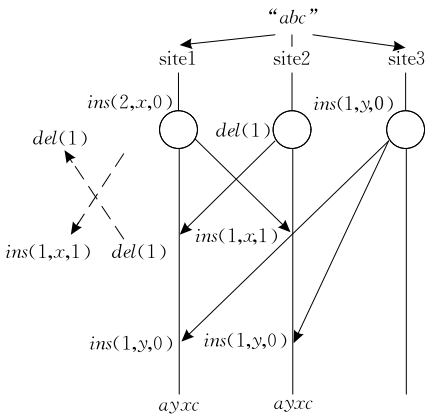


图 6　DIHB 实现 Li 等人的意图保持

4.3　ET 和 IT 的可逆性

重构等价操作历史依赖操作的可交换性. 根据 3.1 节的描述，当 $o_1.t=ins$、$o_2.t=del$ 且 $o_1\|o_2$ 时，$IT(o_1,o_2)$不是可逆的. 然而，MOOT 算法并不会在这种情况下调用 $ET(o_1,o_2)$. 维护 DIHB 结构不需要交换插入操作到删除操作的前面，因此并发操作之间具有可交换性. 当 o_1 和 o_2 是因果关系的时候，ET 也不一定是可逆的. 例如，来自同一个站点的两个操作 $o_1=ins(1,a)$、$o_2=ins(2,b)$，$o_2'=ET(o_2,o_1)=ins(1,b)$，再调用 $IT(o_2',o_1)=ins(1,b)\neq o_2$. 根据总结，只有当同时出现以下情况时，

(1) $o_1.t=o_2.t=ins$；

(2) $o_1\rightarrow o_2$；

(3) $o_2.p=o_1.p\ \|\ o_2.p=o_1.p+1$；

$ET(o_2,o_1)$不是可逆的，记 ET 返回 false. MOOT 算法不会向前交换 HB 中两个插入操作的位置，因此除去上述情况的因果操作之间也具有可交换性.

从上面的分析可以看出，当 HB 中两个相邻的插入操作具有特殊位置关系时，ET 不是可逆的. 文献[41]指出这种关系在远程站点依然成立. 如果已知操作位置的间距，很容易根据其中一个操作推导出另外一个操作. 例如，已知 $ET(o_1,o_2)=$false 并且 $o_1.p-o_2.p=0$，那么 o_1 在远程站点的 HB 也位于 o_2 之后，并且 $o_1.p=o_2.p$.

4.4 压缩策略

DIHB 结构能够方便地识别出无效操作对. 在重构 HB 的过程中，需要将删除操作交换到插入操作的前面. 如果操作 o_1 删除的对象就是另外一个操作 o_2 插入的对象时，那么操作 o_1 依赖于 o_2 的存在，无法将 o_1 交换到 o_2 之前执行. 在图 7 中，当调用 $SWAP$ 交换 $del(1)$ 和 $ins(1,a,0)$ 时返回 false，可以判断出两个操作是无效操作.

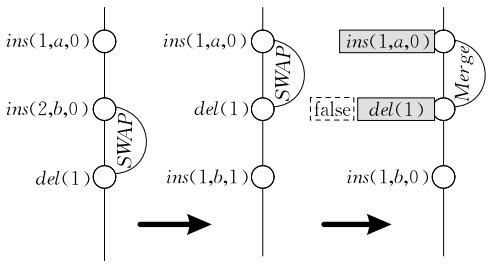


图 7 压缩无效操作对

HB 中不需要保存无效的操作，移除它们不影响其他操作. 在本地站点，如果产生的删除操作 o_1 不能与 HB 中的插入操作 o_2 交换，则从 HB 中移除 o_2 并更新与 o_1 已交换位置的操作的 nd 属性. 同时，将 o_1 作为 o_2 的 undo 操作广播到其他站点. 当远程站点接收到 undo(o_2) 时，首先，在 HB 的第 k 个位置取出该操作；然后，将 o_2 与 $k+1$ 到 n 位置上的操作逐一交换位置得到 o_2'，o_2' 的逆操作即为 o_1 的执行形式；同时将 o_2' 从 HB 中删除. 每个操作都由站点 id 以及逻辑时钟 num 组成的二元组唯一确定，访问 HB 中的任何一个操作是很方便的. 从上面的过程可以看出集成 undo 操作的时间复杂度是线性的.

4.5 Undo 扩展

作为错误恢复的重要机制，实时协同编辑系统需要支持 anyundo(undo any operations at any time)[23,42]. 由于 MOOT 提出的压缩机制，HB 中的某些操作会被删除，造成 undo 目标操作的丢失. 为了能够支持 undo 功能，本文给出一种可行的处理方法，即在每个协同站点额外维护完整的 HB(Full HB，FHB) 以及压缩过的 HB(Compressed HB，CHB). MOOT 的压缩策略并非是发出真正的 undo 命令，而是对撤销操作的模拟. 为了能够区别 undo 操作，可以设置其他的标志来表示删除操作抵消插入操作的请求. 利用已有的 undo 算法和 FHB 计算 undo/redo 操作的执行形式 o'，将 o' 加入到 FHB. 而 CHB 则用于计算普通操作 o 的执行形式. 由前面的叙述可知，FHB 和 CHB 是等价操作集合.

4.6 算法描述

每个站点不仅会产生操作也会接收其他站点的操作. 每个站点需要维护接收队列 R 和状态向量 $\boldsymbol{SV}$.

本地站点 i 产生的操作 o 会被立即执行，然后调用函数 $IntegrateLocal$ 将 HB 重构成 DIHB，$\boldsymbol{SV}[i]=\boldsymbol{SV}[i]+1$，并广播请求 r. 算法 1 排除插入操作 H_i 对操作 o 的影响得到 o'. 根据前面的叙述，不是所有的操作都能交换执行顺序. 如果 $SWAP(o,o_x)=$false，其中 $o_x\in H_i$，r 表示成六元组 $\langle o.id,o.num,flag,o_x.id,o_x.num,o.c\rangle$，当 $o.t=o_x.t=ins$ 时，$flag=(o.p-o_x.p)$ 表示 o 和 o_x 的位置关系；当 $o.t=del$ 并且 $o_x.t=ins$ 时，$flag=$undo 表示 o 作为 o_x 的 undo 操作. 如果 $SWAP(o,o_x)=$true，其中 $o_x\in H_i$，r 表示成四元组 $\langle o.id,o.num,flag,o'\rangle$.

算法 1. $IntegrateLocal$.

输入：站点产生的操作 o

输出：广播到其他站点的请求 r

```
1.  o' := o;
2.  For k := |H_i| − 1 down to 0
3.      [o', isValid] := ET(o', H_i[k]);
4.      IF isValid = false
5.          r := ⟨id, num, 0/1/undo, H_i[k].id,
                  H_i[k].num, o.c⟩;
            //如果是 undo，则删除 H_i[k]
6.          RETURN r;
7.      ELSE
8.          IF o'.t = del
9.              IT(H_i[k], o');
10.         ENDIF
11.     ENDIF
12. ENDFOR
13. r := ⟨id, num, null, o'⟩;
14. IF o.t = ins
15.     H_i.add(o);
16. ELSE
17.     H_d.add(o');
18. ENDIF
19. RETURN r;
```

站点 j 接收到站点 i 的请求时，首先判断是否满足因果接收条件. 如果 $\boldsymbol{SV}_j[i]=r[2]-1$，则调用 *IntegrateRemote* 函数；否则，加入接收请求队列 R. 站点调用线程周期性的访问 R，寻找满足条件的请求并处理.

在集成远程操作时，根据请求 r 推导出待集成的操作 o，经过转换得到的 o' 就是远程操作在本站点的执行形式. 如果 $r.flag=$ undo，则根据 4.4 节描述的过程进行计算；如果 $r.flag=0/1$，则在 HB 中的第 k 个位置取出 $\langle id=r[4], num=r[5]\rangle$ 的操作 o_x，$o.p=o_x.p+0/1$，将 o 与 $k+1$ 到 n 位置上的所有操作进行包含转换得到 o'，执行 o'；如果 $flag=$null，则首先调用 $Separate(o, H_d)$ 将删除操作集 H_d 转换成因操作集 H_{dh} 和并发操作集 H_{dc}. 然后，对 H_{dc} 进行包含转换得到 o''，再对 H_i 进行包含转换得到 o'. 当 o 是插入操作时，将 o' 添加到 H_i 中；当 o 是删除操作时，将 o'' 对插入操作集 H_i 进行对称包含转换，最终将 o'' 添加到 H_d 中.

算法 2.　*IntegrateRemote*.

输入：接收的请求 r

输出：远程操作 o 的执行形式 o'

```
1.  IF r.flag=undo THEN
2.     o:=H_i[k]; //H_i[k].⟨id,num⟩=⟨r[4],r[5]⟩
3.     o'':=SWAP(o,H_i[k+1:|H_i|-1]);
4.     o':=inverse(o''); H_i.delete(o'');
5.  ELSEIF r.flag=0/1
6.     o:=H_i[k]; //H_i[k].⟨id,num⟩=⟨r[4],r[5]⟩
7.     o'':=ins(o.p+0/1,r[6],o.nd,r[1],r[2]);
8.     o':=ITSQ(o'',H_i[k+1:|H_i|-1]);
9.  ELSE
10.    o:=r[4];
11.    n_dc:=Separate(o,H_d);
12.    o'':=ITSQ(o,H_dc);
13.    IF o''=∅ THEN
14.       RETURN o'';
15.    ENDIF
16.    IF o''.t=ins THEN
17.       o':=ITSQ(o'',H_i); H_i.add(o');
18.    ELSE
19.       o_x:=o'';
20.       FOR k:=0 up to |H_i|-1
21.          o_y:=o_x;
22.          o_x:=IT(o_x,H_i[k]);
23.          H_i[k]:=IT(H_i[k],o_y);
24.       ENDFOR
25.       o':=o_x; H_d.add(o'');
26.    ENDIF
27. ENDIF
28. RETURN o';
```

4.7　性能分析

本地产生的操作 o 都会被立即执行，耗费计算时间 $O(1)$. 如果 $o.t=ins$，调用 ET 排除 H_i 中所有操作的影响并遍历 H_d 计算 $o.nd$，耗费计算时间 $O(|H_i|+H_d|)$；如果 $o.t=del$，调用 $SWAP$ 将 o 交换到所有插入操作的前面，耗费计算时间 $O(|H_d|)$.

当站点接收到远程操作 o，首先调用 *IntegrateRemote* 计算出 o 在本站点的执行形式 o'，然后执行 o'. 如果接收到的请求 $r.flag=0/1$（本质上接收的是插入操作），则首先在 H_i 找到相关操作 o_k 并求出 o，然后调用 $ITSQ(o, H_i[k+1:|H_i|-1])$ 计算出 o'，耗费计算时间 $O(|H_i|)$；如果 r 是 undo 请求（本质上接收的是删除操作），则也是首先在 H_i 找到相关操作 o_k 并调用 $SWAP$ 将其交换到 H_i 的尾部得到 o_k'，o' 等于 o_k' 的逆操作，耗费计算时间 $O(|H_i|)$；如果请求 $r.flag=$null，则首先调用 *Separate* 函数将 H 划分成 $H_d \cdot H_i$，然后调用 $ITSQ(o, H_{dc}+H_i)$ 计算出 o'，耗费计算时间 $O(|H_d|^2+|H_{dc}|+|H_i|)$，对于删除操作，还需要将 o' 调整到 H_i 之前. 表 1 给出了集成操作到 HB 中的计算时间复杂度，包括操作的执行时间以及维护 DIHB 结构的时间.

表 1　MOOT 算法时间复杂度

	远程操作		本地操作
	$r.flag=$null	$r.flag\neq$null	
插入操作	$O(\|H_d\|^2+\|H_{dc}\|+\|H_i\|)$	$O(\|H_i\|)$	$O(\|H_i\|+\|H_d\|)$
删除操作	$O(\|H_d\|^2+\|H_{dc}\|+2\|H_i\|)$	$O(\|H_i\|)$	$O(\|H_i\|)$

本地操作都是先执行然后才被集成到 HB 中，因此集成本地操作的过程不会影响操作的执行. 而接收到的远程操作则要求尽快得到执行，集成远程操作的计算效率决定了操作的响应时间. 在上面的讨论中，根据 $\langle id, num\rangle$ 找到第 k 个位置上的操作，可以通过遍历 H_i 的方式也可以通过建立哈希表来访问. 算法维护 HB 需要耗费 $O(|H|)$ 的空间.

5　实例分析

本节用一个实例场景来解释 MOOT 算法的执行过程. 协同场景为 3 个站点的协同，站点之间的优

先级设定为 site1＜site2＜site3，初始编辑状态 $s=$“abc”. 3 个站点并发产生操作 $o_1=ins(2,y)$、$o_2=del(1)$和 $o_3=ins(1,x)$. 站点 1 在接收到 o_2之后产生 $o_4=ins(2,z)$；站点 3 在接收到 o_1之后产生操作 $o_5=del(1)$. 各个站点发送、接收操作的顺序如图 8 所示. 最终所有站点得到一致的结果“$ayzc$”.

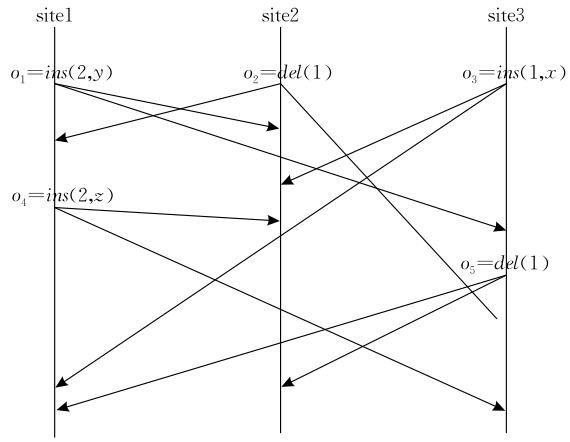


图 8 3 个站点的协同编辑

在站点 1：o_1立即执行，$H_1=[o_1]$；接收到 o_2，调用 $IT(o_2,o_1)$得到$o_2'=del(1)$，执行 o_2'，$H_1=[o_2\ \ o_1']$，其中 $o_1'=IT(o_1,o_2)=ins(1,y,1)$；$o_4$立即执行并对 o_1'进行排除转换 $ET(o_4,o_1')=$ false，将请求 $r=\langle 1,2,1,1,1,z\rangle$广播到其他站点，$H_1=[o_2\ \ o_1'\ \ o_4]$；接收到 o_3，由于 $o_3\parallel o_1\parallel o_2$，$o_3'=ITSQ(o_3,H_1)=ins(1,x,0)$，执行 o_3'，$H_1=[o_2\ \ o_1'\ \ o_4\ \ o_3']$；接收到 o_5，由于o_5是o_3的 undo 操作，$o_5'=inverse(o_3')=del(1)$，执行 o_5'，将 o_3'从 H_1中删除，$H_1=[o_2\ \ o_1'\ \ o_4]$. 此时 $s=$“$ayzc$”.

在站点 2：o_2立即执行，$H_2=[o_2]$；接收到 o_1，$o_1'=IT(o_1,o_2)=ins(1,y,1)$，执行 o_1'，$H_2=[o_2\ \ o_1']$；接收到 o_3，$o_3'=ITSQ(o_3,H_2)=ins(1,x,0)$，执行 o_3'，$H_2=[o_2\ \ o_1'\ \ o_3']$；接收到 o_4，在 H_2中找到 o_1'，$o_4=ins(2,z,1)$，$o_4'=IT(o_4,o_3)=ins(3,z,1)$，执行 o_4'，$H_2=[o_2\ \ o_1'\ \ o_3'\ \ o_4']$；接收到 o_5，由于 o_5是 o_3的 undo 操作，$o_3''=SWAP(o_3',o_4')=ins(1,x,0)$，$o_4''=ins(2,z,1)$，$H_2=[o_2\ \ o_1'\ \ o_4''\ \ o_3']$，$o_5'=inverse(o_3'')=del(1)$，执行 o_5'，从 H_2中删除 o_3''，$H_2=[o_2\ \ o_1'\ \ o_4'']$. 此时 $s=$“$ayzc$”.

在站点 3：o_3立即执行，$H_3=[o_3]$；接收到 o_1，将 o_1与 o_3进行包含转换得到 $o_1'=ins(3,y,0)$，$H_3=[o_3\ \ o_1']$；o_5立即执行，$o_5'=SWAP(o_5,o_1')=del(1)$，$o_1''=IT(o_1',o_5')=ins(2,y,1)$，$ET(o_5',o_3')=$ false，恢复$o_1''.nd=0$，从 H_3中删除 o_3'并将 $r=\langle 3,2,$undo$,3,1,x\rangle$发送出去，$H_3=[o_1'']$；接收到 o_2，$o_2'=IT(o_2,o_1'')=del(1)$，执行$o_2'$，$H_3=[o_2\ \ o_1''']$，其中$o_1'''=IT(o_1'',o_2)=ins(1,y,1)$；接收到 o_4，在 H_3中查找到 o_1'''，$o_4=ins(2,z,1)$，执行 o_4，$H_3=[o_2\ \ o_1'''\ \ o_4]$. 此时 $s=$“$ayzc$”.

6 实验研究

由于 ABT 的时间复杂度低于当前其他的 OT 算法，因此本文通过比较 MOOT 和 ABT 算法的性能来检 MOOT 算法的性能. 所有的算法都采用 C＃编写. 对比实验运行的计算机配置为 Intel Quad Q8300 2.5 GHz 和 2 GB RAM. 操作系统是 Windows 7 Ultimate 32. 参照文献[11,15,37]，实验模拟两个站点之间的协同. 首先，站点 1 和站点 2 并发地产生 M 个操作和 N 个操作；然后，测试将这 N 个操作集成到远程站点 1 的计算时间. 在 300 000 个字符的文档上产生插入、删除操作，操作的位置是均匀分布的. M 和 N 的取值在 300～3000 之间，步长为 300. 测试在插入操作占比例 60％和 80％时，MOOT 和 ABT 算法集成远程操作的计算时间. 实验结果展示在图 9～图 12 中.

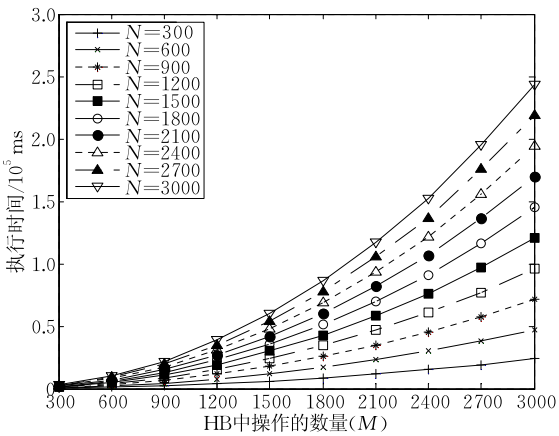


图 9 ABT 在插入操作比例为 60％时的计算时间

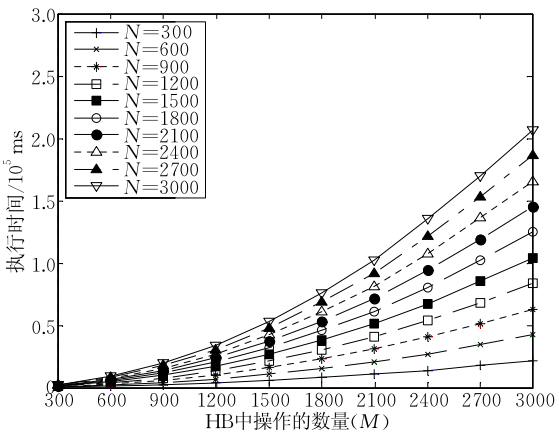


图 10 MOOT 在插入操作比例为 60％时的计算时间

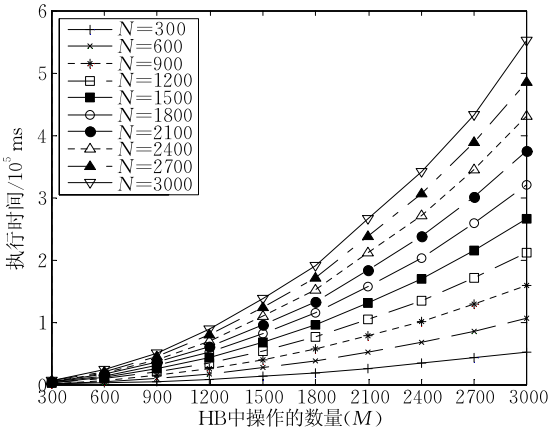


图 11　ABT 在插入操作比例为 80%时的计算时间

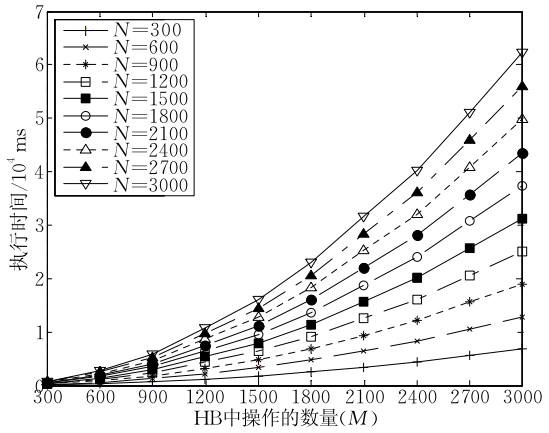


图 12　MOOT 在插入操作比例为 80%时的计算时间

在插入操作比例为 60%时，MOOT 集成远程操作的计算时间比 ABT 提高了 20%左右，并且二者都随着 M 的增大呈现多项式增长. ABT 耗费大约 50 s 集成 $N=2400$ 个操作到远程站点 HB($M=1500$ 个操作)而 MOOT 能集成 2700 个操作. 当 $M=2100$ 的时候，MOOT 每集成 300 个操作需要大约 10 s 的计算时间，ABT 则需要大约 12 s.

在插入操作比例为 80%(文献[15]中指出的 reasonable 比例)时，MOOT 计算效率比 ABT 提高了接近 10 倍. 当 $M=2400$ 的时候，MOOT 每集成 300 个操作花费 4 s 的计算时间，而 ABT 需要 35 s. 当 $M=3000$ 的时候，MOOT 集成 3000 个操作总共需要 61 s，而 ABT 大概需要 560 s.

从图 9 到图 12 可以看出，随着插入操作增多，ABT 算法的计算时间不断在增加，而 MOOT 算法的计算时间却在减少. 假如响应时间要求不高于 100 ms，M 和 N 则需要取合适的值. 例如，当插入比例为 80%时，MOOT 算法能够在 100 ms 内集成 10 个操作到 $M=2100$ 的远程站点. 由于产生的操作位置是均匀分布的，因此无效操作和 $r.flag \neq \text{null}$ 的比例非常小. 如果 20%的删除操作是为了撤销之前插入操作的效果，那么集成操作到 $M=3000$ 和 $M=2400$ 的远程站点耗费的时间是相等的. 可以预期在实际应用中，MOOT 算法计算效率更高. 总之，与 ABT 算法相比，MOOT 在集成远程操作时性能上有较大幅度的提升.

7　总　结

MOOT 算法利用了协同编辑环境的两个特性，即协同站点的操作历史保存了大量无效操作以及插入操作的比例大于删除操作. 本文提出的 DIHB 结构能够有效减少 HB 中操作的数量，同时避免算法的时间复杂度依赖于多数操作. MOOT 算法不仅能够满足意图保持的条件而且提高了 OT 算法的计算效率. 下一步的工作，考虑将 MOOT 算法扩展到支持复杂对象的实时协同.

参　考　文　献

[1] Lamport L. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 1978, 21(7): 558-565

[2] Sun D, Xia S, Sun Chengzheng, Chen D. Operational transformation for collaborative word processing//Proceedings of the ACM Conference on Computer Supported Cooperative Work. Chicago, USA, 2004: 437-446

[3] Sun Chengzheng, Xia S, Sun D, et al. Transparent adaptation of single-user applications for multi-user real-time collaboration. ACM Transactions on Computer-Human Interaction, 2006, 13(4): 531-582

[4] Sun Chengzheng, Wen Hongkai, Fan Hongfei. Operational transformation for orthogonal conflict resolution in real-time collaborative 2D editing systems//Proceedings of the ACM Conference on Computer Supported Cooperative Work. Seattle, USA, 2012: 1391-1400

[5] Wang Xueyi, Bu Jiajun, Chen Chun. Achieving undo in bitmap-based collaborative graphics editing systems//Proceedings of the ACM Conference on Computer Supported Cooperative Work. New Orleans, USA, 2002: 68-76

[6] Ignat C-L, Norrie M C. Draw-together: Graphical editor for collaborative drawing//Proceedings of the 20th Anniversary Conference on Computer Supported Cooperative Work. Banff, Canada, 2006: 269-278

[7] Agustina, Sun Chengzheng. Dependency-conflict detection in real-time collaborative 3D design systems//Proceedings of the ACM Conference on Computer Supported Cooperative Work. San Antonio, USA, 2013: 715-728

[8] Gao Liping, Shao Bin, Zhu Lin, et al. Maintaining time and space consistencies in hybrid CAD environments: Framework

and algorithms. Computers in Industry, 2008, 59(9): 894-904

[9] Liu Huajun, He Fazhi, Li Xiaoxia, Huang Zhiyong. A less constraint concurrency control and consistency maintaince in collaborative CAD system. Chinese Journal of Electronics, 2013, 22(1): 15-20

[10] Cheng Yuan, He Fazhi, Cai Xiantao, Zhang Dejun. A group Undo/Redo method in 3D collaborative modeling systems with performance evaluation. Journal of Network and Computer Applications, 2013, 36(6): 1512-1522

[11] Shao Bin, Li Du, Gu Ning. A fast operational transformation algorithm for mobile and asynchronous collaboration. IEEE Transactions on Parallel and Distributed Systems, 2010, 21(12): 1707-1720

[12] Li Du, Li Rui. An admissibility-based operational transformation framework for collaborative editing systems. Computer Supported Cooperative Work, 2010, 19(1): 1-43

[13] Shao Bin, Li Du, Gu Ning. An optimized string transformation algorithm for real-time group editors//Proceedings of the 15th IEEE International Conference on Parallel and Distributed Systems. Shenzhen, China, 2009: 376-383

[14] Shao Bin, Li Du, Gu Ning. An algorithm for selective undo of any operation in collaborative applications//Proceedings of the 16th ACM Conference on Supporting Group Work. Sanibel Island, USA, 2010: 131-140

[15] Shao Bin, Li Du, Gu Ning. A sequence transformation algorithm for supporting cooperative work on mobile devices//Proceedings of the ACM Conference on Computer Supported Cooperative Work. Savannah, USA, 2010: 159-168

[16] Ellis C A, Gibbs S J. Concurrency control in groupware systems//Proceedings of the ACM SIGMOD International Conference on Management of Data. Portland, USA, 1989: 399-407

[17] Yang Guang-Xin, Shi Mei-Lin. Object data model based concurrency control in fully-replicated architecture. Chinese Journal of Computers, 2000, 23(2): 113-125(in Chinese)
(杨光信, 史美林. 全复制结构下基于对象数据模型的并发控制. 计算机学报, 2000, 23(2): 113-125)

[18] Ressel M, Nitsche-Ruhland D, Gunzenhäuser R. An integrating, transformation-oriented approach to concurrency control and undo in group editors//Proceedings of the ACM Conference on Computer Supported Cooperative Work. Boston, USA, 1996: 288-297

[19] Randolph A, Boucheneb H, Imine A, Quintero A. On synthesizing a consistent operational transformation approach. IEEE Transactions on Computers, 2014, 99(2): 1-1

[20] Vidot N, Cart M, Ferrié J, Suleiman M. Copies convergence in a distributed real-time collaborative environment//Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work. Philadelphia, USA, 2000: 171-180

[21] Sun Chengzheng, Ellis C. Operational transformation in real-time group editors: Issues, algorithms, and achievements//Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work. Seattle, USA, 1998: 59-68

[22] Li Rui, Li Du, Sun Chengzheng. A time interval based consistency control algorithm for interactive groupware applications//Proceedings of the 10th International Conference on Parallel and Distributed Systems. Newport Beach, USA, 2004: 429-436

[23] Sun D, Sun Chengzheng. Context-based operational transformation in distributed collaborative editing systems. IEEE Transactions on Parallel and Distributed Systems, 2009, 20(10): 1454-1470

[24] Sun Chengzheng, Zhang Yanchun, Jia Xiaohua, Yang Yun. A generic operation transformation scheme for consistency maintenance in real-time cooperative editing systems//Proceedings of the ACM Conference on Supporting Group Work. Phoenix, USA, 1997: 425-434

[25] Suleiman M, Cart M, Ferrié J. Serialization of concurrent operations in a distributed collaborative environment//Proceedings of the ACM Conference on Supporting Group Work. Phoenix, USA, 1997: 435-445

[26] Liao Bin, He Fa-Zhi, Jing Shu-Xu. Survey of operational transformation algorithms in real-time computer-supported cooperative work. Journal of Computer Research and Development, 2007, 44(2): 326-333(in Chinese)
(廖斌, 何发智, 荆树旭. 实时协同工作系统中操作转换算法综述. 计算机研究与发展, 2007, 44(2): 326-333)

[27] Zheng Yang, Shen Haifeng, Sun Chengzheng. Leveraging single-user AutoCAD for collaboration by transparent adaptation//Proceedings of the 13th IEEE International Conference on Computer Supported Cooperative Work in Design, 2009: 78-83

[28] Sun Chengzheng, Jia Xiaohua, Zhang Yanchun, et al. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. ACM Transactions on Computer-Human Interaction, 1998, 5(1): 63-108

[29] Li Du, Li Rui. Preserving operation effects relation in group editors//Proceedings of the ACM Conference on Computer Supported Cooperative Work. Chicago, USA, 2004: 457-466

[30] Li Du, Li Rui. Ensuring content and intention consistency in real-time group editors//Proceedings of the 24th International Conference on Distributed Computing Systems. Tokyo, Japan, 2004: 748-755

[31] Oster G, Molli P, Urso P, Imine A. Tombstone transformation functions for ensuring consistency in collaborative editing systems//Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing. Atlanta, USA, 2006: 1-10

[32] Li Rui, Li Du. A new operational transformation framework for real-time group editors. IEEE Transactions on Parallel and Distributed Systems, 2007, 18(3): 307-319

[33] Preguica N, Marques J M, Shapiro M, Letia M. A commutative replicated data type for cooperative editing//Proceedings

of the 29th IEEE International Conference on Distributed Computing Systems. Montreal, Canada, 2009: 395-403

[34] Wu Qinyi, Pu C, Ferreira J E. A partial persistent data structure to support consistency in real-time collaborative editing//Proceedings of the 26th IEEE International Conference on Data Engineering. Long Beach, USA, 2010: 776-779

[35] Weiss S, Urso P, Molli P. Logoot: A scalable optimistic replication algorithm for collaborative editing on P2P networks//Proceedings of the 29th IEEE International Conference on Distributed Computing Systems. Montreal, Canada, 2009: 404-412

[36] Li Du, Li Rui. A performance study of group editing algorithms//Proceedings of the 12th International Conference on Parallel and Distributed Systems. Minnesota, USA, 2006: 300-307

[37] Li Du, Li Rui. An operational transformation algorithm and performance evaluation. Computer Supported Cooperative Work, 2008, 17(5-6): 469-508

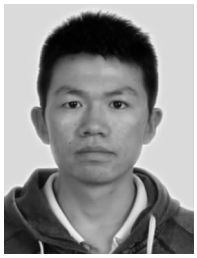
[38] Ignat C-L, Norrie M C. Customizable collaborative editor relying on treeOPT algorithm//Proceedings of the European Conference on Computer-Supported Cooperative Work. Helsinki, Finland, 2003: 315-334

[39] Sun D, Sun Chengzheng. Operation context and context-based operational transformation//Proceedings of the 20th Anniversary Conference on Computer Supported Cooperative Work. Banff, Canada, 2006: 279-288

[40] Sun Chengzheng, Xu Yi, Agustina. Exhaustive search of puzzles in operational transformation//Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing. Baltimore, USA, 2014: 519-529

[41] Imine A. Coordination model for real-time collaborative editors//Proceedings of the International Conference on Coordination Models and Languages. Lisboa, Portugal, 2009: 225-246

[42] Sun Chengzheng. Undo as concurrent inverse in group editors. ACM Transactions on Computer-Human Interaction, 2002, 9(4): 309-361



CAI Wei-Wei, born in 1988, Ph. D. candidate. His research interests include computer supported collaborative work (CSCW), collaborative CAD.

HE Fa-Zhi, born in 1968, Ph. D., professor. His research interests include CAD graphics and images, collaborative computing.

LV Xiao, born in 1983, Ph. D. candidate. Her research interests include computer supported collaborative work (CSCW), collaborative CAD.

Background

This work focuses on an optimistic concurrency control algorithm called Operational Transformation (shorted as OT), which is the key technique of real-time collaborative editing systems. There are many promising consistency maintenance methods for replicated data, such as CRDTs, OT and optimistic lock. Among these approaches, CRDT and OT have attracted large amount of research. Compared to the CRDT, OT provides a stronger consistency from the condition of intention preservation. OT algorithms replicate documents and allow local operations to be executed immediately for fast response. The local response time is not sensitive to network latencies. Remote operations that are concurrent to locally executed operations are transformed before execution. To improve the response time of remote operations, the efficiency of OT becomes an important topic. The state of the art OT algorithm is ABT algorithm, which reorders the history buffer with insertions before deletions. The special organization of history buffer simplifies the implementation of intention preservation. However, the history buffer of ABT is not the optimum. The puzzles of OT algorithms have been pointed out by many researchers, therefore the correctness of OT becomes an important issue. In 2014, two papers report the latest achievement of this topic from a different perspective. The one states that all puzzles have been discovered. The other one contributes a novel transformation function which satisfies both TP1 and TP2 property. We propose a novel operational transformation algorithm, called MOOT. The idea comes from the fact that the number of insertions is obviously more than that of deletions in collaborative editing environment. Therefore, an optimized history buffer is constructed with deletions before insertions, which reduces the time of integrating remote operations. In terms of the correctness, MOOT utilizes the transformation function which satisfies both TP1 and TP2 conditions. Furthermore, in the course of reordering history buffer, the useless operations are identified and removed. By compressing the history buffer, the algorithm reduces the computational time further. This work is supported by the National Natural Science Foundation of China (No. 61472289).