一种协同编辑中并发控制算法的研究

孙 敏 王瑞花
(山西大学计算机与信息技术学院 太原 030006)

摘 要 针对协同编辑中存在的各种不一致性问题,提出一种基于操作转换的并发控制算法ICOT。此算法是在COT算法的基础上进行改进的,通过合理利用操作的中间转换版本,减少了操作之间转换执行的次数,解决了操作转换重复的问题,同时给出了具体的实例分析来验证改进后算法的正确性和有效性,结果说明ICOT算法能使得各个编辑副本得到有效的一致性维护。

关键词 并发控制,协同编辑,序列转换,操作转换,操作上下文,一致性维护
中图法分类号 TP391 **文献标识码** A

Study of Concurrent Control Algorithm in Collaborative Editing

SUN Min WANG Rui-hua
(School of Computer & Information Technology, Shanxi University, Taiyuan 030006, China)

Abstract Aiming at the problems of various kinds of inconsistency in the collaborative editing, this paper proposed a concurrency control algorithm named ICOT(Improved Context-based Operation Transformation) based on operational transformation. This algorithm is improved on the basis of the COT algorithm and it can reduce the number of operator's transformation. It solves the problem of the repeating operation transformation by reasonable version of operation in the middle of the transformation. At the same time, this paper gave a concrete example analysis to verify the correctness and effectiveness of the improved algorithm. The results illustrate ICOT algorithm can make copy editor get effective consistency maintenance.

Keywords Concurrent control, Collaborative editing, Transformation of a sequence, Operational transformation, Context of operation, Consistency maintenance

1 引言

实时协同编辑,是指多个在地域上分散的用户在同一时间并发地对一共享区域的内容进行编辑。它强调的是人与人之间的交互,对交互过程中的响应性和并行性有较高的要求。由于人为或者网络延迟等无法预料因素的存在,若不对各协同站点的操作加以控制,会导致站点副本的不一致性的产生。对协作编辑中的一致性维护及并发控制,除令牌环机制、锁机制和串行化机制等传统的方法外,目前主流的有基于对象复制的多版本方法和基于操作转换的方法。

Kanawati等在文献[1]中提出了基于多版本技术的共享对象一致性保持算法。该算法通过将冲突操作的效果应用到同一操作对象的不同副本中,将无冲突操作应用到同一对象上,使共享对象保持一致。之后窦万峰等人提出了基于版本复制的分布式多版本模型[2],并设计了MVIC(Multiple Versions Incremental Creation)算法。之后许多的研究人员又在MVIC算法的基础上进行多种改进,如薛良贵针对协同图形编辑中部分操作的可合并关系,通过引入校正操作,定义校正操作的合并,对MVIC算法进行改进,提出基于操作合并的版本复制技术[3]。但是,随着协同用户的增加以及协同工作的逐渐深入,需要站点维系的版本数量增多,这对信息的管理和存储提出了挑战,同时也使得最终版本的选择策略趋于复杂,不利于多用户的实时操作。

操作转换算法是协同编辑中进行并发控制和一致性维护的有效方法,自提出就受到人们的关注。Ellis等在研究群系统的相关理论时最早提出了该类算法[4,5],他提出的dOPT算法(分布式操作转换算法)是一种基于操作语义(插入和删除)和操作参数(在第几个位置插入和删除)的算法,是适用于实时协作系统中的一种乐观并发控制方法。该方法能够很好地解决因果关系和操作意愿维护问题,但由于dOPT构造的转换函数限于特定的应用,存在着不足。在之后的研究中出现了大量的对dOPT的改进算法,如Ressel和Gunzenbauser等提出的adOPTed算法[6,7],就是在dOPT算法基础上,采用了L-转换,并用多维交互图来保证被用来做转换的操作是定义在相同的状态上。adOPTed算法保证了因果关系和操作意愿的一致性维护。但经过研究发现,它并不能完全解决数据的不一致问题。之后Sun和Ellis改进dOPT算法,提出GOT算法,并在REDUCE系统上实现,在进一步的研究中又提出了优化的算法即GOTO算法[8]。GOT算法只是针对简单的并发问题,通过包含转换(IT)和排斥转换(ET)两个转换

本文受山西省科技基础条件平台建设项目(2014091004-0105),山西省高等学校教学改革重点项目(J2013010)资助。
孙 敏(1965-),女,副教授,硕士生导师,主要研究方向为网络安全等,E-mail:minsun@sxu.edu.cn;王瑞花(1989-),女,硕士生,主要研究方向为计算机网络应用。

函数来实现"操作定义时的文档状态与操作执行时的文档状态一致",在保证一致性的同时维护了用户操作意愿,但对偏并发问题则没有给出合理的解决办法。GOTO 算法主要采用位移转换在 GOT 的基础上解决了偏并发问题。Sun 还提出一种基于上下文的操作转换算法——COT 算法[9,10](Context-based Operational Transformation),其通过引入操作的上下文来系统地表述和判断操作之间的关系,为操作副本的一致性维护和 Undo 等问题的解决提供了有效途径。在分布式的协同编辑中,目前认为最有效的,是基于操作转换思想的并发控制算法,其已经应用于如 CodoxWord, CoPPT, Co-Maya, CoVim, CoFlash, CoCKEditor 等协作编辑系统中。COT 算法为协同编辑中的并发控制和一致性维护打下扎实的理论基础。但 COT 算法仍然存在缺陷。在算法的 transform 转换过程中,循环地选取上下文差异集 CD 中的操作,使其经过一定的转换后,将操作的上下文上升到 O 的上下文,然后再将 O 与转换后的操作进行转换。循环结束后,O 的上下文上升到站点的文档状态 DS,然后便可在站点执行。但是,在进行 transform 转换的过程中,存在操作之间的转换重复执行多次的现象,无形之中增加了操作进行转换的次数。本文就是针对 COT 算法的这一不足之处进行改进,提出 ICOT 算法。ICOT 算法能够减少在转换过程中的转换次数,同时解决操作间转换重复的问题,还能够保证协同副本的一致性得到维护。

2 相关理论基础

2.1 基于操作转换思想的操作之间的关系描述[6,7]

1. 操作上下文:是指操作 O 产生时站点的文档状态。

(1)对于一个原始操作,其操作上下文 $C(O)=DS$;

(2)对于转换操作 O',则操作上下文可表示为 $C(O')=C(O)\cup\{org(O_x)\}$,其中 $O'=T(O,O_x)$。

2. 文档状态(DS):

(1)初始时,定义文档的状态 $DS=\{\}$;

(2)当有任意类型的操作 O 执行后,文档状态 $DS'=DS\cup\{org(O)\}$其中 $org(O)$表示 O 的原始操作。

3. 因果关系(→):

给定任意两个分别位于站点 i 和站点 j 的操作 O_a 和 O_b,称 O_a 和 O_b 存在因果关系 $O_a\rightarrow O_b$,当且仅当满足以下的条件之一:

(1)$i=j$,操作 O_a 发生在 O_b 之前;

(2)$i\neq j$,操作 O_a 在站点 j 的执行优先于 O_b 的产生。

4. 上下文依赖/并发关系:

给定两个操作 O_a 和 O_b,如果满足:(1)$O_a\in C(O_b)$;(2)存在一个操作 O_x,有 $O_a\xrightarrow{c}O_x$ 且 $O_x\xrightarrow{c}O_b$,称 O_b 上下文依赖于 O_a,即 $O_a\xrightarrow{c}O_b$。若既不存在 $O_a\xrightarrow{c}O_b$,也不存在 $O_b\xrightarrow{c}O_a$,则称 O_a 和 O_b 是上下文并发的,即 $O_a\parallel O_b$。

5. 上下文等价关系:

给定两个操作 O_a 和 O_b 以及它们的上下文 $C(O_a)$和 $C(O_b)$,则 O_a 和 O_b 的上下文是等价的,当且仅当 $C(O_a)=C(O_b)$。

6. 全序关系"⇒":

任意操作 O_a、O_b 和 O_c 之间存在全序关系,当满足以下条件:(1)若 $O_a\Rightarrow O_b$,$O_b\Rightarrow O_c$,则 $O_a\Rightarrow O_c$,即全序关系具有传递性;(2)只存在 $O_a\Rightarrow O_b$ 或 $O_b\Rightarrow O_a$,即全序关系具有唯一性。

2.2 协同编辑中涉及到的不一致性问题

由于人为或网络延迟等原因,到达站点的操作的执行顺序是任意的,有可能出现副本不一致性的现象。常见的不一致性有:结果不一致、因果关系不一致和操作意愿不一致。

(1)结果不一致。若操作 O_1,O_2,O_3 是不可交换的,且没有对操作的执行顺序进行限制,在操作的执行中,各个站点上副本的结果有可能不一致。图 1 中,假设初始为"A12Ba";O_1=Insert(2,"C"),O_2=Delete(3),O_3=Insert(4,"5")。在站点 1,最终结果为"A12B5a";在站点 2,最终结果为"A1C2B5"。两个站点维护的副本不一致。

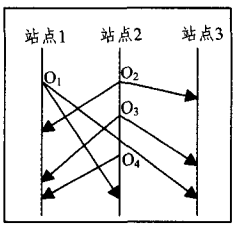

图 1 协同站点间的并发操作

(2)因果关系不一致。由于网络延时,不能保证操作是按照产生时的顺序到达各个站点。这些操作如果不加限制地在各个站点上执行,将会导致副本的不一致。

(3)操作意愿不一致。用户期望的效果与实际操作产生的结果不同。仍以(1)为例,O_2 实际是删除"A12Ba"中第三个位置的"2",但在站点 1,实际删除的是 O_1 所插入的"C",这违背了协作者的意愿。

因此,在协同编辑中,需进行 3 种一致性维护:结果一致性维护、因果关系一致性维护和操作意愿一致性维护。

3 COT 算法及其改进

3.1 基于上下文的操作转换控制算法——COT 算法

操作转换算法的核心是:本地操作立即执行,远程操作需进行转换后才能执行。基于上下文的操作转换(COT)算法,通过引入操作的上下文,对操作的转换与执行进行统一的管理。通过操作上下文来检测和确定操作间的转换和执行方式。在各个站点,都维护着一个文档状态,是一个已执行操作的集合。只有当操作 O 的上下文 $C(O)$与接收站点的文档状态(DS)相等时,O 才能在站点执行。因此,当接收到 O 时,需先对 O 进行检测,查看是否上下文因果就绪(用来保证因果一致性)。若就绪,则查看 O 的上下文与其站点维护的文档状态是否一致,若一致,则 O 可以立即执行;否则需通过一定的转换,将其转换为可执行的形式,然后才能在站点执行。具体的 COT 算法为算法 1 和算法 2[6,7]。

算法 1 COT 算法

```
COT(O,DS)
    EO=transform(O,DS-C(O));
    执行 EO;
    DS=DS∪org(O);
```

算法 2 COT 算法中 transform 转换

```
transform(O,CD)
    循环直到 CD={};
        从 CD 中选择并移出 Ox,满足 C(Ox)⊆C(O);
```

transform(O_x,C(O)−C(O_x));
O=IT(O,O_x);
C(O)=C(O)∪org(O_x);

3.2 COT 算法的改进

在 COT(O,DS)算法的 transform 转换过程中,循环地选取上下文差异集 CD 中的操作,使其经过一定的转换后,将上下文上升到 O 的上下文,然后再将 O 与转换后的操作进行转换。循环结束后,O 的上下文上升到站点的文档状态 DS,然后便可在本站点上执行转换后的操作形式。但是,在进行 transform 转换时,存在操作之间的转换重复多次的现象。以图 1 中的操作执行为例来简单说明这一缺陷。当站点 1 接收到 O_3 时,上下文因果就绪,$C(O_3)=\{O_2\}$,应调用 transform(O_3,$DS-C(O_3)$),而 $CD-C(O_3)=\{O_1\}$,由于 $C(O_3)\neq C(O_1)$,会递归地调用 transform(O_1,$C(O_3)-C(O_1)$),即调用 transform(O_1,$\{O_2\}$),由于 $C(O_2)=C(O_1)$,故有 $O_1'=IT(O_1,O_2)$,此时 $C(O_3)=C(O_1')$,有 $O_3'=IT(O_3,O_1')$,经过转换之后 O_3 变为可执行的形式。当站点 1 接收到 O_4 时,上下文因果就绪,$C(O_4)=\{O_2,O_3\}$,调用 transform(O_4,$DS-C(O_4)$),而 $DS-C(O_4)=\{O_1\}$,由于 $C(O_3)\neq C(O_1)$,会递归调用 transform(O_1,$C(O_4)-C(O_1)$),即 transform(O_1,$\{O_2,O_3\}$),在这个 transform 过程中,会有 $O_1'=IT(O_1,O_2)$ 转换的存在。而这个转换在之前的操作转换时已经存在。这样,对相同的操作进行了重复的转换。针对 COT 算法存在的这一不足,本文进行了改进。在详细地介绍改进算法前先给出 3 个重要定义。

定义 1 假设存在一组操作序列 $S=\{O_1,O_2,\cdots,O_n\}$,若 $C(O_{i+1})=C(O_i)\cup O_i$,$i\in[1,n]$,则称 S 是上下文有序的,且此操作序列的上下文 $C(S)=C(O_1)$。

定义 2 存在两个上下文有序的操作序列 L_1,L_2,若满足条件 $C(L_1[0])=C(L_2[0])$,则称 L_1,L_2 是上下文并发的,即 $L_1\parallel L_2$。

定义 3 存在两个操作序列 L_1,L_2,满足条件 $L_1\parallel L_2$,当且仅当 $C(L_1)\gg L_1=C(L_2)\gg L_2$ 时,L_1 与 L_2 是等价的,即 $L_1\equiv L_2$($C(L_1)\gg L_1$ 表示在文档状态 $C(L_1)$下顺序执行 L_1 中的操作)。

改进后的算法 ICOT,其先决条件是要保存操作转换的最新版本。先获得上下文差异集 CD,明确需要进行转换的操作,然后对 CD 中的操作进行重新排序,再进行分类替换和转换,使得最终的操作序列 CD' 上下文有序,且 $C(CD')=C(O)$;然后再将 O 依次与 CD' 中的操作进行转换,便可将 O 转换为可执行的形式。在整个转换过程中,不仅减少了转换执行的次数,且有效地解决了操作之间转换的重复执行的问题。具体的 ICOT 算法见算法 3。

算法 3 ICOT(O,DS)

第 1 步:根据操作的上下文之间的关系,判断操作 O 是否上下文因果就绪,若就绪,则转第 2 步;否则,等待使得操作上下文因果就绪的条件,并将其保存在等待队列中(一旦因果就绪,则立即对其进行转换)。

第 2 步:判断操作 O 与文档状态 DS 之间的关系,若 C(O)=DS,则 O 可在站点立即执行,并更新站点的文档状态 DS=DS∪org(O),否则转第 3 步。

第 3 步:获得上下文差异集 CD=DS−C(O),对 CD 中的操作先进行重新排序,再进行一定的分类替换和转换,得到一个新的操作序列 CD′,使得 CD′中的操作上下文有序,且

C(CD′)=C(O)

第 4 步:将 O 与 CD′中的操作依次进行 LT 转换,便可将 O 转换为可执行的形式 O′,然后执行 O′,并更新相应的文档状态。具体的转换如算法 4 所示。

算法 4 操作与序列之间的转换

```
LT(O,CD′):
for(i=0;i<CD′. lenght;i++)
    O=IT(O,CD′[i]);
    C(O)=C(O)∪CD′[i];
return O′;
```

ICOT(O,DS)的核心是第 3 步:对 CD 中的操作进行重排序及分类替换和转换。本算法是按操作之间存在的全序关系,先对 CD 中的操作进行排序,并根据实际,将 CD 中的各个操作适当替换为其最近转换版本,使得 CD 中操作的上下文从左到右均是被包含的关系;然后再按操作之间存在的上下文关系,对 CD 中的操作进行分类;分类完成后,对操作进行转换,使得最终的序列上下文有序,且序列的上下文与 O 的上下文相等。具体的转换如下所示。

转换情形 1:若 CD 中只存在一个操作或只存在一组上下文有序的序列 L,且这个操作(或操作序列)与 O 是上下文等价的,则直接转向算法的第 4 步即可;若此操作(或操作序列)的上下文包含于 O 的上下文,则先将此操作(或操作序列)的上下文上升到 O 的上下文,对操作序列的转换采用 LLT 转换方法,然后再转向第 4 步。具体的 LLT 转换如算法 4 所示。

转换情形 2:若 CD 中仅存在多个游离的操作,即任意两个操作都不能简单地组成上下文有序序列,则先将 CD 中的第一个操作转换为与 O 是上下文等价的,然后将 CD 中剩余的操作依次转换为与 CD 中的第一个操作是上下文有序的,然后转向第 4 步。

转换情形 3:若 CD 中是一组上下文有序的操作序列 L 和一组游离的操作的组合,则根据操作的全序关系,对上下文有序序列,根据转换情形 1 中的方法对 L 进行转换;对游离的操作集合,按转换情形 2 中的方法依次将游离的操作进行转换,使得最终得到的序列上下文有序,且上下文有序的序列与 O 是上下文等价的,最后转向第 4 步。

转换情形 4:若 CD 中存在多组上下文有序的操作序列,若操作序列的上下文不相等,则依次对这些上下文有序序列进行转换,直到 CD 的操作最终上下文有序,且序列与 O 上下文等价;若操作序列的上下文是相等的,但与 O 的上下文不等,则先将各个操作序列的上下文上升到 O 的上下文,再进行 LLT 转换或直接将操作序列进行 LLT 转换;然后转向第 4 步。

转换情形 5:若 CD 中的操作是多组上下文有序的操作序列和多个游离操作的组合,则根据操作的全序关系,利用转换情形 3 和转换情形 4 中的方法对 CD 中的操作进行转换。

算法 5 序列之间的转换

```
LLT(L1,L2)
    L1′=L1;L2′=L2;
    for(i=0;i<|L2′|;i++)
        for(j=0;j<|L1′|;j++)
            O2,i=L2′[i];
```

$O_{1,j}=L_1'[j]$;

$(O'_{2,i},O'_{1,j})=SIT(O_{2,i},O_{1,j})$;

$L_1'[i]=O'_{2,i}$;

$L_2'[j]=O'_{1,j}$;

$L_1=L_1+L_2'$;

算法 6 操作之间的对称转换

$SIT(O_1,O_2)$

$O_1'=IT(O_1,O_2)$;

$O_2'=IT(O_2,O_1)$;

return(O_1',O_2');

在 LLT 转换的中，存在两个重要的结论：

结论 1 两个上下文有序且上下文等价的序列 L_1，L_2，经过转换后得到的序列 L_1'，L_2'是上下文有序的序列。

结论 2 两个上下文有序且上下文等价的序列 L_1，L_2，若经过 LLT 转换之后，$L_1=L_1+L_2'$，$L_2=L_2+L_1'$（L_1'，L_2'分别为 L_1，L_2 经过 LLT 转换后的序列），则 L_1，L_2 也是上下文有序的序列，且 $L_1\equiv L_2$。

这两个结论提示我们，在有多个上下文并发的有序序列进行 LLT 转换时，序列之间进行转换的顺序的先后并不影响最终的转换结果，故多个序列以任意的顺序进行转换均可。

3.3 COT 算法实例分析

本文通过一个协作实例，来再现在协作中通过 ICOT 算法控制操作之间的具体转换执行过程。假设存在 3 个协同站点和 8 个协同操作，初始为空，在各个站点的执行顺序如图 2 所示。

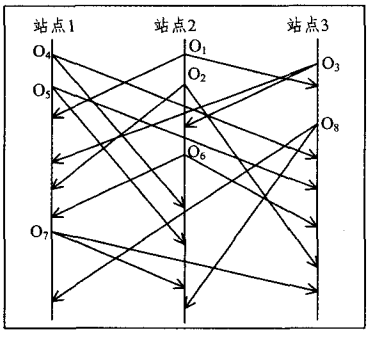

图 2 在各站点的协同操作

仅以站点 3 为例给出 ICOT 算法的具体应用。

首先，确定协作中操作的上下文：$C(O_1)=C(O_3)=C(O_4)=\{\}$，$C(O_2)=\{O_1\}$，$C(O_5)=\{O_4\}$，$C(O_6)=\{O_1,O_2,O_3\}$，$C(O_7)=\{O_4,O_5,O_1,O_3,O_2\}$，$C(O_8)=\{O_1,O_3\}$；且操作之间存在全序关系：$O_1\Rightarrow O_2\Rightarrow O_3\Rightarrow O_4\Rightarrow O_5\Rightarrow O_6\Rightarrow O_7\Rightarrow O_8$。

在站点 3，执行顺序为$[O_3,O_1,O_8,O_4,O_5,O_2,O_6,O_7]$。

O_3 是本地站点产生的操作，可立即执行，文档状态更新为 $DS=\{O_3\}$。

当 O_1 到达站点时，上下文因果就绪，且 $C(O_3)\neq DS$，故 $CD=DS-C(O_1)=\{O_3\}=CD'$，满足转换情形 1，且 $C(O_3)=C(O_1)$，因此，O_1 与 O_3 进行转换，便可将其转换为可执行的形式：O_1'，$C(O_1')=\{O_3\}$。执行完后，更新站点的文档状态 $DS=\{O_3,O_1\}$。

O_8 立即执行，文档状态更新为 $DS=\{O_3,O_1,O_8\}$。

当 O_4 到达站点时，上下文因果就绪，且 $C(O_4)\neq DS$，故 $CD=\{O_3,O_1,O_8\}$，重排序后 $CD'=[O_1,O_3,O_8]$，满足转换情形 5，且 $C(O_1)=C(O_3)$。因此，O_1 与 O_3 进行 LLT 转换，得到上下文有序序列$[O_1,O_3']$，之后和 O_8 组合得到序列$[O_1,O_3',O_8]$，是上下文有序的，且与 O_4 的上下文等价，故 O_4 依次与其中的操作进行转换，转换为可执行的形式：O_4'，$C(O_4')=\{O_3,O_1,O_8\}$。执行完后，更新站点的文档状态 $DS=\{O_3,O_1,O_8,O_4\}$。

当 O_5 到达站点时，上下文因果就绪，且 $C(O_5)\neq DS$，故 $CD=\{O_3,O_1,O_8\}$，重排序后 $CD'=[O_1,O_3,O_8]$，进行适当的替换后有 $CD'=[O_1,O_3',O_8]$。满足转换情形 1，但 $C(O_5)\neq C(O_1)$，应先将 CD'的上下文上升到 $C(O_5)$，得到 CD''。最后，O_5 与 OD''中的操作依次进行转换，即可转换为可执行的形式：O_5'，且$(O_5')=\{O_3,O_1,O_8,O_4\}$，执行后更新站点的文档状态 $DS=\{O_3,O_1,O_8,O_4,O_5\}$。

当 O_6 到达站点时，不满足上下文因果就绪，需等待。

当 O_2 到达站点时，上下文因果就绪，$C(O_2)\neq DS$，故 $CD=\{O_3,O_8,O_4,O_5\}$，$CD'=[O_3,O_4,O_5,O_8]$，发现 O_8 是游离的操作，$[O_4,O_5]$上下文有序，且 $C(O_4)=C(O_3)$，满足转换情形 5，故先将 O_3 与$[O_4,O_5]$进行 LLT 转换，得到上下文有序序列 $L=[O_3,O_4'',O_5'']$；之后再将 $C(O_8)$的上下文上升到 $\{O_3,O_1,O_4,O_5\}$，L 与 O_8'组合之后就得到整体有序的操作序列$[O_3,O_4',O_5',O_8']$。将 O_2 依次与$[O_3,O_4',O_5',O_8']$中的操作转换，便可转换为可执行的形式：O_2'，$C(O_2')=\{O_3,O_1,O_8,O_4,O_5\}$。执行完后 $DS=\{O_3,O_1,O_8,O_4,O_5,O_2\}$。

执行完 O_2 后，处在等待队列中的 O_6 上下文因果就绪，$C(O_6)\neq DS$。$CD=\{O_8,O_4,O_5\}$，由于 $CD'=[O_4,O_5,O_8]$，经过适当替换变为 $CD'=[O_4',O_5',O_8']$，上下文有序，满足转换情形 1，应先将 L 的上下文上升到 $C(O_6)$。然后 O_6 与转换后的 L 中的操作依次进行包含转换，便可将其转换为可执行的形式：O_6'。然后执行 O_6'，并更新站点的文档状态 $DS=\{O_3,O_1,O_8,O_4,O_5,O_2,O_6\}$。

当 O_7 到达站点时，上下文因果就绪，$C(O_7)\neq DS$，故 $CD=\{O_8,O_6\}$。$CD'=[O_6,O_8]$是游离的操作，满足转换情形 2，应逐个将 CD 中的操作进行转换，$C(O_6')=\{O_4,O_5,O_1,O_3,O_2\}$，$C(O_8')=\{O_4,O_5,O_1,O_3,O_2,O_6\}$。转换完成后，$O_7$ 就可以依次与$[O_6',O_8']$进行包含转换，得到其可执行的形式：O_7'。执行完后，更新站点的文档状态 $DS=\{O_3,O_1,O_8,O_4,O_5,O_2,O_6,O_7\}$。

4 ICOT 算法的正确性与一致性维护分析

4.1 基于操作转换的算法中操作的正确执行和转换

在 ICOT 算法中，一个远程操作能够正确执行，需满足其定义上下文（操作产生时的文档状态）和执行上下文（操作执行时的文档状态）是相等的；任意两个操作之间能够正确进行转换，需满足生成两个操作的站点的文档状态是一致的。为保证改进后算法中并发操作能够正确执行和转换[8]，ICOT 算法应该满足以下 4 个转换执行条件。

CC1：给定一个操作 O 和文档状态 DS，$O\notin DS$，只有当 $C(O)\subseteq DS$ 时，O 才可以在 DS 上转换执行。

CC2：给定一个操作 O 和文档状态 DS，$O\notin DS$ 且$C(O)\subseteq DS$，则 $DS-C(O)$是一组操作，当 O 在 DS 上执行之前必须先与之进行转换。

CC3:给定一个操作 O 和文档状态 DS,只用当 $C(O)=DS$ 时,O 才能在 DS 上执行。

CC4:给定两个操作 O_x 和 O,O_x 和 O 能够进行 IT 转换,当且仅当 $C(O)=C(O_x)$。

其中 CC1 确定了操作 O 在文档状态上进行转换和执行的条件,CC2 明确了 O 需要与哪些操作进行转换,CC3 确定了能够在站点上执行需满足的条件,CC4 确定了两个操作之间能进行 IT 转换的条件。在 ICOT 算法中,当远程站点的操作到达时,要先进行检测,查看操作是否已因果就绪。若满足,查看其上下文与站点维护的文档状态是否一致,若不一致,根据 CC1 和 CC2,将 O 与上下文差异集进行转换;当进行 LLT 转换、SIT 转换和 LT 转换时,操作之间的转换满足 CC4,当 LT 转换执行完成之后,将会满足 CC3,远程操作可以被执行。通过分析 ICOT 中操作的转换过程,可知 ICOT 算法满足上述 4 个转换条件。

4.2 ICOT 算法的一致性维护

在基于操作转换的并发控制算法中,存在两种方法来保证收敛[11],使最终副本得到一致性维护。一是设计一个有效合理的底层转换函数,使之能够保持转换属性 CP1 和 CP2,二是设计高层控制算法来避免 CP1 和 CP2[12]。目前,在基于操作转换思想进行并发控制和一致性维护方面,最有效且常用的方法是:设计一个合适的转换函数来保证 CP1,同时设计控制算法来避免 CP2,或者破坏掉 CP2 的前提条件[9,13]。本文改进的 ICOT 算法也是从这两方面来进行副本一致性维护的有效性验证。

1. 收敛属性 CP1

假设存在操作 O_1,O_2,如果 $C(O_1)=C(O_2)=DS$,如果 $O_1'=IT(O_1,O_2)$,$O_2'=IT(O_2,O_1)$,则 $DS\gg O_2\gg O_1'=DS\gg O_1\gg O_2'$(符号"≫"表示操作的顺序执行)。

2. 收敛属性 CP2

假设存在操作 O_1,O_2,O_3,$C(O_1)=C(O_2)=C(O_3)=DS$,如果 $O_2'=IT(O_2,O_3)$,$O_3'=IT(O_3,O_2)$,则 $IT(IT(O_1,O_2),IT(O_3,O_2))=IT(IT(O_1,O_3),IT(O_2,O_3))$。

3. CP2 的前提条件(pre-CP2)

CP2 是转换函数需要保持的属性,当且仅当基于操作转换思想的系统允许两个上下文相等的操作 O_x,O_y 能够在不同的上下文中进行包含转换,即 $IT(O_x,O_y)$和 $IT(O_x',O_y')$。

在 ICOT 算法中,只有在 LLT 转换、SIT 转换和 LT 转换中涉及到操作转换且只是包含转换。根据包含转换的定义,$O_1'=IT(O_1,O_2)$,转换结果 O_1'将包含已经执行的操作 O_2 的作用效果,即通过 IT 转换后,副本能够保持 O_1 的操作意图,亦即在相同的文档状态下,以不同的顺序执行两个上下文等价的操作不会对操作的结果造成影响,不会出现副本不一致的情况。故转换函数能保持 CP1 属性。

因此,只要证明 ICOT 算法能破坏掉 CP2 的前提条件,即 ICOT 算法能够保证一对操作总是在相同的上下文条件下进行转换,则 ICOT 算法就能使得操作副本得到一致性维护。下面给出具体的证明:

在 ICOT 算法中,进行转换之前,对 CD 中的操作利用操作之间存在的全序关系进行重新排序,便能够保证对 CD 中的任意两个操作 O_a 和 O_b,若 $O_a\Rightarrow O_b$,则 O_a 在 O_b 之前被选中与远程操作进行转换。这样,就能保证任意一个远程操作进行转换和执行的顺序是一致的,即能够保证任意一对操作总是在相同的上下文条件下进行转换。故 ICOT 能破坏掉 CP2 的前提条件。

由文献[15]可知,当 CP2 的前提条件被破坏时,CP1 就可作为唯一的条件来保证算法的正确性,故改进后的算法是正确的。

结束语 本文针对在协同编辑中遇到的不一致性问题进行了详细的阐述,并在此基础上,对基于上下文的操作转换算法 COT 进行了改进,提出了可进行一致性维护的并发控制算法 ICOT。ICOT 算法解决了 COT 算法中存在的操作之间转换重复执行的不足,能很好地解决在协同编辑中的一致性维护的问题。但是本文并没有设计出具体的操作之间的包含转换函数,这将是下一步工作研究的重点。

参 考 文 献

[1] Kanawati R. A replicated-data management algorithm for distributed synchronous groupware application [J]. Parallel computing, 1997, 22(13): 1733-1746

[2] 杨君. 窦万峰. 一种新的多版本增创算法[J] 计算机学报, 2008(4): 702-710

[3] 薛良贵. 协同图形编辑系统中操作合并的研究[D]. 广州: 华南理工大学, 2010

[4] Ellis C A, Gibbs S J. Concurrency control in groupware systems [C]// ACM SIGMOD. 1989: 399-407

[5] 许坚. 蒋晓峰. 基于图形对象的一致性维护问题的研究[J]. 计算机应用与软件, 2012(2): 261-265

[6] Ressel M, Nitsch-Ruhlmrd D, Gunzenbauser R. An integrating, transformation-oriented approach to concurrency control and undo in group editors [C]// ACM Conf. Computer Supported Cooperative Work' 96. Boston, 1996

[7] 廖斌, 何发智, 荆树旭. 实时协同工作系统中的操作转换算法综述[J]. 计算机研究与发展, 2007, 44(2): 326-333

[8] Sun D, Sun C. Context-based operational transformation in distributed collaborative editing systems[J]. IEEE TPDS, 2009, 20(10): 1454-1470

[9] Sun D, Sun C. Operation Context and Context-Based Operational Transformation [C] // Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW'06). 2006(10): 279-288

[10] Sun C. OTFAQ: operational transformation frequent asked questions[OL]. http://cooffice. ntu. edu. sg/otfaq

[11] Sun C, Wen H, Fan H. Operational transformation for orthogonal conflict resolution in collaborative two-dimensional document editing systems[C]// ACM CSCW. 2012: 1391-1400

[12] Sun A C, Xu D. Operational transformation for dependency conflict resolution in real-time collaborative 3D design systems [C]// Proc. ACM CSCW. 2012: 1401-1410

[13] Agustina, Sun C. Dependency-conflict detection in real-time collaborative 3D design systems[C]// ACM CSCW. 2013: 715-728

[14] Xu Y, Sun C, Li M. Achieving convergence in operational transformation: conditions, mechanisms and systems [C] // ACM CSCW. 2014