一种集成组播代理和操作转换的并发控制方法*

杨武勇+, 史美林, 姜进磊

(清华大学 计算机科学与技术系,北京 100084)

An Operation Transformation Based Concurrency Control Integrating Multicast Agent

YANG Wu-Yong+, SHI Mei-Lin, JIANG Jin-Lei

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

+ Corresponding author: Phn: +86-10-62785609, E-mail: ywyong@csnet4.cs.tsinghua.edu.cn, http://www.cs.tsinghua.edu.cn

Received 2003-04-10; Accepted 2003-09-05

Yang WY, Shi ML, Jiang JL. An operation transformation based concurrency control integrating multicast Agent. *Journal of Software*, 2004,15(4):497~503.

http://www.jos.org.cn/1000-9825/15/497.htm

Abstract: Existing distributed real-time cooperative systems mainly use operation transformation technique to provide concurrency control service. But the performance of a run-time system will be degraded when the system produces a large amount of collaborative data. A novel concurrency control framework named madOPT based on collaborative data objects is then proposed in this paper. madOPT resolves conflicts by utilizing operation semantics of the accessed data objects. In addition, it integrates a data transmission framework with an operation transformation method and allows the running system to load the data objects dynamically. The enhancement of madOPT over dOPT (distributed operation transformation) is that it reduces the number of queries in operation log during an operation transformation. It takes object attributes as the granularity of concurrency control and thus can support graph and image objects. It also adopts multicast agent to improve the efficiency of data distribution as well as the performance of the system.

Key words: computer supported cooperative work; reliable multicast; concurrency control; fully-replicated architecture; dOPT (distributed operation transformation)

摘 要: 现有的分布式实时协作系统多采用操作转换的方法来提供并发控制服务,但是在系统数据量很大时,系统性能不高.为解决这一问题,提出了一种新的并发控制方法 madOPT:该方法利用对象所定义的操作的语义进行冲突解析,集数据对象的动态载入、数据传输和操作转换功能为一体.madOPT 改进了 dOPT(distributed

* Supported by the National Natural Science Foundation of China under Grant No.60073011 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2001AA113150 (国家高技术研究发展计划(863)); the National High-Tech Research of Beijing Municipal Science & Technology Commission of China under Grant No.199950 (北京市科委高科技研究项目)

作者简介: **杨武勇**(1973－),男,云南怒江人,博士,主要研究领域为计算机支持的协同工作;**史美林**(1938－),男,教授,博士生导师,主要研究领域为计算机网络及其应用,计算机支持的协同工作;**姜进磊**(1975－),男,博士后,主要研究领域为计算机支持的协同工作.

operation transformation)算法在操作转换过程中对操作日志的遍历以提高并发控制效率,并将对象属性作为并发控制的粒度,使之支持对图形、图像对象的并发操作;同时结合组播代理,提高数据传输率,使系统整体性能得到改善.

关键词: 计算机支持的协同工作;可靠组播;并发控制;全复制结构;dOPT(distributed operation transformation)

中图法分类号: TP311 **文献标识码**: A

在 CSCW(computer supported cooperative work)系统中实现支持人机及协作者之间的自然、自由交互的并发控制方法一直是实时群件系统的研究热点.为了提高系统的响应速度及可靠性,一般都采用全复制式结构,将协作过程中产生和操作的对象分别对等地复制在各协作参与者处[1].Greenberg 等人将全复制结构下的并发控制方法分成"乐观"与"悲观"两类[2].总体而言,悲观的方法可能会导致滞涩、受限的用户界面,而乐观的方法大都会导致用户界面不自然地突变.这些方法距离"自然、自由交互"的目标仍然有相当大的差距.

在现有的并发控制算法中,Ellis 等人提出的基于操作转换方法 dOPT(distributed operation transformation)[3]极有希望实现"自然、自由交互"的目标.它通过立即执行用户的本地操作以保证用户界面良好的响应速度;通过"状态向量"来保证操作之间的因果依赖关系[4];通过操作转换函数以保证对象的一致性及满足用户操作意图.在执行一个操作 c 之前,dOPT 依次用操作历史记录中与 c 没有因果关系的操作(并发操作)按预定义的转换函数对 c 进行转换.这种策略在"偏并发"的情况下(即 c 与用来对它进行转换的并发操作并不是针对同一对象状态定义的)将会失败.为弥补这一缺陷,Suleiman 等人借助于"前向和后向转换"重排历史记录中的操作[4],Reseel 等人借助于"交互模型"作为历史记录,以保证被用来作转换的一对操作是定义在相同状态上的[5],Sun 等人通过"包含与排除变换"[6,7]而使得用于对 c 进行转换的操作序列具有与 c 相同的上下文.这些改进方法具有不同的时间与空间复杂度,但基本思想却是相同的,即满足转换函数所要求的作为其参数的两个操作必须是定义在同一对象状态上这一前提条件.

上述基于操作转换并发控制的问题研究均是在实时协同字符串编辑器的背景下进行的.实时协作系统(如电子白板、协同编著系统等)中并发操作不仅限于字符串,还包括图形、图像等协作数据.实时协作系统中的所有共享的协作信息都可以抽象为协作对象,它们是协作的基础.在协作系统具体实现过程中,提高自然交互的并发控制服务能力,不仅需要提高并发控制算法的并发控制粒度和能力,还需要提高数据传输效率.目前实时协作系统一般都采用应用层组播来实现可靠组播传输.我们将数据传输框架和并发控制服务结合起来,引入 Agent 技术来实现应用层组播,即组播代理(multicast Agent,简称 MA),利用数据传输的协作服务器进行数据传输和差错恢复,以及利用分层存储协作对象的操作日志以减少并发控制服务过程中的查找次数,提高系统性能.所以,我们在 dOPT 等操作转换算法的基础上,提出一种基于对象操作语义的全复制结构下的并发控制方法 madOPT.madOPT 的命名是从组播代理和操作转换组合而来的.

1 并发控制机制

基于操作转换的并发控制算法所处理的操作的粒度及对象的层次均是固定不变的[3,4,6].但实际上多用户的并发操作经常发生在不同的层次上,例如:段落、句子、单词、字符等等.当并发操作被分解到不同的层次上时,操作之间的相互依赖将被减弱,从而实现减少约束、提高并发性、达到更加自由交互的目的.为提高并发控制算法的效率和可扩展性,我们以协作数据对象的逻辑结构及其操作语义为基础,将所有协作数据作为对象处理,对数据对象分别定义其所有的属性及其在该属性上的操作语义,构造相应的转换函数,使并发控制的粒度层次定义到数据对象的属性这一层次.为方便讨论,我们定义了下面一组并发控制过程中必须的定义,包括对象操作结果、并发操作冲突、站点状态、操作请求、操作历史记录等.

定义 1. 对对象操作的执行结果的定义.当在协作对象 o 上执行某个操作 c 时,它可能会产生两种不同类型的结果:

(1) 给调用者返回某个对象.用 $o.c$ 表示所返回的对象.

(2) 改变了 o 的状态.用 $o{:}c$ 表示修改之后的对象[8].

定义 2. 并发操作冲突的定义.对于由实时协作系统中两个不同站点所生成的两个并发操作 c_1 和 c_2,在任一站点处,当按不同的次序依次执行它们时,若 3 个条件:① $(o{:}c_1).c_2 \neq o.c_2$,② $(o{:}c_2).c_1 \neq o.c_1$,③ $(o{:}c_1){:}c_2 \neq (o{:}c_2){:}c_1$ 中至少有一个被满足,则称操作 c_1 和 c_2 是冲突的[8],记为 $c_1 \times c_2$;否则,c_1,c_2 不冲突则称为 c_1,c_2 相容,记为 $c_1 \| c_2$.只有在同一对象上的操作才有可能发生冲突.

定义 3. 站点执行操作的状态向量 $V_i.V_i=\langle v_i^1,v_i^2,\ldots,v_i^n\rangle$,$v_i^j$ 表示站点 S_i 已经执行过的由站点 S_j 产生的操作个数.

定义 4. 两个状态向量之间的领先关系:V_i 领先于 V_j 当且仅当 $\forall k, v_i^k \geq v_j^k$,记为 $V_i \geq V_j$.

定义 5. 操作请求向量 $R.R=\langle s,t,o,n,v,p,V_s\rangle$,$s$ 表示生成包含此操作的操作请求的站点的标识,t 表示操作类型(引用 R、更新 U、插入 I、删除 D、Undo、Redo 等 6 种操作),o 表示被操作的协作对象的标识,n 表示被操作的属性在协作对象中定义的序号,v 表示此操作所设置的值,p 表示与 R 操作请求相关联的优先数,V_s 表示站点 s 在执行 R 之前的状态向量.

定义 6. 操作日志 $L_i.L_i=\{\langle R_i^j,r\rangle | j$ 为正整数$\}$,L_i 表示站点 S_i 的操作日志,R_i^j 表示站点 S_i 接收到的第 j 个操作请求向量,r 表示 R_i^j 操作请求在站点 S_i 的执行结果类型,有两种:正常修改 NORMAL_UPDATE、伪修改 PSEUDO_UPDATE(是指因为并发操作冲突而实际上没有被执行的操作请求).

下面介绍 madOPT 的工作原理以及在 dOPT 算法基础上进行的改进.协作用户的每一个操作都可以细化为协作数据对象 o 的属性 i 的方法调用 m_i 或一组方法调用.每个站点维护一个状态向量 V.对于某个操作请求向量 R,只有当其所附带的状态向量 V_R 和本地状态向量 V 满足 $V \geq V_R$ 时,控制算法才会调度 R 去执行,为保证复制对象的一致及各操作在各站点的执行结果的一致,任意 R 在被任一站点实际执行之前,将用被操作的数据对象的操作语义函数与历史记录中的那些操作进行解析转换,即站点 S_1 和 S_2 在对象 o 上执行 c_1,c_2 并发操作,站点 S_1 上 o 的结果为 $o{:}c_1$,站点 S_2 上 o 的结果为 $o{:}c_2$,为了保证数据一致性,必须通过转换函数使 $(o{:}c_1){:}c_2'=(o{:}c_2){:}c_1'$,以满足用户操作意愿的一致,其中 c_2' 为 c_2 在站点 S_1 上通过转换函数转换之后的操作,c_1' 为 c_1 在站点 S_2 上通过转换函数转换之后的操作.例如现有字符串“concurrence”,站点 S_1 准备在第 3 个字符后插入一个字符‘a’,站点 S_2 将删除第 7 个字符‘r’,c_1=Insert(o,3,a),c_2=Delete(o,7),所以在 S_1 上通过转换函数转换后的 c_2'=Delete(o,8),S_2 上的 $c_1'=c_1$.当解析后的操作被执行之后,它被加入到操作日志中,同时,状态向量的相应分量将被加 1.

由定义 1 和 dOPT 算法的分析可以推出两个操作 c_1,c_2 冲突 $c_1 \times c_2$ 的另一种形式,有如下定理:

定理 1. 任意一个协作对象 o 在任何一个站点上都有它的一份复制;站点 S_1 产生操作 c_1,其操作请求向量为 R_1,V_{S_1} 为 R_1 附带的状态向量,站点 S_2 产生操作 c_2,其操作请求向量为 R_2,V_{S_2} 为 R_2 附带的状态向量;c_1,c_2 操作对象均为同一个协作对象 o;站点 S 接收 R_1 并执行 R_1 后站点 S 的操作日志中对应该操作的记录中的状态向量为 V_{S_1};当站点 S 接收到 R_2 后,此时有:$c_1 \times c_2$ 必有 $v_{S_1}^{S_1} \geq v_{S_2}^{S_1}$ (站点 S_1 和 S 可为同一个站点.)

证明(反证法):

假设在 $c_1 \times c_2$ 时 $v_{S_1}^{S_1} \geq v_{S_2}^{S_1}$ 不成立,即 $v_{S_1}^{S_1} < v_{S_2}^{S_1}$;

因为状态向量中的第 S_1 个分量是各站点执行站点 S_1 产生的操作的个数,$v_{S_2}^{S_1}$ 为站点 S_2 执行 c_2 前站点 S_2 执行站点 S_1 产生的操作的个数,$v_{S_1}^{S_1}$ 为站点 S_1 执行 c_1 前站点 S_1 执行它自己产生的操作的个数;

由 $v_{S_1}^{S_1} < v_{S_2}^{S_1}$ 说明,站点 S_2 执行站点 S_1 产生的操作的个数比站点 S_1 执行它自己产生的操作个数要多,而基于操作转换的并发控制算法是立即响应本地操作,$v_{S_1}^{S_1} < v_{S_2}^{S_1}$ 必然不成立;

所以,与假设矛盾. □

由定理 1 可知,在并发控制算法中,对于 $c_1 \times c_2$ 的操作,仅需要对同一个数据对象的同一个属性的操作日志中状态向量分量大的记录进行比较和转换,然后调用数据对象的操作方法,即某数据对象的操作日志 L 中的记录为 $\langle s_l,t_l,o_l,n_l,v_l,p_l,V_l,r_l\rangle$,当前调度的操作请求向量 $R=\langle s_R,t_R,o_R,n_R,v_R,p_R,V_R,r_R\rangle$,只需要比较 L 中的 $v_l^{s_l} \geq v_R^{s_l}$ 记录,然后进行转换.从上述过程的描述中可以得出下面的推论:

推论 1. 在 dOPT 算法中,两个操作 c_1,c_2 相容或冲突,在进行操作转换时,仅需要比较操作日志中最新的部分

记录,而不必遍历操作日志.

根据推论 1,我们把数据传输和并发控制功能结合起来,数据传输服务器收集每一个站点执行操作 c 的信息,通过核心服务器(将在数据传输部分中介绍)将操作 c 已执行完毕的消息广播给所有站点,各个站点接收到该消息后,在操作日志记录中删除操作 c 的历史记录,这样,操作日志中始终只保持最新的操作记录,使冲突解析时需要查找的时间大大减少,提高了系统的运行效率.

在 dOPT 等操作转换算法的基础上,我们对并发控制算法在并发控制的粒度和数据对象层次以及并发控制服务的可扩展性等方面还进行了改进.根据协作数据对象的静态和动态两方面的特性,将用户的操作转化为定义在协作数据对象的数据属性上的方法调用,使协作数据对象的方法调用作为其操作粒度和操作对象的层次,通过移动 Agent 技术,可以动态地将协作数据对象及其属性和操作方法的定义加载到系统中.一般地,协作系统会预先定义出基本数据对象以及这些对象的属性和方法,如果有新增的协作对象,只需将该对象定义成 Cova[9] 类,或者将已有的简单操作组合成复合操作,定义其复合操作的操作语义并编写为 Cova 代码,同时通过组播代理将该代码发布给所有协作用户,使系统支持对该数据对象的并发控制服务,从而提高了系统并发控制服务的可扩展性.

2 数据传输

我们把能够支持 IP 组播的局部范围称为组播域(multicast domain),如局域网等,各个组播域之间通过单播协议(如 TCP,UDP)进行数据传送,这样就在 Internet 上实现了多点数据通信.参与协作应用的协作用户构成一个树形结构,其中的叶子节点为协作用户(CoUser),中间节点称为协作服务器(CoServer),根节点为核心服务器(CoreServer),也称为中心站点,如图 1 所示.

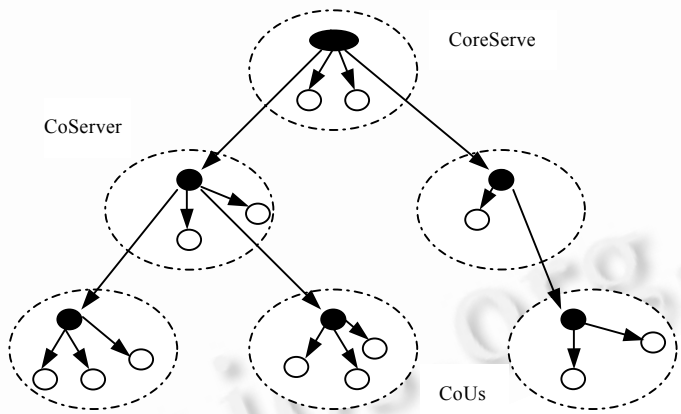

Fig.1 Data transmission framework

图 1 数据传输框架

协作系统的一个应用(例如电子白板的一个主题讨论、协同编著系统中的对一个文档的协同编辑等)只有一个核心服务器,每一个组播域中有且仅有一个协作服务器.服务器之间形成一棵以核心服务器为根节点的共享组播树.当一个协作用户要发送协作数据时,首先把数据用可靠的单播(如 TCP)发送到位于树形结构根节点的核心服务器,核心服务器在给数据加上附加信息(如全局序列号、时戳等)之后沿共享组播树中与之相邻的子节点发送,其他协作服务器接收到协作数据之后,将该数据沿共享组播树依次传送给下一个协作服务器,并在本地组播域中组播发送给本地所有协作用户.

我们把组播传输功能以代理(Agent)的形式嵌入到每一个系统客户端程序中,所以我们也称其为组播代理.组播代理分为协作服务器和协作用户两种角色.每个客户端在启动应用时,同时启动组播代理.由于协作系统中存在着协作用户的加入和退出应用的随意性,如果在每一个加入应用的组播域中预先设置协作服务器,那么明显地会造成资源浪费,同时也是不现实的;在组播域中每个组播代理在启动时以自组织的方式动态地推举出其中一个充当协作服务器的角色,这样也提高了资源利用率和系统的可伸缩性.当组播代理的角色确定之后,协作

用户的组播代理向本地协作服务器注册登记和进行数据的组播传输;协作服务器向核心服务器登记注册并获得共享组播树的拓扑结构,获取和共享组播树中的各叶子节点间的网络链路状况信息(主要通过 TTL 测试等手段),然后根据其值加入共享组播树中,在数据传输过程中,协作服务器不仅负责本地组播域中的组播数据的传输,还要与共享组播树中的相邻节点进行交互和转发数据,同时还提供本地组播域及其子树上的组播域中的数据恢复功能.

在实时协作系统中,音频、视频等媒体应用对数据的实时性要求较高,而电子白板等系统要求数据全局一致,对数据传输可靠性要求较高,对于数据传输的抖动不太敏感;每一个应用组成一棵共享组播树,而对于单个应用来讲,其用户数不会很大,那么其共享组播树的层次相应比较少,这样协作数据从数据源到各个叶子节点所转发的跳数也较少,协作服务器之间通过层次模型,保证了在最短时间内逐级地进行数据传输和差错恢复,这样虽然采用组播代理实现数据传输以及并发控制中状态向量的传输增加了一定的通信量,考虑到应用的数据流量不大,在保证数据的可靠传输的前提下,系统运行时,其数据传输的实时性还是可以接受的.

核心服务器定期发出 NOTICE 消息,NOTICE 消息中包含已发出的最高序列号,协作服务器和协作用户可以根据序列号的缺口发现丢失的数据.如果协作用户发现有数据丢失,就向本地组播代理发出 NACK,请求数据恢复,本地组播代理接收到该请求后,首先在自己的缓冲区中查找,若存在相应的数据,则进行本地重传,否则这个请求将沿着共享组播树中的上一级协作服务器转发,依此类推,直到数据被恢复.这样,每个协作服务器只需处理本地组播域中的协作用户和其子节点的协作服务器的重传请求,分散了处理负担.在构造共享组播树时,通过限制一个协作服务器的直接子节点的个数,可以使协作服务器的负担趋于一个常数,而不会随整个应用的成员数量的增加而增加,解决了确认内爆问题,提高了系统鲁棒性.

3 调度控制算法

madOPT 算法在 dOPT 的基础上作出一些改进,在维护数据对象及其操作日志的方式上,本文的算法可以维护多个数据对象,并为每一个数据对象维护一个独立的操作日志,同时还将数据传输中的层次结构和并发控制服务有机地结合起来,在本地协作服务器中为每一个数据对象保存完整的操作日志,而对于协作用户,仅为每一个数据对象的每一个属性保存几条最新的操作日志记录,当一个协作用户处理完一条操作请求 R 时,向本地协作服务器报告 R 已执行的消息,协作服务器接收到本地所有协作用户及其子树上的协作服务器的关于该 R 已执行的消息后,汇总向上一级协作服务器报告,直到核心服务器为止,然后由核心服务器沿共享组播树广播,协作用户接收到该消息后,从操作日志中删除关于R的记录,若此时操作日志为空,则保留最后一条记录,保证了不会因为操作日志的记录不全而导致后续的操作解析出错.对 R,U,I 和 D 等操作的调度控制算法具体如下:

每个协作站点维护一个本地状态向量 V、每个协作数据对象的操作日志 L 和操作请求队列 Q,其初始状态均为空集,即 $V=\langle 0,0,\ldots,0\rangle$,$L=\varnothing$,$Q=\varnothing$.

① 站点 s 的本地操作 op,op 在站点 s 立即执行,并生成操作请求向量 $R_s=\langle s_R,t_R,o_R,n_R,v_R,p_R,V_R\rangle$,修改 V 中相应分量(状态向量的生成与 dOPT 相同),r=NORMAL_UPDATE,将$\langle R_s,r\rangle$插入到 L 尾部,并向核心服务器发送 R_s;

② 核心服务器接收到 R_s 后,根据站点 s 的用户级别和接收时的时戳,封装 R_s 向量中的优先数 p 和全局序列号,然后沿共享组播树广播;

③ 其他协作站点接收到 R_s 时,通过序列号来判断数据传输的一致性,若发现数据丢失,则向本地或者上一级协作服务器发出数据恢复请求,并将 R_s 放入操作请求队列 Q 中;

④ 当站点 s_1 调度执行 R_s 时,首先比较 V_R 和本地状态向量 V,若 $V^s\geq V_R^s$,将 R_s 从 Q 中删除并转⑤,否则说明有在产生 R_s 之前的操作请求向量由于网络拥塞而延迟传输或丢失,将 R_s 保留在 Q 中,等待下一次调度;

⑤ 若 t_R=U(其他操作类型 R,I,D 等情况类似),将协作数据对象 O_R 的操作日志 L 中的关于 O_R 的第 n_R 个属性的操作记录中的 V_L 和 V_R 比较:

(a) 若 $V_L==V_R$,且 $s_R==s_L$,此 R_s 为重复操作请求,若 $s_R=s_1$,则 $p_L=p_R$,转⑦;

(b) 若 $V_R^s>V_L^s$,或者 $V_L^s\geq V_R^s$,且 $p_R\geq p_L$,则用调用 O_R 的第 n_R 个属性的方法调用 m_r,将 O_R 的第 n_R 个属性赋值为 v_R,修改 V 中相应分量:$V^s=V^s+1$;r=NORMAL_UPDATE,将$\langle R_s,r\rangle$添加到 L 尾部;

(c) 若 $V_L^s \geq V_R^s$,且 $p_R < p_L$,r=PSEUDO_UPDATE,若本站点为协作服务器,将$\langle R_s, r\rangle$添加到 L 尾部,修改 V 中相应分量:$V^s=V^s+1$;

⑥ 向本地协作服务器发送该操作结束消息,协作服务器接收完其子树和协作用户的操作结束消息后向上层提交,直到核心服务器为止,核心服务器汇总后再广播该操作已结束消息;

⑦ 本次调度算法结束.

Undo,Redo 等操作需要完整的操作日志信息以及维护协作用户的操作意愿,以保持数据的全局一致性,对此,我们通过协作服务器来处理:当协作服务器处理 Undo,Redo 等操作类型的操作向量请求 R 时,将遍历该数据对象的操作日志 L,从 L 中找到对应 R 要 Undo 或 Redo 操作的操作日志记录,比较其中的状态向量 V_L、优先数 p_L 和执行结果类型 r_L,若 r_L=PSEUDO_UPDATE,说明该操作结果为伪修改操作,是在本地由于并发操作冲突而实际上没有被执行的操作,将 r=PSEUDO_UPDATE 和 R 加入 L 中,反之,r_L=NORMAL_UPDATE,从 L 中找到 R 的关于 O_R 的第 n_R 个属性的上一次操作的记录,修改 O_R 的第 n_R 个属性的值为上一次操作的值,将 r=NORMAL_UPDATE 和 R 加入操作日志中,修改本地状态向量 V,并将该次操作的解析和执行结果向本地组播域的协作用户广播,协作用户接收到后修改本地操作日志、状态向量和相应数据对象的值.

4 系统实现与性能分析

在电子白板 SmartBoard 系统中基本实现了 madOPT 并发控制框架,提供数据传输和并发控制服务,在可扩展性方面与 Cova 服务器进行交互,将所需数据对象及其操作语义定义成 Cova 类并进行编译、载入系统,使系统支持对新数据对象的并发控制服务,提高系统并发控制的可扩展性.同时,系统实现时提供了对协作数据对象加锁的功能,使协作用户可以申请对某个数据对象的加锁来完成独立的操作.下面将对本文的 madOPT 算法在运行效率方面作出的改进进行性能分析.令比较 1 条操作日志记录的平均时间为 T_0,执行 1 个转换函数的平均时间为 T_1,协作对象的操作日志平均记录数为 M_0,对于一个协作对象 o,被操作次数为 m,那么对象 o 的操作执行时间为 $M_0\times T_0+T_1$,其提高的效率为

$$\eta=(m-M_0)\times T_0/(m\times T_0+T_1) \tag{1}$$

从式(1)可以看出,m 越大,即协作对象的被操作次数越多,η值越大.在 madOPT 算法中,由于通过组播代理收集每一个操作在各个站点的操作结果,从而使协作对象的操作日志大大减小,M 的具体数值根据网络传输状态、操作具体情况而定.在协作过程中,对协作对象的操作过程满足 Poisson 分布,则整个系统提高的效率为

$$\Delta\eta=\Sigma(m_i-M_0)\times T_0/\Sigma(m_i\times T_0+T_1) \tag{2}$$

由于 T_1 可以简化为 T_0 的整数倍,令 $T_1=n\times T_0$,那么式(2)可以简化为

$$\Delta\eta=\Sigma(m_i-M_0)/\Sigma(m_i+n) \tag{3}$$

令系统的总操作次数为 $M=\Sigma m_i$,系统的协作数据对象总数为 $N=\Sigma o_i$,式(3)进一步简化为

$$\Delta\eta=1-N\times(M_0+n)/(M+N\times n) \tag{4}$$

由式(3)、式(4)可知,M_0 和 n 为常数(一般在系统实际运行时,M_0=3,n=5),$M>>N$,M,m_i 越大,提高的效率$\Delta\eta$越大.这与本文要解决的问题相一致,即在协作过程中产生大量协作数据时,由于 dOPT 算法需要遍历操作日志,导致系统运行效率降低,而 madOPT 并发控制方法有效地解决了这一问题.如图 2 所示,当系统中分别有 50,60 个协作数据对象时,取 n=5,M_0=3,随着操作次数的增加,其系统运行效率也提高越多,当 M 趋向于无穷大时,$\Delta\eta$接近于 1,即 madOPT 算法在极限情况下,与 dOPT 算法相比,可提高运行效率 1 倍.

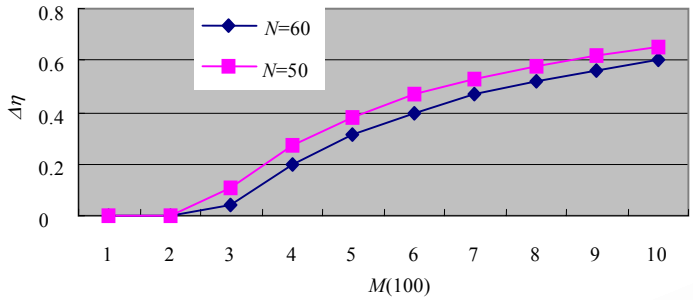

Fig.2 Performance improvement of madOPT algorithm

图 2 madOPT 算法的性能改善

5 结 语

本文在 dOPT 等操作转换算法的基础上提出了基于协作对象数据属性及其操作语义的并发控制框架 madOPT,针对实时协作系统的并发控制过程中由于遍历操作日志带来的系统性能降低等问题作出了改进,并结合数据传输框架,利用 Agent 把协作对象及其操作方法的定义加载到并发控制服务中,提高了系统并发控制服务的可扩展性.madOPT 有效地解决了系统运行效率的问题,但是对于解决 Undo 等操作的并发意愿冲突问题,仍然是依靠优先级来解决争端,并不能完全公平地解决这一问题,加上实时协作系统对本地响应速度的高度敏感,本文的层次处理模式仍然有不够完善的地方,今后我们将沿着这两个方向继续进行研究.

References:

[1] Ellis CA, Gibbs SJ, Rein GL. Groupware: Some issues and experiences. Communications of the ACM, 1991,34(1):39~58.

[2] Greenberg S, Marwood D. Real-Time groupware as a distributed system: Concurrency control and its effect on the interface. In: Smith JB, ed. Proc. of the ACM Conf. on Computer Supported Cooperative Work. Chapel Hill: ACM Press, 1994. 207~217.

[3] Ellis CA, Gibbs SJ. Concurrency control in groupware systems. In: James C, ed. Proc. of the ACM SIGMOD Conf. on Management of Data. Seattle: ACM Press, 1989. 399~407.

[4] Suleiman M, Cart M, Ferrie J. Serialization of concurrent operations in a distributed collaborative environment. In: Stephen C, ed. Proc. of the ACM SIGGROUP Conf. on Supporting Group Work. Phoenix: ACM Press, 1997. 435~445.

[5] Ressel M, Nitsche-Ruhland D, Gunzenhauser R. An integrating, transformation-oriented approach to concurrency control and Undo in group editor. In: Gary O, ed. Proc. of the ACM Conf. on Computer Supported Cooperative Work. Cambridge: ACM Press, 1996. 288~297.

[6] Sun CZ, Ellis C. Operational transformation in real-time group editors: Issues, algorithms, and achievements. In: Poltrock S, ed. Proc. of the ACM Conf. on Computer Supported Cooperative Work. Seattle: ACM Press, 1998. 59~68.

[7] Sun CZ, Jia XH, Zhang YC, Yang Y. A generic operation transformation scheme for consistency maintenance in real-time cooperative editing systems. In: Stephen C, ed. Proc. of the ACM SIGGROUP Conf. on Supporting Group Work. Phoenix: ACM Press, 1997. 425~434.

[8] Yang GX, Shi ML. oodOPT: A semantics-based concurrency control framework for fully-replicated architecture. Journal of Computer Science and Technology, 2001,16(6):531~543.

[9] Yang GX, Shi ML. Cova: A programming language for cooperative applications. Science in China (Series F), 2001,44(1):73~80.