维普资讯 http://www.cqvip.com

协同编辑系统中保持共享对象一致性的研究

戚伟业，刘 弘
（山东师范大学 信息科学与工程学院，山东 济南 250014）

摘 要：如何保持共享对象的一致性是实时协同设计中的核心问题。文中介绍了一种新颖的保持共享对象一致性的方法。根据协同图形编辑系统中操作冲突的特点，使用消息驱动机制，配合同步机制和冲突检测与消解机制，使得系统既能保持共享对象的一致性，又满足了实时协同设计中的响应性、因果顺序一致性，并最大限度地保持了用户的操作目的。

关键词：协同设计；消息驱动；冲突检测与消解；保持一致性
中图分类号：TP391 **文献标识码**：A **文章编号**：1673－629X(2008)03－0063－04

Research on Consistency Maintenance of Shared Object in Collaborative Graphics Editing Systems

QI Wei-ye, LIU Hong
(Information Science and Engineering School, Shandong Normal University, Jinan 250014, China)

Abstract: How to maintain consistency of shared object is the kernel problem in real－time collaborative design. Introduces a novel method on consistency maintenance of shared object. Based on characteristic of operation conflict in collaborative graphics editing systems, employing the mechanism of message driving, with synchronization mechanism, collision detection and resolve mechanism, making system satisfy with consistency of shared object, responsiveness and causal ordering, consistency of operation intention in collaborative design.

Key words: collaborative design; message driving; collision detection and resolve; maintain consistency

0 引 言

在协同系统的研究中，主要有以下几种方法保持共享对象的一致性。锁机制是最早采用的方法，由于响应性差和锁粒度不好掌控的原因限制了它的应用[1]。操作转换法在协同文本编辑系统中得到很大的发展，但它明显不适用于协同图形编辑系统[2,3]。操作序列化法一般作为辅助方法应用到协同系统中，但在单独使用时存在响应性差，无法保持所有用户的目的等情况[4,5]。

多版本法是现在解决协同图形编辑系统中保持共享对象一致性问题最常用的方法[6,7]。它较好地满足了收敛性、因果关系一致性、操作目的一致性等要求，但也存在存储与检索多版本中对象困难，不能有效地解决非几何属性操作之间冲突等问题。

文中介绍的保持共享对象一致性的方法，根据协同图形编辑系统中操作冲突的特点，利用消息驱动机制，配合同步机制、冲突检测与消解机制，使得系统能够满足文中提出的一致性模型。

收稿日期：2007－06－08
基金项目：国家自然科学基金(69975010, 60374054)；山东省自然科学基金(Y2003G01, Z2004G02)
作者简介：戚伟业(1980－)，男，山东日照人，硕士研究生，研究方向为CSCW、并发控制、实时同步；刘 弘，博士，教授，主要研究遗传算法、多Agent协作、协同设计。

1 一致性模型

为了说明该一致性模型，首先介绍几种符号的涵义[6]。

OP代表协同编辑会话中的操作集；Oid表示操作的标志符；Target(O_i)表示操作O_i所作用的对象；Att. Key(O_i)表示操作O_i所作用的对象的属性键。

下面是各协同站点产生的操作之间关系定义：

1)因果顺序关系"→"：O_1和O_2是OP中由站点I和J生成的操作，如果(1)I=J并且O_1产生在O_2之前；或者(2)I≠J并且O_1在站点J的执行发生在O_2产生之前；或者(3)存在一个操作O_x，使得$O_1 \rightarrow O_x$并且$O_x \rightarrow O_2$；以上条件成立，就说O_1在因果顺序上早于O_2，表示为$O_1 \rightarrow O_2$。

2)依赖与并发关系：如果OP中两个操作O_1和O_2，(1)存在$O_1 \rightarrow O_2$，则O_2依赖于O_1；(2)如果既不存在$O_1 \rightarrow O_2$，也不存在$O_2 \rightarrow O_1$，则说明O_1和O_2存在并发关系，表示为$O_1 \| O_2$。

维普资讯 http://www.cqvip.com

3)冲突关系"□":两个操作 O_1 和 O_2,当且仅当(1)$O_1 \| O_2$;(2)Target(O_1) = Target(O_2);(3)Att. Key(O_1) = Att. Key(O_2);以上条件成立,则这两个操作 O_1 和 O_2 冲突,表示为 $O_1 \square O_2$。

4)相容关系"⊙":如果操作 O_1 和 O_2 不是互相冲突的,则 O_1 和 O_2 是相容的,表示为 $O_1 \odot O_2$。

一致性模型要求具有以下三个属性:

(1)收敛性:它要求在执行同一个操作集 OP 中的相同操作之后各协同站点的共享对象的副本相同。

(2)因果一致性:它要求有上面定义的因果顺序关系的操作在各协同站点按照它们的因果顺序执行。

(3)目的保持性:当操作之间有上面定义的冲突关系时,它要求保持具有更高角色权限的操作的目的。

文中使用消息驱动机制结合传统的时间戳机制[8]保持了系统中操作的因果一致性。同时使用冲突检测与消解机制解决并发操作中的冲突,并结合同步机制与角色权限机制保持具有更高角色权限的操作的目的。

2 消息驱动机制

文中系统中的核心机制是消息驱动机制。每一项协同设计操作对应一个注册的全局唯一的标准消息,由于各个协同设计站点都是对等的结构,因此消息的发出端与消息的响应端有相同的操作和响应集合,任意设计人员发往各协同站点的已注册标准消息,被各协同设计端接收和处理,然后响应端根据标准消息与操作的映射调用本地的操作更新视图。在网络的环境下通过对注册消息的传递以及协同客户端对消息的识别,实现设计操作在各个协同设计端的再现,从而保证共享对象的一致性。

为了详细介绍这种机制,首先给出几个定义:

* 实体操作函数:协同设计环境下的基本设计操作,如创建各类实体、删除、编辑等等。

* 消息结构:系统中有两种消息。一种是控制消息,一种是应用消息。控制消息是用作同步的 fence 消息与 count 消息;应用消息是用来实现用户操作目的的标准消息。标准消息满足如下的结构:

消息名	对象标志符	对象属性	角色优先级	执行类型	逻辑时钟	.... 参数 n

* 消息注册:调用注册函数将全局唯一的标准消息同一个操作映射类相关联。

* 操作映射类:在该类中完成对一个回显函数的调用。

* 回显函数:本地操作在异地站点实现时需要调用的函数,是标准消息的最终调用对象。

在这里对标准消息的几个参数的确切涵义作进一步的介绍:

1)Execute 参数:当操作之间发生冲突,它记录冲突消解的结果。它的初始值为 true,当操作之间发生冲突后,角色优先级低的操作对应的标准消息的 Execute 参数值改为 False。

2) LogicalClock 参数:它由时段数 TimesliceCount 与状态矢量 SV[i] 组成。在设计开始 LogicalClock 参数的时段部分标为 1,以后标为 fence(同步机制部分定义)携带的变量 f 的值加 1。系统初始化时,SV[i] = 0,对所有的 $i \in \{0,1,\cdots,N-1\}$(N 是协同站点数)。如果 SV[i] 的值为 x,则表示这个标准消息在因果顺序上排在 i 站点发出的第 x 个标准消息之后。

消息驱动的过程为:

(1)为系统中的每个实体操作函数,声明并实现一个该操作的"回显函数"。

(2)定义操作映射类,在该类中调用上面声明的回显函数。

(3)将操作映射类同一个全局唯一的标准消息关联,并注册标准消息到各协同站点。

(4)在实体操作函数中添加发送消息的代码,消息按照上面定义的标准消息结构填写。

这样就建立了本地实体操作函数到一个全局唯一的标准消息、标准消息到操作映射类、消息映射类到回显函数的映射集合。

3 同步机制和冲突检测与消解机制

3.1 同步机制

在介绍同步机制之前,先介绍一个概念:

角色——它是一种权限集,在文中用它来表示权限的高低。比如管理员角色的权限高于普通协同操作者角色的权限。

在协同站点中有且仅有一个协同站点,它担任管理员角色。每隔一定的时间它就会发送同步消息 fence 给包括它自己在内的所有协同站点,fence 本身携带一个变量 f,它表明是那一个时段,它的初始值为 1。

显然,整个系统的设计时间按照接收到的 fence,分成不同的时段。各协同站点把两个 fence 之间的时间作为一个时段。

在 DRMQ(在下文定义)中有专门位置存储时段消息和时段消息总数。当各协同站点接收到 fence 时,按照 f 标明的时段,把它存入这个时段的本地 DRMQ 中。并且各协同站点在接收到 fence 之后,会立即发送一个同步消息 count 给除本地站点之外的所有协同站点。count 本身携带两个变量 fi 和 fc,fi 与接收到的

维普资讯 http://www.cqvip.com

fence的变量f具有相同值,fc的值为这个站点在fence标明的时段内所发送的标准消息个数。当各协同站点接收到count时,根据fi的值确定这个时段的DRMQ。然后统计各个count携带的fc的总数并存入DRMQ中。然后将统计的总数与应用程序处理的这个时段的标准消息个数比较,如果相等,则删除这个时段的DRMQ。如果不相等,则每次在处理完一个新的此时段的标准消息后,进行比较,直到相等为止,然后删除这个时段的DRMQ。

因为文中的一致性模型要求在每个时段内,每个对象上保持具有最高角色优先级的操作的目的,所以在各协同站点,只有同步消息fence被接收,并且它标记的时段的所有标准消息都处理完,而且此时段的fence和count删除后,才开始处理下一时段的标准消息。

3.2 冲突检测与消解机制

高效的冲突检测与消解机制是协同系统设计的关键,它关系到系统的并发性、一致性和响应性[9]。

由操作冲突的定义知道,相同对象相同属性上的操作之间会发生冲突。而文中的驱动机制是消息驱动机制,这就为冲突检测提供了新的方向。只要在消息携带的参数中添加对象标志符参数与属性参数,就可以对消息封装的操作进行冲突检测。

在设计过程中,有三种情况需要进行冲突检测,这将在下文的分布式算法中介绍。具体到检测操作之间的冲突,在文中只需检测两个标准消息的操作对象参数与属性参数是否相同即可。如果两个参数都相同,则这两个操作发生冲突;否则操作不冲突。

文中的系统中,每个协同站点都具有不同的角色,即它们具有一个权限优先级序列。文中的一致性模型要求保持具有更高角色权限的操作的目的。因为系统是消息驱动运转的,所以冲突消解机制也是建立在标准消息携带的角色优先级参数与Execute参数之上的,它们协助解决操作冲突。

当需要进行冲突消解时,只需把封装这两个操作的标准消息中所携带的角色参数进行比较,将权限低的标准消息的布尔型参数Execute值改为false即可。最后只有Execute参数值为true的消息,才能调用它映射的函数执行。Execute参数值为False的消息,只把它放入OOQ中。经过上面的冲突检测与消解,在某一个时段,保留在某个对象上的操作效果是这时段此对象上具有最高角色优先级的操作结果。

4 消息发送与接收处理机制

各协同站点的应用程序在运行时维持了四种队列:临时消息队列(TMQ,Temporary Message Queue)、执行消息队列(EMQ,Execute Message Queue)、对象操作队列(OOQ,Object Operations Queue)、动态接收消息队列(DRMQ,Dynamic Receive Message Queue)。协同站点的每个对象都有一个TMQ和OOQ。TMQ存储的是某个时段相应对象上具有最高角色权限的操作对应的标准消息,所以它只存储一个标准消息。它是用来方便冲突检测的队列。为了方便用户了解每个对象的操作历史,应用程序为每个对象建立了OOQ。EMQ中按因果序存储了一些标准消息,它们可以立即被用来驱动它们映射的实体函数执行。DRMQ是按照fence划分的时段建立的。当本地站点接收到第一个这个时段的消息时,建立动态接收队列$DRMQ_I$(I代表第几个时段),当这个时段的消息处理完后,$DRMQ_I$被删除。

每个协同站点的应用程序都维持了一个状态矢量$Exe[i]$,系统初始化时,$Exe[i]=0$,对所有的$i\in\{0,1,\cdots,N-1\}$(N是协同站点数)。如果$Exe[i]$的值为y,则表示EMQ接收的最近的i站点发出的标准消息是i站点发送的第y个标准消息。这个状态矢量是用来协助按照因果顺序处理标准消息。如果i站点发送一个标准消息,它的逻辑时钟参数中的状态矢量的值是这样的,$SV[i]$是i站点发送消息的计数,$SV[j]=Exe[j],j\in\{0,1,\cdots,N-1\},j\neq i$。

消息发送和消息接收与处理的分布式算法:

```
void SendMessage(ObjectID, ObjectAttributeID, OID, Sitei) //消息发送算法
{if(不存在ObjectID表示的对象)
    {调用实体操作类中OID对应的函数立即执行;
    创建ObjectID对应的TMQ与OOQ;
    提取这个操作的一些参数,比如OID,ObjectID等,再附加另外一些参数,如Sitei,LogicalClock等,把这些参数封装成一个标准消息发送给除本地站点之外的各协同站点;
    把这个标准消息存入相应的TMQ与OOQ;
    return;}
else(存在ObjectID表示的对象)
    {将ObjectID对应的TMQ中的标准消息封装的操作与OID进行冲突检测与消解;
    if(OID对应的标准消息的Execute参数值为true)
    {撤销TMQ中的标准消息封装的操作的执行效果;
    调用实体操作类中OID对应的函数立即执行;
    提取这个操作的一些参数,并封装成一个标准消息发送给除本地站点之外的各协同站点;
    用这个标准消息替换ObjectID对应的TMQ中的标准消息;
    把这个标准消息存入相应的OOQ;
    return;}
else
```

维普资讯 http://www.cqvip.com

Notice("已经有更高权限优先级的操作在此对象的这个属性上执行");
return;}
}
void RMADW(Message$_i$)// 接收消息与处理算法
{if(Message$_i$ 是 fence 类消息)
{将时段数设为 Message$_i$ - >f + 1;
把消息放入相应的 DRMQ;
return;}
if(Message$_i$ 是 count 类消息)
{将消息总数设为 MessageSum + Message$_i$ - >fc;
把消息放入相应的 DRMQ;
return;}
if(Message$_i$ 是标准消息)
{if(不存在相应的 DRMQ)
{创建相应的 DRMQ;
把消息 Message$_i$ 放入 DRMQ;}
else(存在相应的 DRMQ)
{按照因果顺序查找 Message$_i$ 在 DRMQ 中的位置并将其放入;}
将 Message$_i$ 与 DRMQ 中的所有的标准消息进行冲突检测与消解;}
if(Message$_i$ 在 DRMQ 的对首)
{while(Message$_i$)
{//将 Message$_i$ 的逻辑时钟的状态矢量部分与本地应用程序维持的 Exe[i]对应比较,看它是否是下一个按照因果顺序可以出队列的标准消息
if(Message$_i$ 满足出队列的条件)
{将 Message$_i$ 从 DRMQ 中取出并放入 EMQ 中;
Exe[Message$_i$ - >Site] + +;
Message$_i$ = Next(Message$_i$,DRMQ$_j$);}
else break;}}
return;}
void DoWithEMQ (EMQ)// 对 EMQ 进行处理的算法
{if(EMQ 为空) return;
else(EMQ 不为空) {
查看 EMQ 队首的标准消息的对象标志符,看系统中是否已经有这个对象;
if(对象不存在){
查找此标准消息映射的实体函数映射类中的相应实体函数调用执行;
创建这个对象的 TMQ 队列与 OOQ 队列;
把这个标准消息存入 TMQ 与 OOQ;
把此时段的计数器 Sum 加 1;
查看 DRMQ 中存储的时段消息总数,将它与 Sum 比较,如果相等删除这个时段的动态接收队列,清空本地站点所有对象的 TMQ;}
else(对象存在){
if(这个标准消息的 Execute 参数值为 true) 将此标准消息封装的操作与相应对象的 TMQ 中的标准消息封装的操作进行冲突检测和消解;
else(这个标准消息的 Execute 参数值为 false)
只是把此标准消息放入相应 OOQ 中;
if(经以上处理后这个标准消息的 Execute 参数值仍为 true)
{
使用 UNDO 机制取消对象的 TMQ 中标准消息封装的操作的执行效果;
调用此标准消息映射的实体函数映射类中的相应实体函数执行;
把此标准消息放入相应 OOQ 中,并且用它替换 TMQ 中的那个标准消息;
把此时段的计数器 Sum 加 1;
查看 DRMQ 中存储的时段消息总数,将它与 Sum 比较,如果相等删除这个时段的动态接收队列,清空本地站点所有对象的 TMQ;}}}

需要注意的是,在文中的系统中每个协同站点的应用程序只有在本地没有或者只有一个 DRMQ 时才能接收用户的操作。这样才能保证每个时段每个对象上保持的是最高角色优先级用户的操作目的。

5 同步设计系统的实现

文中的研究所基于的基本开发环境为 VC. NET 2003,所选用的 3D 支持环境为 ACIS 开发环境。基于以上平台实现了一个协同化、集成化的设计系统。

系统的各个协同设计端采用对等的网络结构。系统实现主要有以下几个类:客户类 HoopsClient,网络环境类 HNet,消息注册类 HRegistrar,消息 - 回显映射类 HMyMessageHandler。

(1)HoopsClient 用于标识加入到设计任务中的客户,它封装了基本的消息发送和消息接收功能,同时扩展了消息广播和消息列表广播等功能。另外,设计任务的加入和退出的相关操作也在此类中封装。

(2)网络环境类 HNet 是对 Socket 通信机制的一个封装,提供系统进行通信所必需的 socket 网络环境。这当中包括环境初始化操作,协同站点之间建立连接的操作。除此之外,该类封装了相应的对设计任务的操作,例如创建删除 session 等。

(3)HRegistrar 类提供给系统将以全局唯一的标准消息同一个 HMyMessageHandler 类的对象建立映射的方法。

(4)在 HMyMessageHandler 类内部调用消息映射的回显函数更新视图保持共享对象的一致性。

6 结论与展望

通过网络上多个协同用户共同设计产品的实验,

(下转第 70 页)

维普资讯 http://www.cqvip.com

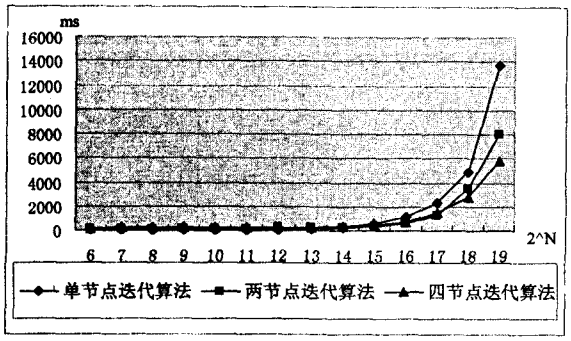

图 2 不同节点迭代算法比较

同时,以上也表明了,在基于 GT4 的分布式并行计算环境下,使用合适的并行算法进行分布式并行计算是可行的。

4 结束语

文中 GT4 环境下,实现了并行 FFT 迭代算法和并行 FFT 递归算法,并对实验数据进行了分析,说明了网格平台下进行分布式并行计算的可行性。但是如何以更小的代价来实现高性能的分布式并行计算? 网格环境中资源调度[9]对计算性能起着至关重要的作用,高效的调度算法或策略可以充分利用网格系统的处理能力,从而提高应用程序的性能。这正是下一步要研究的课题。

参考文献:

[1] Foster I, Kesselman C. The Grid: Blueprint for a New Computing Infrastructure[DB/OL]. San Francisco, CA: Morgan Kaufmann, 1999. http://www.gridforum.org.

[2] 易 明,金 海. 基于 WSRF 的 Web 服务资源的设计[J]. 计算机工程,2006,32(23):262 - 264.

[3] Foster I. Globus Toolkit Version 4: Software for Service - Oriented Systems[J]. Journal of Computer Science and Technology, 2006, 21(4): 513 - 520.

[4] Foster I, Kesselman C. The Physiology of the Grid[M]//Grid Computing: Making the Global Infrastructure a Reality. Chinchester, UK: John Wiley & Sons, 2003: 217 - 249.

[5] 林伟伟,张志立,齐德昱. 基于网格的分布并行计算策略[J]. 计算机工程,2006,32(9):104 - 106.

[6] 都志辉,陈 渝,刘 鹏. 网格计算[M]. 北京:清华大学出版社,2002.

[7] 林伟伟,齐德昱,王振宇. 网格计算环境下分布并行计算的一种实现方法[J]. 计算机工程与应用,2005,41(27):32 - 34.

[8] 陈国良. 并行算法实践[M]. 北京:高等教育出版社,2003.

[9] 赵宏伟,邵一川. 基于 Globus 网格服务的作业调度的实现[J]. 计算机工程与设计,2006,27(9):1665 - 1668.

(上接第 66 页)

证明文中的保持共享对象一致性的方法较好地满足了一致性模型提出的收敛性、因果顺序性和操作目的性的要求。因为本地的操作可以立即执行,所以很好地满足了实时协同设计的响应性要求。通过消息驱动机制进行协同,网络传输的数据主要是信息量很少的标准消息,所以网络负载轻,同时各协同站点在本地调用标准消息映射的实体函数执行,提高了响应性。因为系统只保留最高角色权限的操作效果,也就不再有非几何属性操作冲突后的问题,而且每次需要撤销用户的操作效果时,只需要 UNDO 一次即可。

参考文献:

[1] Preston J A, Prasad S K. Exploring Communication Overheads and Locking Policies in a Peer - to - Peer Synchronous Collaborative Editing System[C]//ACM Southeast Conference. New York, NY, USA: ACM Press, 2005: 286 - 394.

[2] Ellis C A, Gibbs S J. Concurrency control in groupware systems[C]//ACM SIGMOD Conference on Management of Data. Seattle, WA: ACM, 1989: 399 - 407.

[3] Sun C, Ellis C A. Operational transformation in real - time group editors: Issues, algorithms, and achievements[C]//In: Poltrock S, Grudin J, eds. Proceedings of ACM 1998 Conference on Computer Supported Cooperative Work. New York: ACM Press, 1998: 59 - 68.

[4] Enokido T, Takizawa M. Concurrency Control Based on Significancy on Roles[C]//The IEEE 11th International Conference on Parallel and Distributed systems. New York: ACM Press, 2005: 196 - 202.

[5] Sabbir A, Ravindran K N. Concurrency Control Frameworks for Interactive Sharing of Data Spaces in Real - time Distributed Collaborations[R]. USA: Department of Computer Science, City University of New York, 2005: 16 - 26.

[6] Sun C, Chen D. Consistency Maintenance in Real - time Collaborative Graphics Editing Systems[J]. ACM Transactions on Computer - Human Interaction, 2002, 9(1): 1 - 10.

[7] Dou Wanfeng, Zhu Ming. Cooperative multi - versioning technique based on version replication[C]//Proc. of the 10th International Conf. on Computer Supported Cooperative work in Design. Nanjing, China: [s.n.], 2006: 159 - 164.

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

[9] 余春艳,侯宏仑,吴明晖,等. 协同设计中以实体为中心的并发操作控制机制[J]. 计算机辅助设计与图形学学报,2004,16(9):1289 - 1294.