分布进程间通信合作因果关系及因果序的保证*

赵 宏 李华天

(东北大学计算机系,沈阳 110006)

摘要 本文介绍了在计算机分布式环境下相互合作的进程中通信合作事件间的因果关系(Causal Relations),基于在一组相互合作的进程间可能同时服务于多个任务,而且这些任务间可能并不相关这一观点,引入了事件类(Event Class)概念,并利用事件类概念,提出了保证因果序(Causal Ordering)的算法.

关键词 进程组,通信合作,因果序列,事件类.

在分布环境下,进程群体(Process Group)为不同应用客体间的各种通信合作,从而也为分布应用任务提供了可靠和抽象的支持机制.一个功能实体由一组与某个或某些应用相关的进程组成,这些进程相互合作来完成相应任务[1-3].

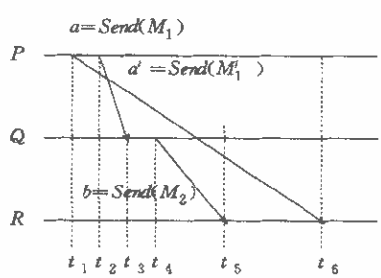

图1 事件的因果关系

合作进程间的一个重要关系是事件间的因果关系.在一组合作进程中,任何进程的动作或任何两个进程间的消息传递,称为**事件**(在这里,事件是指进程间的消息传送).由于一组合作进程往往服务于同一个任务,可能涉及共同的元素(如变量、文件等),使得事件间在时间上有制约关系.如图 1 所示,在时间 t_1,进程 P 发生事件 $a=Send(M_1)$到进程 R,在时间 t_2,进程 P 发生事件 $a'=Send(M_1')$到进程 Q. 事件 a 对进程 P 的状态可能有修改,这会影响事件 a'. 由于继承性,进程 Q 在时间 t_4 发生的事件 $b=Send(M_2)$到进程 R 同样可能受这种影响.对进程 R 来说,为了保证该进程群体一致的观点(View),要求能按照先 a 后 b 的次序受事件 a 和 b 的影响,这就是因果序.然而,由于消息传递时间的变化和不确定性,消息 b 可能在消息 a 之前到达进程 R,从而破坏进程群体的一致观点,所以需要一种方法来保证这种因果序.

因果序保证算法基本可分为两类:基于消息到达后马上交付处理的[1]和基于延迟处理

* 本文 1992-03-12 收到,1992-12-02 定稿

作者**赵宏**,1954 年生,副教授,主要研究领域为计算机分布式系统,多媒体系统.**李华天**,1922 年生,教授,博士导师,主要研究领域为计算机应用,网络协议工程及分布式系统应用.

本文通讯联系人:赵宏,沈阳 110006,东北大学计算机系

的[4]. 第一种方法中,每个合作进程保持一个缓冲区,其中含有该进程发出和接收的消息. 如果一个消息 M 从一个进程发出,它将携带该进程缓冲区中所有内容. 这样,消息可以在到达后立即得到处理. 因为如果 M 到达了目的地 P,其因果相依的所有消息也一定到达了 P. 另一种方法中,消息只携带它可能因果相依的消息中少量必要信息. 此时,有些消息在其到达目的地后不能立即处理,而在以后某个适当时候交付处理. 这种方法可以减少消息所需要携带的信息量.

通过分析我们可以发现,一组相互合作的进程可能同时服务于多个任务. 在某些时候,这些任务以及属于不同任务的事件间没有相互关系. 无论看起来它们是否在时间上有先后关系,相互无关的事件不应该相互影响. 然而目前一些因果序保证算法都是基于一组合作进程中的事件,都是互相相关的这种观点,它们无法或不适于处理以上描述的情况.

1　事件类及其定义

为了区别相互无关(或在一定时间内无关)的事件序列,我们这里引进一个事件类(Event Class)的概念. **事件类**是表明合作进程之间发生的事件是否因果相关的一种抽象. 属于同一类的事件是相关的,相互之间可能有因果制约关系,需要按一定顺序发生和处理. 属于不同类的事件则是不相关的,可以同时或以任意顺序发生.

定义 1. 令 a 和 b 是一组合作进程中的两个事件,则 $a \Rightarrow b$(a 直接影响 b),如果:

(1)a 和 b 发生于同进程(对应相同任务),且 a 先于 b 发生;或者

(2)a 和 b 发生于不同进程(对应相同任务),且 a 是发送一个消息,而 b 是接收该消息.

定义 2. a 相关 b($a \rightarrow b$)如果:

(1)$a \Rightarrow b$ 或者

(2)$a \Rightarrow b$ 为假,但存在一个事件集合 $\{e_i\}, i \in [N]$,使得 $a \Rightarrow e_1 \Rightarrow e_2 \Rightarrow \cdots\cdots \Rightarrow e_N \Rightarrow b$.

(3)如果 a 相关 b 为真,则 b 相关 a 为假,即"相关"关系是反对称的.

定义 3. 如果既没有"a 相关 b"也没有"b 相关 a",则 a 和 b 是不相关的,记作 $a // b$.

定义 4. 对任意 a,令 $Pa = \{a' \mid a'$ 相关 $a\}$,则 Pa 称为事件 a 的**"先行集"**(Precedent).

对任意观察时间 t 和事件 b,令 $D_b^t = \{b' \mid b$ 相关 $b'\}$,则称 D_b^t 为事件 b 在时间 t 时的**"后继集"**(Descendent).

定义 5. 对任意事件 C 在时间 t 时的事件类为 E_c,满足以下关系:

(1)$c \in E_c$

(2)如果 $x \in E_c$,则对所有 $y \in P_x$,有 $y \in E_c$

(3)如果 $x \in E_c$,则对所有 $y \in D_x^t$,有 $y \in E_c$

该定义在图论中意味着,在时间 t,事件 c 的事件类是在对应的无向图上从事件 c 可达的(Reacheble)所有事件的集合. 见图 2.

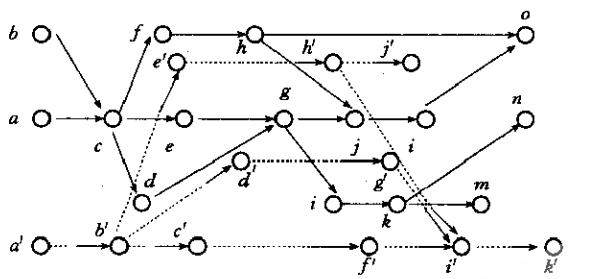

图2 相关和不相关事件

由以上定义我们知道:若 $E_a \neq E_b$,则 $E_a \cap E_b = \varnothing$. 因此,对任意两个事件 $a' \in E_a$ 和 $b' \in E_b$,a' 和 b' 是不相关的. 在图 2 中,圆圈表示事件,实线和虚线分别表示两种独立的"⇒"关系. 其事件类分别为:

$E_0 = \{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o\}$

$P_0 = \{a,b,c,d,e,f,g,h,j,l\}$

$D_g^t = \{i,j,k,l,m,n,o\}$

$E_k' = \{a',b',c',d',e',f',g',h',i',j',k'\}$

$P_k' = \{a',b',c',d',e',f',g',h',i'\}$

$D_h^t = \{i',j',k'\}$

这里,$E_0 = E_m = E_n$ 是相同的事件类. E_0 和 E_k' 是不同的事件类,则 $E_0 \cap E_k' = \varnothing$. 在属于事件类 E_0 的事件中,$d \,//\, e$,$e \,//\, f$,$d \,//\, h$,$j \,//\, k$ 等等. 在属于事件类 E_k' 的事件中,$c' \,//\, d'$,$c' \,//\, e'$,$d' \,//\, f'$ 等等. 对任何在 E_0 中的事件 x 和在 E_k' 中的事件 x' 来说,$x \,//\, x'$ 是真的.

2 基于事件类的延迟提交算法

首先,为便于描述算法,我们作如下说明:

每个进程 P 保持一个发送序列向量 SSV(Sending Sequence Vector):$S = (S_1, S_2, \cdots, S_n)$ 和一个接收序列矩阵 RSM(Receipt Sequence Matrix):

$$R = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1m} \\ r_{21} & r_{22} & \cdots & r_{2m} \\ \cdots\cdots & & & \\ r_{i1} & r_{i2} & \cdots & r_{im} \\ r_{n1} & r_{n2} & \cdots & r_{nm} \end{bmatrix}$$

这里 n 表示进程个数,m 表示不同事件类个数. S_i 表示从进程 P 发向进程 P_i 的消息序号,r_{ik} 表示在进程 P 提交的来自进程 P_i,且属于事件类 K 的最后一个消息的消息序列号 $(1 \leqslant i \leqslant \eta, 1 \leqslant k \leqslant m)$.

每个合作进程拥有一个信息缓冲区,记作 $BUF\_INF()$,其中含有关于可能要影响该进程将来发生的事件的事件信息. 这些信息是多个元组,每个元组记作 (P_i, I_d, S, P_j),表示一个事件,即从进程 P_i 向进程 P_j 发送的属于事件类 Id 和带有消息序列号 S 的消息已经发出,但可能还没有到达 P_j 或没有在那里提交. 同时,每个进程还保持着另一个延迟提交消

息缓冲区 $BUF\_UND(\ )$,其中包含由于违反了因果序而被暂时延迟提交的消息.

当进程 P_i 向进程 P_j 发送消息 M 时,在 M 中要携带一些信息,包括:

(1)一个"因果印记"(Causal_Stamp),记作 $M:(P_i,I_d,S,P_j)$,其中 I_d 是该消息对应的事件类标识,S 是进程 P_i 中的 SSV 内对应于 P_j 的分量.

(2)一个伴随缓冲区(Accompanying_Buffer),记作 BUF(M),其中包含进程 P_i 中的 $BUF_INF(P_i)$中与该消息相关的消息的信息.

由于彼此合作的进程之间消息传送的时间花费是变化的和不可预测的,有些消息可能先于它们所因果依赖的消息而到达其目的地.在我们下面的算法中,当这种情况发生时,根据是否为相同事件类而决定是否延迟对其处理:

当发送消息时,例如,进程 P_i 向进程 P_j 发送消息 M:

(1)P_i 保持的 SSV 中对应分量 S_j 增加,即 $S_j=S_j+1$.

(2)用给定的对应于 M 的事件类标识 I_d 和 S_j 以及发送及接收进程标识组成一个因果标识(P_i,I_d,S_j,P_j),附给 M,形成 M 的因果印记.同时,在 M 的伴随缓冲区中携带 $BUF_INF(P_i)$中所有相同类属标识的元组.

(3)一个对应于 M 的因果印记的元组被加入到 $BUF_INF(P_i)$中,并且去掉 $BUF_INF(P_i)$中所有满足条件 $S_x<S_j$元组(P_i,I_d,S_x,P_j),即与 M 具有相同的源和目的进程标识,相同的事件类属标识,而且消息序列号小于 S_j 的所有元组.注意:在伴随缓冲区 $BUF(M)$中并没有包括元组(P_i,I_d,S_j,P_j),而在 $BUF_INF(P_i)$中则包含了它.

当进程 P_j 接收到因果印记为(P_i,I_d,S,P_j)的消息 M 时:

(1)如果在 M 的伴随缓冲区 $BUF(M)$中并不存在任何元组(P_x,I_d,S_x,P_j),即与 M 有相同目的地、相同事件类属标识的任何元组,说明 M 并不与什么消息构成因果序.因此,可以在进程 P_j 中直接提交处理它.

(2)否则,对于 $BUF(M)$中的每个元组(P_x,I_d,S_x,P_j),如果 $S_x=r_{zk}$(为方便起见,令 $k=I_d$,r_{zk}是 P_j 中对应于 P_x 的类属的消息序列号),表示某个 M 所依赖的消息已经到达进程 P_j.因此,可以从 $BUF(M)$中把该元组除掉.此后,如果在 $BUF(M)$中仍然存在类似于(P_x,I_d,S_x,P_j)的元组,则把消息 M 暂时放入队列 $BUF—UND(P_j)$中,延迟 M 的提交,一直等到其它在此进程提交的消息引起的动作将 $BUF(M)$中所有类似于(P_i,I_d,S_x,P_j)的元组全部去除,方可提交 M.

当一个因果印记为(P_i,I_d,S,P_j)的消息 M 在进程 P_j 提交时:

(1)将 $BUF(M)$合并到 $BUF_INF(P_j)$中去,即对在 $BUF(M)$中每个元组(P_m,I_d,S_x,Q),如果在 $BUF_INF(P_j)$中存在某个对应的元组(P',I_d',S_y,Q'),它们之间有 $P'=P_m$,$I_d=I_d'$及 $Q'=Q$,则将 BUF_INF 中对应的元组改成(P',I_d',S_z,Q'),其中 $S_z=\max(S_x,S_y)$.否则的话,把(P_m,I_d,S_x,Q)直接加入到 $BUF_INF(P_j)$中去.

(2)修改 RSM.按照 M 的因果印记(P_i,I_d,S,P_j),将 RSM 中的对应元素 $r_{ik}(K=id)$改变成为 M 的消息序列号,即 $r_{ik}=S$.这意味着从进程 P_i 发给 P_j 的事件类属标识为 I_d,消息序列号为 S 的消息已给到达 P_j 并在那儿提交了.

(3)检查 $BUF—UND(P_j)$,看是否有什么延迟提交的消息由于消息 M 的提交而可以提交处理了.若有,则提交它.

按照此算法，一个消息 M 只携带与其相关的元组，每个元组可能只由四个元素(一般为整数)组成，而不是象有些算法那样，携带其它消息的整体. 因而该算法减少了许多要传送的信息量，避免了可能使传送的信息量过大而无法有效地工作这一问题.

需要说明的是，该算法区分事件的类别方法是在每个合作进程保持一个发送序列向量和一个接收序列矩阵. 每个发送的消息都有一个消息类属标识来表示其事件类. 相同事件类的消息其标识符相同.

该算法也可很容易地适应和处理动态的情况. 在一组相互合作的进程中，有时会出现新的成员加入、旧的成员退出或失败等情况. 这种变化经常会影响到成员间的关系，同时也影响到事件的因果序以及保持因果序的算法. 因果序的保持算法应该适应这种动态的情况. 该算法对这些都有所考虑，限于篇幅无法一一解释，有兴趣的读者可与作者联系.

3 结 论

在本文中，我们提出了事件类的概念，并基于此概念提出了一种保证因果序的算法. 我们在这里描述了该算法的性能和工作原理，以及它怎样以较低的开销取得有效的结果. 该算法已在分布应用客体的协会模型中应用[5]. 该算法区分了事件的类别，使不同类属的事件序列相互无关，它亦可很容易地适应和处理动态的情况. 实践证明该算法是可行的和有效的. 现有的工作对支持进程组间的通信、对更好地理解分布活动的因果关系，都有着很大的实用价值.

参考文献

1 Birman K P, Joseph T A. Reliable communication in the presence failures. ACM Transactions on Computer Systems, 1987, **5(1)**: 47—76.

2 Sape Mullender. Distributed systems. New York: Addison Wesley—Publishing, 1989.

3 David R Cheriton, Willy Zwaenepoel. Distributed process groups in the V Kernel. ACM Transactions on Computer Systems, 1985, **3(2)**: 77—107.

4 Andve Schiper, Jorge Eggli, Alain Sandoz. A new algorithm to implement causal ordering. Distributed Algorithms: 3rd International Workshop, Nice France, Proceedings Springer—Verleg, 1989: 219—232.

5 Zhao Hong, Wayne McCoy. An association model for distributed application object in distributed systems. ACM SIGOPS, Operating Systems Review, 1990, **24(4)**: 34—51.

6 Leslie Lamport. Time, clock and the ordering of events in a distributed system. Communication of ACM, 1978, **21(7)**: 558—565.

7 Friedemann Matterm. Virtual time and global states of distributed systems. Parallel and Distributed Algorithms, Elsevier Science Publishers B. V., North—Holland, 1989: 215—226.

8 Fefferson D R. Virtual time. ACM Transactions on Programming Languages and Systems, 1985, **7(3)**: 404—425.

CAUSAL RELATIONS AND CAUSAL ORDERING GUARANTEEING ALGORITHM

Zhao Hong　Li Huatian

(*Department of Computer Science, Northeastern University, Shenyang* 110006)

Abstract　This paper introduces the causal relations between events among communicating and cooperating processes in distributed systems. Based on the viewpoint that more than one task may progress among a group of cooperating processes and the tasks may not relate with each other, the authors propose the concept of event class. Furthermore, they propose a causal ordering guaranteeing algorithm using the event class concept.

Key words　Process group, communication and cooperation, casual ordering, event class.