计算机科学1996 Vol. 23 №. 4

消息传递的逻辑瞬时模型

陈华平　陈国良
（中国科技大学计算机系　合肥230027）

摘　要　Most of MPP systems are MIMD distributed memory systems in which the message passing environment is a very important component. With wide research and application of Net of Workstations and Parallel Virtual Machine, it has been become a research point how to communicate effienciently among different stations. This paper describes a logically instantaneous message passing model and a communication algorithm on this model.

关键词　*Message passing*, *Logic instant*, Parallel distributed processing, Process communication.

一、前　言

通常不同站点间的计算进程进行异步通讯时必须相互协调，由于消息在传过程中存在通讯延迟，通常有以下三种类型的错误发生：

(1)进程 A 发送一个消息 m_1“执行任务 X”给进程 C，过十分钟后进程 A 发送一个消息 m_2“请检查进程 C 的执行情况”给进程 B，进程 B 接到 m_2 后，就发送一个消息 m_3 给进程 C 来检查执行情况。但是，进程 C 接到 m_3 时，还没收到消息 m_1，这时进程 C 就搞不清进程 B 在查问什么。

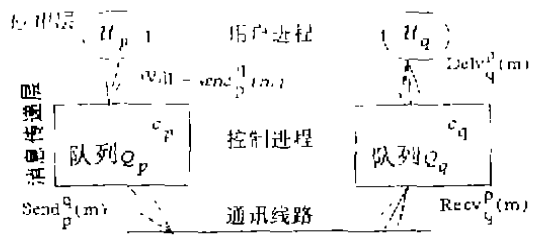
(2)进程 A 发送一个消息 m_1“请将结果放入地点 X”给进程 B，而同时进程 B 发送一个消息“我已把结果放入地点 Y”后往下继续执行其它任务，这样进程 A 一直在地点 X 等待结果。

(3)监控进程 A 正在寻找持有令牌的进程，分别发送消息 m_1 和 m_2“你持有令牌吗?”给进程 B 和进程 C，而这时令牌正在由 B 发送给 C 的路途中，所以 B 和 C 均回答“NO”，这样监控进程 A 错误地认为令牌已丢失。

如果假定每个消息的通讯延迟为零，也就是说每个消息能瞬时传递到目的地，这样上述几类错误均能避免发生。但实际消息传递系统中消息是不可能瞬时传送到目的地的，为此我们可以建立在逻辑上具有瞬时传递特性的消息传递环境，也就是说并行分布处理系统中的每个进程(子任务)执行时就好象执行在理想的瞬时传递模型上。

二、模型和定义

我们假定系统中有 n 个协作进程，每个进程由用户进程 u_p 和控制进程 c_p 组成，u_p 通过 c_p 来同其它进程进行通讯。u_p 和它的 c_p 位于同一站点，它们内部通过共享存储器进行通讯。所有用户进程的集合称为应用层，所有控制进程的集合称为消息传递层。系统结构如下图所示，我们认为每个进程有一个唯一的标识符(ID)，为了讨论简单起见，不妨设进程标识符集合为 $P=\{1,2,\cdots\cdots,n\}$。下面是系统模型结构图：



如果用户进程 u_p 想发送一个应用消息 m 给用户进程 u_q，u_p 首先给它的控制进程 c_p 发一个 will-send$_p^q$(m)命令，以表示它想发送一个消息。如果该命令被控制进程 c_p 选中执行，c_p 就通过通讯线路传送该消息，并通知 u_p 发送事件 Sent$_p^q$(m)已完成。经过一段通讯延迟后，消息被控制进程 c_q 通过 Recv$_q^p$(m)接收到，然后通过 Delv$_q^p$(m)事件将该消息传送给 u_q。在某应用消息相应的 Sent 事件发生之前，用户进程可能由于收到其它消息后，改变了自身的状态，所以它可以取消已发给相应控制进程的该应用消息的 Will-send 命令。除了 Sent 和 Delv 这两个事件外，可能还有其它事件，如接收到系统外部消息请求的 Trigger 事件，以及只与用户进程局部变量有关的

陈华平　博士，讲师，主要从事 PVM 和 MPI 的研究。陈国良　教授，博士生导师，现主要从事并行分布算法和智能计算的研究。

internal 事件。这里特意指出"应用消息"主要是同消息传递系统中的一般控制消息(信号)加以区别。

为了讨论事件发生的先后关系,首先我们有如下定义:

定义2.1 事件 e_1 发生于事件 e_2 之前(用 $e_1 \rightarrow e_2$ 表示),当且仅当满足下列某一条件:(1)e_1 和 e_2 发生于同一个进程中,以局部时钟为标准,e_1 发生于 e_2 之前。这时可表示为 $e_1 \rightarrow e_2$。(2)e_1 是某一应用消息的 Sent 事件,而 e_2 是该应用消息的 Receive 事件。(3)存在事件 e_3,满足 $e_1 \rightarrow e_3$ 及 $e_3 \rightarrow e_2$。

例如,e_1 是用户进程 u_p 的 $Sent_p^s(m)$ 事件,e_2 是控制进程 c_q 的 $Recv_q^r(m)$ 事件,e_3 是用户进程 u_q 的 $Delv_q^s(m)$ 事件,那么有 $e_1 \rightarrow e_2 \rightarrow e_3$。注意这里"$\rightarrow$"是一个偏序关系。

有了上面事件发生先后关系的定义后,下面就可定义消息间的关系。

定义2.2 应用消息 m_1 存在于应用消息 m_2 之前(用 $m_1 < m_2$ 表示),当且仅当满足下列某一条件:(1)m_1 和 m_2 由同一用户进程 u_p 发送,并且 $Sent_p(m_1) \rightarrow Sent_p(m_2)$。(2)$m_1$ 和 m_2 分别由同一用户进程 u_p 发送和接收,并且 $Sent_p(m_1) \rightarrow Delv_p(m_2)$。(3)$m_1$ 和 m_2 分别由同一用户进程 u_p 接收和发送,并且 $Delv_p(m_1) \rightarrow Sent_p(m_2)$。(4)$m_1$ 和 m_2 发送给同一用户进程 u_p,并且 $Delv_p(m_1) \rightarrow Delv_p(m_2)$。(5)存在 m_3,满足 $m_1 < m_3$ 和 $m_3 < m_2$。

例如本文前言所举例子的第一个中,事件发生的先后关系满足 $Sent_A(m_1) \rightarrow Sent_A(m_2)$,$Delv_B(m_2) \rightarrow Sent_B(m_3)$,$Delv_C(m_3) \rightarrow Delv_C(m_1)$,所以消息 m_1,m_2 和 m_3 满足 $m_1 < m_2 < m_3 < m_1$。第二个例子中,事件发生的先后关系满足 $Sent_A(m_1) \rightarrow Delv_A(m_2)$,$Sent_B(m_2) \rightarrow Delv_B(m_1)$,所以消息 m_1 和 m_2 满足 $m_1 < m_2$ 和 $m_2 < m_1$。上面(1)(2)(3)(4)这四种情况分别称为(-,-),(-,+),(+,-)和(+,+)关系。

我们假定用 UP 表示用户进程集合,用 E 表示 UP 中进程所发生的事件集合,M 是应用消息集合,它代表了 Sent 事件 Delv 事件的对应关系。在三元组(UP,E,M)上我们定义四个历史函数,即:

(1)C:时钟历史函数,它把物理时间空间映射到每个进程的时钟空间,即 $UP \times R \mapsto N$,用 $C(u_p,t)$ 表示物理时间 t 时进程 u_p 的时钟值。

(2)Q:状态历史函数,它反映了某个时钟时进程的状态,即 $UP \times N \mapsto US$,用 $Q(u_p,c)$ 表示进程 u_p 在它的时钟 c 时的状态。

(3)A:事件历史函数,它反映了某个时钟时进程的执行事件,即 $UP \times N \mapsto L$,用 $A(u_p,c)$ 表示进程 u_p 在它的时钟 c 时所发生的事件。在某一时钟内最多只允许发生一个事件。

(4)MP:消息传递历史函数,它把应用消息映射到一对物理时间上,即 $M \mapsto R \times R$,如果 $MP(m)=(t_1,t_2)$,那么消息 $m(\in M)$ 在时间 t_1 时发出,在时间 t_2 时被接收到。由于通讯延迟是非负的,所以如果 $MP(m)=(t_1,t_2)$,那么 $t_1 \leqslant t_2$。

一个应用层历史(代表消息传递系统的一次执行操作)H 就是由上述四个历史函数组成,即 $H=(C,Q,A,MP)$,一个消息传递系统 S 是由所有可能的历史组成的集合。

定义2.3 对任何的 $c \in N$,如果 $Q_1(u_p,c)=Q_2(u_p,c)$ 和 $A_1(u_p,c)=A_2(u_p,c)$,那么 H_1 和历史 H_2 关于进程 u_p 等价,记为 $H_1 \overset{p}{\sim} H_2$。如果对任何 $u_p \in UP$,有 $H_1 \overset{p}{\sim} H_2$,那么 $H_1 \sim H_2$。

定义2.4 如果任给 $H \in S_1$,存在 $H' \in S_2$,那么我们称 S_1 蕴含于 S_2(或 S_2 蕴含 S_1),记为 $S_1 \subseteq S_2$。

定义2.5 如果一个消息传递系统 S 对于任何应用消息 m,$Sent_s^s(m)$ 和 $Delv_r^s(m)$ 两个事件之间的物理传递延迟为零,那么称该消息传递系统具有瞬时传递特性(I 特性),记为 S(I)。如果一个消息传递系统 S' 蕴含于 S(I),即 $S' \subseteq S(I)$,那么称系统 S' 具有逻辑瞬时传递特性(LI 特性)。

定义2.6 如果一个消息传递系统 S 中不存在一对消息 m_1 和 m_2 同时满足 $m_1 < m_2$ 和 $m_2 < m_1$,那么我们称这个消息传递系统具有"无消息交叉"(NMC,No-Message-Crossing)特性,记为 S(NMC)。

定理2.1 一个消息传递系统具有逻辑瞬时特性的充要条件是该系统具有 NMC 特性。

证明:即要证明 $S' \subseteq S(I) \Leftrightarrow S' \subseteq S(NMC)$。(1)首先我们证明对任何 $H \in S(NMC)$,存在 $H' \in S(I)$,有 $H \sim H'$。我们把每一个应用消息 m_i 与一个相应的物理时间 t_i 联系起来,满足 $m_i < m_j \Rightarrow t_i < t_j$,由于"$<$"是一个偏序关系,所以这一点可以实现,其中 t_i 是 S(I)系统中发送 m_i 并且立即传递到接收者处的时刻。对 H' 中的任一个用户进程 u_p,我们定义一个时钟历史函数 $C'(u_p,t')$:(i)对 u_p 中 m_i 的 Sent 或 Delv 事件(用 e_p^i 来表示),定义 $C'(u_p,t')=C(u_p,t)$,其中 t' 是与 m_i 相对应的时间,$C(u_p,t)$ 是 H 中事件 e_p^i 发生时的时钟值。(ii)对发生于 e_p^i 与 e_p^{i+1} 之间的其它事件或状态,我们按照"$\rightarrow$"偏序关系给一个介于 t_i 与 t_{i+1} 之间的时间 t',并且把 $C'(u_p,t')$ 定义为 H 中相应事件或状态的时钟值 $C(u_p,t)$,这样,$C'(u_p,t')$ 满足时钟条件,即 $t'_1 < t'_2 \Rightarrow C'(u_p,t'_1) \leqslant C'(u_p,t'_2)$。因此,$H=(C,Q,A,NMC)(\in S(NMC))$ 和 $H'=(C',Q',A',I)(\in S(I))$ 对任何的 $c \in N, u_p \in P$,有 $Q'(u_p,c)=Q(u_p,c)$,$A'(u_p,c)=A(u_p,c)$,即任给 H

$\in S(NMC)$，存在 $H\in S(I)$，有 $H\sim H'$。(2)反之，如果 S' 不满足 NMC，那么 $S'\subseteq S(I)$。设 $H'(\in S')$ 中存在一对应用消息 m 和 m'，它们满足 $(m<m')\wedge(m'<m)$。假定存在 $H''\in S(I)$ 有 $H'\sim H''$，那么对 H'' 中的应用消息 m 和 m' 分别给一个对应的物理时间 t 和 t'，对于 $H''\in S(I)$ 来说，满足 $m_i<m_j\Rightarrow t_i<t_j$，所以可推出 $t<t'$ 和 $t'<t$，这是自相矛盾。 □

下面我们将给出一个消息传递算法，并且证明该算法具有逻辑瞬时传递特性。

三、消息传递算法

该算法主要有三个步骤，即请求、允许、和发送。每一控制进程 c_p 有一队列 Q_p 来存放应用消息，直到它们被发送给另一个进程，或者被传送给用户进程 u_p。c_p 接收到 u_p 发来的 $Will\text{-}Send_p^q(m)$ 命令后，它首先同接收控制进程 c_q 进行协调，等待 c_q 发来"允许"信号，并且允许发送的应用消息已移到队列 Q_p 的前面时，就把该消息发给 c_q，并且通知 u_p 已完成 $Sent_p^q(m)$ 事件。

控制进程 c_q 接收到消息 m 后，如果该消息位于队列 Q_p 的前面，那么就通过 $Delv(m)$ 事件把该消息传给 u_q。用户进程 u_q 收到该消息后就改变它的状态，它还可能撤消一些已通过 $Will\text{-}send_q(m')$ 命令传给 c_q，但还没有通过 $Sent(m')$ 事件发送出去的应用消息。

每一个控制进程可产生时间戳(timestamps) T_p，这些时间戳基于进程的系统时钟。由于不同站点的进程一般具有不同的时钟值，所以我们把进程标识符 ID 附加在时钟 C_p 上，即 $T_p=(C_p,p)$。一个带有时间戳控制的消息记为"T_p :mes"。用户进程 u_p 和控制进程 c_p 每发生一个事件将增加该进程的时钟 C_p 值，在同一时钟期内最多只能有一个事件发生。每当从控制进程 c_q 处接收到消息"T_q :mes"时，控制进程 c_p 就把它的时钟设置为 $C_p'=\max\{C_p,C_q\}+1$。这里 $T_q=(C_q,q)$，C_p 是进程 u_p 的当前时钟值。队列按照时间戳的递增次序排列，也就是队头元素的时间戳最小。

用户进程和控制进程根据进程当前状态和所发生的事件，采取下列操作：

(1)发送请求信号：控制进程 c_p 从用户进程 u_p 处接到命令"$Will\text{-}send_p^q(m)$"后，就给控制进程 c_q 发送一个带有时间戳的请求消息"T_p :request"，并把消息 m 记为"T_p :q, m, requesting"存放入队列 Q_p 中，这里 $T_p=(C_p,p)$，C_p 为进程 u_p 的当前时钟值。

(2)发送允许信号：一接收到来自 c_q 的请求消息"T :request"后，(i)如果 $T>LT_p$，控制进程 c_p 就把"T :reserved"放入 Q_p，并传送一个允许信号"T :permitted(T)"给 c_q。(ii)如果 $T<LT_p$，控制进程 c_p 就把"T' :reserved"放入 Q_p，这里 $T'=(C_p,p)$，C_p 是进程 u_p 的当前时钟值，并传送一个允许消息"T' :permitted(T)"给 c_q。

这里变量 LT_p 的初值为 ∞，当某一消息从 c_p 发送给其它进程，或者从 c_p 传给 u_p 时，它的值就取 max{最近从 c_p 发送出的应用消息的时间戳，最近传给 u_p 的应用消息的时间戳}。上述第一种情况 $T>LT_p$ 表示自 c_i 发出请求消息起，控制进程 c_p 没有发送出任何消息，同时也没向 u_p 传递任何消息。但一般发生的是第二种情形，即自 c_q 发出请求消息起，到 c_p 接收到该消息为止，进程 u_p 已发生了一些其它事件，所以必须修改该请求消息的时间戳。

(3)接收允许信号：一从 c_i 处接收到允许消息"T' :permitted(T)"后，控制进程 c_p 就把队列中的"T :q, m, requesting"项改为"T' :q, m, permitted(T)"，并按照时间戳的递增次序对队列 Q_p 重新排列。

(4)发送应用消息：如果 Q_p 的头部是一个允许发送的应用消息，控制进程 c_p 就把该消息"T' :m"传送给控制进程 c_q，把 LT_p 设置为 T'，把"T' :q, m, permitted(T)"项从队列 Q_p 中移去，并通知用户进程 u_p 已完成 $Sent_p^q(m)$ 操作。这个过程重复执行，直到队头项不是允许发送的消息为止。

(5)接收应用消息：一接收到消息"T' :m"，那么控制进程 c_p 就把队列 Q_p 中的"T' :reserved"改为"T' :m"。如果接收到一个空消息"T' :null"，控制进程 c_p 只是把"T' :reserved"项从队列 Q_p 中删除(见下面说明)。

(6)传送应用消息：如果队列 Q_p 头部是一个已接收到的消息"T' :m"，那么 c_p 就把它传送给用户进程 u_p，把 LT_p 设置为 T'，并从队列中移去"T' :m"项。这个过程重复执行，直到队头项不是已接收到的消息为止。

Sent 或 Delv 事件都可能改变用户进程 u_p 的内部状态，从而 u_p 可能想修改或取消已通过 $Will\text{-}send_p(m')$"命令存入队列 Q_p，但还没有发送出去的消息。如果 u_p 只是把消息内容由 m' 改为 m"，那么 c_p 就把"T :q, m', *"改为"T :q, m", *"，这里 * 代表"requesting"或"permitted"。如果 u_p 想取消它发出的"$Will\text{-}send_p(m')$"，那么 c_p 就把"T :q, m', *"改为"T :q, null, *"。

四、对算法具有逻辑瞬时特性的证明

结论1 上述算法满足 NMC 特性，即上述算法

形成的消息传递环境在任何时候，不存在一对消息 m 和 m′，满足 m<m′ 和 m′<m。

证明：(反证法) 如果在某一时刻存在一对消息 m 和 m′ 满足 m<m′ 和 m′<m，那么也就是说存在一消息序列 $MS_i=(m_0,m_1,\cdots,m_{i-1},m_i(=m'),m_{i+1},\cdots,m_{k-1})$ 满足 $m_i<m_{i+1(\bmod k)}$，其中 $i\in\{0,1,2,\cdots,k-1\}$。下面我们要证明的是，如果 $m_i<m_{i+1}$，那么就有 $T_i<T_{i+1}$，这样，就有 $T_0<T_1<T_2<\cdots<T_{k-1}<T_0$，这是自相矛盾不成立的。

正如前面定义2.2所述，有(+，+)，(+，−)，(−，+)，(−，−)这四种关系可实现 $m_i<m_{i+1}$，我们就这四种情况下面分别加以证明。

(1)(−，+)情况：这时 m_i 的 Sent 事件发生于 m_{i+1} 的 Delv 事件之前，也就时说控制进程 c_{pi} 先发送消息 m_i，然后把 m_{i+1} 传给 u_{pi}。如果消息 m_{i+1} 的时间戳 T_{i-1} 在发送出消息 m_i 之前已在队列 Q_{pi} 中，那么由于发送消息 m_i 时，“T_i:p_{i-1},m_i,permitted”已在队列 Q_{pi} 的头部(见算法(4))，而队列 Q_{pi} 是按时间戳的递增偏序排列的，所以 $T_i<T_{i+1}$。否则的话，消息 m_{i+1} 是在发送出消息 m_i 之后才进入队列 Q_{pi} 中的，那么 m_{i+1} 的时间戳 $T_{i+1}\geqslant LT_{pi}>T_i$。

(2)(+，−)情况：这种情况下控制进程 c_{pi} 在发送消息 m_{i+1} 之前，已把消息 m_i 传给 u_{pi}。如果 c_{pi} 在传递消息 m_i 给 u_{pi} 之前，已传送了 m_{i+1} 的请求消息，那么由于“T_i:m_i”在队列 Q_{pi} 的头部(见算法(6))，所以 $T_i<T_{i+1}$。否则的话，消息 m_{i+1} 是在传送消息 m_i 之后才请求发送进入队列 Q_{pi} 中的，由于这时消息 m_{i+1} 得到的时间戳 T_{i+1} 不小于 c_{pi} 的当前时钟值，而该时钟值大于 T_i(见算法(1))，所以 $T_i<T_{i+1}$。

(3)(−，−)情况：根据算法(4)，位于队列 Q_{pi} 头部的发送给其它进程的消息具有最小的时钟值，所以 $T_i<T_{i+1}$。

(4)(+，+)情况：根据算法(6)，位于队列 Q_{pi} 头部的传送给 u_{pi} 的消息具有最小的时钟值，所以 $T_i<T_{i+1}$。

上面说明证明了该系统中不可能存在一对消息 m 和 m′ 满足 m<m′ 和 m′<m，即该消息传递系统具有 NMC 特性，再根据定理2.1，该消息传递系统具有逻辑瞬时特性。 □

结论2 该消息传递系统不会出现死锁。

证明：根据该算法，如果队列 Q_{pu} 头部元素是允许发送的消息项“T_u:p_k,m_a,permitted”，那么该消息马上就会被发送出去，并从队列中移去。如果队列 Q_{pu} 头部元素是已接收到的消息项“T_l:m'”，那么该消息马上就会被传送给 u_{pc}，并从队列中移去。如果队列 Q_{pa} 头部元素是请求发送项“T_l:q,m,requesting”，根据该算法的“请求、允许、发送”机制，该项最终会被允许发送，所以，我们只要考虑队列头部元素是“T_l:reserved”，而 c_{pc} 在收到 c_{p1} 发出的带有时间戳的消息“T_1:m_1”之前，不能发送已得到允许的消息“T_a:m_c”的情况，这里当然 $T_a>T_1$。类似地，c_{p1} 在收到 c_{p2} 发出的消息“T_2:m_2”之前，不能发送消息“T_1:m_1”($T_1>T_2$)，c_{p2} 在收到 c_{p3} 发出的消息“T_3:m_3”之前，不能发送消息“T_2:m_2”($T_2>T_3$)，…，c_{pi-1} 在收到 c_{pi} 发出的消息“T_i:m_i”之前，不能发送消息“T_{i-1}:m_{i-1}”($T_{i-1}>T_i$)，…，c_{pk} 在收到 c_{pu} 发出的消息“T_0:m_0”之前，不能发送消息“T_k:m_k”($T_k>T_0$)，只有这样构成循环回路时才会发生死锁，而这时就有 $T_0>T_1>T_2>T_3>\cdots>T_{i1}>T_i>\cdots>T_k>T_0$，这是自相矛盾不成立的。 □

参考文献

[1] R. Bagrodia, Synchronization of asynchronous process in CSP, ACM Trans. on Programming Languages and Syst., Vol. 11, No. 4, 1989

[2] Terunao Soneoka 等, Logically Instantaneous Message Passing in Asynchronous Distributed Systems, IEEE Trans. on Comput., Vol. 43, No. 5, 1994

[3] 王鼎兴等，一种实现并行计算的新主流技术-NOW，《小型微型计算机系统》，1995. 2

[4] 林洪等，PVM 与网络并行计算，《小型微型计算机系统》，1995. 2

[5] L. Lamport, Time, oclocks, and the ordering of events in a distributed system, CACM, Vol. 21, No. 7 1978

[6] K. Birman 等, Reliable communications in presence of failures, ACM Trans. Comput. Syst., Vol. 5, No. 1, 1987

[7] A. D. Birrel 等, Implementing remote procedure calls, ACM Trans. Comput. Syst., Vol. 2, No. 1, 1984

[8] K. J. Goldman, Highly concurrent logically synchronous multicast, Distributed Computing, Vol. 4, No. 4, 1991