对一个分布式终止探测算法的改进*

刘旭民, 苏运霖+

(暨南大学 计算机科学系,广东 广州 510632)

Improvement of a Distributed Termination Detection Algorithm

LIU Xu-Min, SU Yun-Lin+

(Department of Computer Science, Jinan University, Guangzhou 510632, China)

+ Corresponding author: Phn: 86-774-5837723, Fax: 86-774-5828752, E-mail: ylsu@gxuwz.edu.cn

http://www.jnu.edu.cn

Received 2001-07-04; Accepted 2002-05-23

Liu XM, Su YL. Improvement of a distributed termination detection algorithm. *Journal of Software*, 2003,14(1):49~53.

Abstract: DTD (distributed termination detection) is an important problem in the field of distributed algorithm research. If the termination of a computation can not be detected, the algorithm will lose its realistic significance. The credit-recovery algorithm proposed by Mattern is message optimal, but it is limited to centralized computation. In this paper, this algorithm is improved to be available in decentralized computation, which makes it more applicable to distributed environment.

Key words: credit-recovery; distributed termination detection; decentralized computing

摘 要: DTD(distributed termination detection)是分布式算法研究中的一个重要问题.如果不能探测计算的终止,算法就失去了其现实意义.Mattern 提出的 credit-recovery 算法实现了消息最优,但只局限于在集中式计算中应用.对其进行了改进,使其能够应用在非集中式计算中,以更适合分布式环境.

关键词: credit recovery;分布式终止探测;非集中式计算

中图法分类号: TP301 **文献标识码**: A

分布式终止探测(DTD)问题是分布式计算中的一个重要问题.其重要性主要表现在:(1) 只有正确判定计算是否终止,才能确定计算的结果是否最终结果,并判断计算的正确性,同时决定是否可以丢弃计算中所使用的变量;(2) 一个计算的死锁会导致它的终止,在这种情况下,系统需要对终止状态作出判断,然后重新开始计算.从 1980 年 Dijkstra 和 Schotlen 提出了第一个 DTD 算法[1]至今,这一领域的科学家们纷纷提出了许多解决方法.这些算法按消息传送方式,可以划分为波动算法和非波动算法两类.DTD 算法自身的特点决定了这些算法以波动算法为主流.然而,波动算法有着重复的本性,因此无法实现消息最优.1989 年,Mattern 提出了 credit-recovery 非波动算法[2],采用“搭肩”方式,将控制消息嵌入到基本的计算消息中,实现了消息最优,由此它被称为一个“很聪明”的算法[3].下面给出 Mattern 的 credit-recovery 算法.

var $state_p$: (active, passive) init if $p=p_0$ then active else passive;

* 第一作者简介: 刘旭民(1979－),女,辽宁辽阳人,硕士生,主要研究领域为分布式系统,分布式算法.

$cred_p$: fraction init if $p=p_0$ then 1 else 0;
ret: fraction init 0; for p_0 only

S_p: {$State_p$=active}
Begin send ⟨mes, $cred_p$/2⟩
End

R_p: (A message ⟨mes, c⟩ has arrived at p)
Begin receive ⟨mes, c⟩;
$state_p$:=active;
$cred_p$:=$cred_p$+c
End

I_p: ($state_p$=active)
Begin $state_p$:=passive;
send ⟨ret, $cred_p$⟩ to p_0;
$cred_p$:=0
End

A_{p_0}: {A ⟨ret,c⟩ message has arrived at p_0}
Begin receive ⟨ret,c⟩;
ret:=ret+c;
if ret=1 then Announce
End

在 Mattern 算法中,只有 p_0 是计算的启动者,可见它是针对集中式的计算而设计的.众所周知,非集中式计算更符合分布式系统的特点,因而它具有重要的地位.为此,本文对 Mattern 算法进行了改进,使其能够应用在非集中式计算中.

1 改进的 credit-recovery 算法

本算法是在 Mattern 的 credit-recovery 算法的基础上,针对分布式系统中的非集中式计算方式给出的.其计算图是一个类森林.其中计算的启动者集合 $Q=\{p_0,p_1,\ldots,p_{m-1}\}$分别对应于每棵类树的根结点.之所以称为类森林,是因为各个启动者在计算中建立起来的是类似于树的计算图,我们称之为类树.本算法的基本思想是对每棵类树的终止采用 credit-recovery 算法进行探测,之后对启动者集合形成的环形拓扑启动波动算法进行类森林的终止探测.具体思路如下:

对于每棵类树,将其中的每个传送中的基本消息和活动进程都分配一个 credit 值(在 Mattern 的算法中缩写为 cred,以下沿用此记号),cred 值在 0 和 1 之间.另外,每个启动者都拥有一个 return 值(在 Mattern 的算法中缩写为 ret,以下沿用此记号),即类树中的非根结点向根结点返回的 cred 值之和.它的初始值为 0.算法中对 cred 的分配要保证以下断言为不变量:

S1. 在每棵类树中,所有的活动进程和传送中的基本消息的 cred 值之和与这棵类树的根结点的 ret 值相加为 1.

S2. 所有传送中的基本消息的 cred 值都为正数.

S3. 所有活动的进程的 cred 值都为正数.

这样,对于一棵类树,设其根结点为 p.当 p 的 ret 值为 1 时,由 S1 得出,这棵类树中所有传送中的基本消息的 cred 值和所有活动的进程的 cred 值之和为 0,由 S2,S3 可知,这意味着类树中没有在传送中的基本消息和处于活动状态的进程.由此可以得出,这棵类树的计算已经终止了.设 $term_p$ 表示命题“根结点 p 所在类树的计算终止”,我们可以得到以下规则:

规则 1. $term_p$=true⇔ret_p=1,其中 ret_p 表示 p 的 ret 值.这就是说,p 所在类树的计算终止,当且仅当 p 的 ret 值

为 1.

规则 2. 当一个进程变成被动状态时,它将其 cred 值发送给其所在类树的根结点.根结点将这个值加到自己的 ret 值上.

在初始状态中,类树中只有根结点是活动的.它的 cred 值为 1,ret 值为 0.可以看出,初始状态满足 S1,S2, S3.为了保证在计算中维持这些不变量,我们需要分析在计算中出现的情况.首先,S2 要求一个基本消息的 cred 值必须是一个正数.基于它的发送者是活动的进程,而这个进程又有一个为正的 cred 值,这样,就可以将这个进程的 cred 值分一部分给这个消息.同时,为了区分来自不同类树的消息,进程要在发送消息的时候把自己的根结点标识嵌入到这个消息中.由此,我们提出规则 3.

规则 3. 当一个活动的进程发送一个消息时,它把其 cred 值分给自己和这个消息,并把它所在类树的根结点标识嵌入到这个消息中.

当一个进程被激活时,S3 要求得到一个为正的 cred 值.幸运的是,激活它的那条消息本身就带有一个为正的 cred 值,而这个消息已经结束了传送状态,不再需要拥有 cred 值,因此我们可以把它的 cred 值转给这个进程,并把这个进程连到消息所在的类树中,这样就可以在计算的过程中建立和修改类树的计算图.如果被激活的进程已经属于一棵类树,我们就要把这个进程从它原先的类树中删除,再连到这棵类树中.于是有以下规则:

规则 4. 当进程被一个消息激活时,这个消息将自己的 cred 值转给这个进程,并让这个进程指向所在类树的根结点.

在分布式计算中,还剩下一种情况没有提及,即一个已经处在活动状态的进程可能会接收到一个基本消息.这条消息有可能来自进程所在的类树,也有可能来自其他类树.在这种情况下,为了防止类树中的 cred 值流失和受到干扰,我们规定这个进程接收到消息以后,要把这个消息的 cred 值返回给消息所在类树的根结点.于是有以下规则:

规则 5. 当一个活动的进程接收到一条基本消息时,进程将消息的 cred 值返回给消息所在类树的根结点.

至此,类树的终止探测得以完整地实现.

在类森林的终止探测中,当环内的第 1 个结点 p_0 所在的类树探测到计算终止时,马上启动波动算法对其他根结点进行终止探测.具体做法是,p_0 向环内的下一个结点发送令牌,而接受令牌的结点必须满足它所在类树的计算已经终止这个条件才能拿到令牌,然后向下一个结点传送令牌.由此我们可以得到,以下断言 S4 为不变量.

S4. 在启动者集合形成的环内,当前拿到令牌的结点之前的所有结点都满足"所在类树的计算已经终止"这一条件.

形式化地表示 S4,我们有,在环内,设 t 为当前拿到令牌的结点下标,S4≡ $\forall i\ (t \geq i \geq 0)\ \ term_{p_i}$ =true.

当令牌再次传到 p_0 时,说明环内的所有结点都已经探测到所在类树的计算终止.因此,可以得出这个类森林的计算已经终止这一结论.从而得到以下规则:

规则 6. 当且仅当 $term_p$ =true 时,p 拿到令牌.

规则 7. 在令牌传送的过程中,令 t 为当前拿到令牌的结点下标,当 $p_t=p_0$ 时,Announce.即发布整个计算已经终止.

算法实现如下:

```
var   state_p: (active,passive)     init if p∈Q then active else passive;
      cred_p: fraction              init if p∈Q then 1 else 0;
      ret_p: fraction               init 0, only for p∈Q;
      term_p: Boolean               init false, only for p∈Q;
      root_p: Q                     init if p∈Q then p else udef;
S_p: {state_p=active} (* Rule 2 *)
   Begin  send ⟨mes,cred_p/2,root_p⟩;
          cred_p:=cred_p/2
   End
```

R_p: {A message $\langle mes,c,q\rangle$ has arrived at p}
Begin if $state_p$=passive then (* Rule 4 *)
Begin $state_p$:=active;
$cred_p$:=c;
$root_p$:=q;
End
else send $\langle ret, c\rangle$ to q (* Rule 5*)
I_p: {$state_p$=active}
Begin $state_p$:=passive;
send $\langle ret,cred_p\rangle$ to $root_p$ (* Rule 3 *);
$cred_p$:=0;
End
T_p: {A $\langle ret,c\rangle$ message has arrived at p}
Begin ret_p:=ret_p+c;
if ret=1 then
Begin $term_p$:=true (* Rule 1 *);
if p=p_0 then send $\langle tok\rangle$ to next_p
End
End
W_p: {A $\langle tok\rangle$ is arrived at p and $term_p$=true} (* Rule 6 *)
Begin if p=p_0 then Announce (* Rule 7 *)
else send $\langle tok\rangle$ to $next_p$
End

2 算法的证明

定理 1. *改进后的算法是一个正确的 DTD 算法.

证明:设 T 是以 p 为根结点的类树中的结点集合,在 T 中,因为算法满足了规则 1~规则 5,这意味着 S1∧S2∧S3 是不变量,其中:

S1≡1=($\sum_{\langle mes,c,q\rangle}c$)+($\sum_{q\in T}cred_p$)+($\sum_{\langle ret,c\rangle}c$)+$ret_p$;

S2≡∀$\langle mes,c,q\rangle$ in transit: c>0;

S3≡∀$q\in T$: ($state_q$=passive⇒$cred_p$=0)∧($state_q$=active⇒$cred_p$>0).

因此,当 ret_p=1 时,T 中既没有活动的进程,也没有传送中的基本消息,这就说明类树中的计算已经终止.

另外,算法满足了规则 6 和规则 7,这意味着 S4 也是不变量,S4≡∀i(t>i>0) $term_{p_i}$ = true.

因此,当 p_t=p_0 时,类森林中每棵类树的根结点都已经探测到计算终止,这意味着整个森林所有的类树的计算都已经终止,因此整个类森林的计算终止(假设一棵类树的根结点已经探测到本类树的计算终止,此时它又被来自另一棵类树上的消息激活,当出现这种情况时,这个根结点将被接到那棵类树上,但这并不影响原先类树的终止状态.$term_p$ 一旦被赋值 true 以后,就不会再被更改了). □

定理 2. 改进后的算法具有活性属性.

证明:当类树 T 中的计算终止以后,T 中的所有的进程都已经处于被动状态,基本消息也传送完毕(规则 4 保证如果这时有进程被来自其他类树的基本消息激活,它将被从 T 中删除掉,然后连到那棵类树上.因此,这种情况也不会影响 T 的计算终止).这时,通道中只剩下了控制消息$\langle ret,c\rangle$,而每次控制消息的接收都会减少一个传送中的消息,因此 T 的终止探测算法最终会达到终止状态.在这种状态下,T 中既没有基本消息$\langle mes,c,q\rangle$,也没有控制消息$\langle ret,c\rangle$,而且每个进程的 cred 值都为 0,于是,p 的 ret 值为 1,这样就探测到了类树的计算的终止.

当整个类森林的计算终止以后,类森林中所有的进程都已经处于被动状态,基本消息也已传送完毕.类森林中只剩下环内令牌$\langle tok\rangle$的传送,因为当根结点 p 探测到本类树的终止以后就可以拿到令牌,因此,令牌将被环内

的结点连续地传送下去,最终传送到 p_0.波动算法达到终止,并且探测到了整个计算的终止. □

3 特性分析

3.1 消息复杂度

在基本消息传送的过程中,cred 值和 $root_p$ 信息都是嵌入到基本消息中的,另外,每个非根结点在接收到基本消息时,如果它是被这个消息激活的话,那么它在变成被动的时候,需要向其所在类树的根结点发送消息⟨*ret*,*c*⟩,如果这个结点在接受消息之前已经处于活动状态,那么它就要向这条消息所来自的类树的根结点发送消息⟨*ret*,*c*⟩.在初始状态中,所有非根结点都处于被动状态,这意味着每条基本消息的接收都要对应着一个消息⟨*ret*,*c*⟩的发送.在之后进行的环形波动算法中,我们可以知道,令牌只需走一圈就可以判定整个类森林的终止,因此,计算中共交换了 $M+m$ 条控制消息(M 是基本计算中交换的消息数,m 是计算的启动者数目),从而算法保证了消息最优性.

3.2 算法对称性

在类树的终止探测算法中,S_p,I_p,R_p 对于所有结点都是对称的.而在波动算法中,T_p,W_p 对于环内结点也是对称的.因此本算法大体上是对称算法.

3.3 消息传送方式

由算法可以看出,在计算和探测过程中,消息传送采用的是异步传送方式,这更符合分布式系统的特点,因而应用也更加广泛.

3.4 通信容错性

在算法中,各个结点需要向其他结点发送消息,非根结点又需要向根结点发送消息,因此它要求网络通信是无向的、连通的.另外,算法并没有考虑容错性问题,它假设通信线路都是正常的,且无通信差错.

3.5 结点的了解

在类树的终止探测过程中,各个结点不需要了解其他结点的有关知识,而对根结点知识的获取是通过消息的传送来实现的.在环形波动算法中,各个结点则需要知道环内下一个结点的知识,才能将令牌传送下去.

综上所述,本算法结合了波动算法和非波动算法的优点,基本上实现了消息最优、算法对称和异步的消息传送方式,因此它在分布式系统中更加适用.

4 结束语

Credit-Recovery 算法的一个突出问题是,对发送基本消息的活动进程的 cred 值的分配是对其一分为二.如果 cred 值被存储在一个固定字节的单元里,就存在一个最小正数的 cred 值,它将无法被一分为二.如果必须将它一分为二,则基本计算就要被暂时挂起,等到根结点从 ret 值中减去一部分赋给这个 cred 时为止(ret 值有可能因此变成负数),这就违背了 DTD 算法不能影响基本计算的初衷.另外,对 cred 值的分配会导致当结点增多时,大大增加算法的空间复杂度.针对这个问题,1995 年 Tseng 提出了一种 credit 的编码方案,使这个问题得以解决[4].本文不再对这个问题进行讨论.

References:

[1] Tel G. Introduction to Distributed Algorithms. 2nd ed., Cambridge: Cambridge University Press, 1994. 270~276.

[2] Mattern F. Global quiescence detection based on credit distribution and recovery. Information Processing Letters, 1989,30(4): 195~200.

[3] Matocha J, Camp T. A taxonomy of distributed termination detection algorithms. Journal of Systems and Software, 1998,43: 207~211.

[4] Tseng, Y. Detecting termination by weight-throwing in a faulty distributed system. Journal of Parallel and Distributed Computing, 1995,25:7~15.