基于心跳故障检测器的无阻塞弱原子提交问题解决方案

蔡　强[1]　赵致琢[2]　倪子伟[2]
[1](厦门大学软件学院　福建 厦门 361005)
[2](厦门大学计算机科学系　福建 厦门 361005)

摘　要　在有故障发生的情况下，使用不可靠的故障检测器无法解决无阻塞原子提交问题。本文减弱无阻塞原子提交问题的非平凡性条件，得到一个较弱的问题，再用一个扩展的心跳故障检测器在包含进程故障和链路故障的异步消息传递系统中解决弱化后的问题。

关键词　心跳故障检测器　无阻塞原子提交　合意　可归约

SOLUTION TO NON-BLOCKING WEAK ATOMIC COMMITMENT PROBLEM BASED ON HEARTBEAT FAILURE DETECTOR

Cai Qiang[1]　Zhao Zhizhuo[2]　Ni Ziwei[2]
[1](Software School, Xiamen University, Xiamen Fujian 361005, China)
[2](Computer Science Department, Xiamen University, Xiamen Fujian 361005, China)

Abstract　It is impossible to solve non-blocking atomic commitment problem using unreliable failure detector when failure occurs. So we weak the nontriviallity condition of non-blocking atomic commitment problem and solve the weakened problem using extended heartbeat failure detector in asynchronous message-passing system with process and link failures.

Keywords　Heartbeat failure detector　Non-blocking atomic commitment　Consensus　Reducible

1　引　言

心跳故障检测器可以用来求解包含进程故障和链路故障的异步消息传递系统中的静止可靠通信问题。原子提交问题要求对某个事务，所有参与进程都有共同的结果，即或者全部输出提交,或者全部输出中止。

考虑实际应用中通常采用的无阻塞原子提交问题，它要求即使某些进程发生故障，正确进程仍能判定共同的结果。然而 R.Guerraoui 已经证明，在有故障发生的情况下，使用不可靠的故障检测器无法解决无阻塞原子提交问题。如果减弱无阻塞原子提交问题的非平凡性条件，可以得到一个较弱的问题，我们称之为无阻塞弱原子提交问题，那么，该问题可以用不可靠的故障检测器解决，而且在实际的事务处理系统中，这种减弱的非平凡性条件已经足够了。

目前，解决无阻塞弱原子提交问题大多采用传统的不可靠故障检测器，在同步系统中或在包含进程故障但具有可靠链路的异步系统中实现。

针对异步系统可能同时存在进程故障和链路故障的情况，我们采用心跳故障检测器在包含进程故障和链路故障的异步消息传递系统中解决无阻塞弱原子提交问题。为此，我们先扩展心跳故障检测器的定义，然后用扩展后的心跳故障检测器把无阻塞弱原子提交问题归约到一致合意问题，最后用扩展后的心跳故障检测器实现一致合意问题。

2　心跳（Heartbeat）故障检测器 HB

HB 最早由康奈尔大学的 Aguilera 等人在 1997 年提出。简单地说，进程 p 的心跳故障检测器模块输出一个计数器向量，向量的每个分量对应 p 的一个邻居结点 q。如果 q 没有发生故障，那么它在 p 的计数器可以无限增加；如果 q 发生故障，那么它在 p 的计数器最终停止增加。HB 的提出是基于以下思想：每个进程周期性地发送“I-am-alive”的消息(即一次“心跳”)，收到该消息的每个进程则增加相应发送进程的计数器。

上述心跳故障检测器 HB 中，每个进程 p 的模块只能输出 p 的邻居的心跳值。为简化本文的算法，我们给出 HB 的一个扩展，使每个进程 p 的故障检测器模块能够输出系统中所有进程的心跳值。下面给出 HB 的扩展定义和实现。

2.1　HB 的扩展定义

心跳故障检测器 D 在每个进程 p 的输出是序列(p_1,n_1)，$(p_2,n_2),\ldots$，(p_n,n_n)，其中 $p_1,p_2,\ldots,p_n$ 是系统中所有的进程，每个 n_j 是个非负整数。当 p_j 没有发生故障时，n_j 增加，否则，n_j 停止增加。称 n_j 为 p_j 在 p 的心跳值。D 在时刻 t 在 p 的输出 H(p,t) 可以看作下标集合为$\{p_1,p_2,\ldots,p_k\}$的一个向量，因此，$H(p,t)[p_j]=n_j$。 p_j 在 p 的心跳序列定义为随时间而变化的 p_j 在 p 的心跳值

收稿日期:2005-06-15。福建省自然科学基金项目(A0310007)。蔡强，硕士生，主研领域：计算模型，分布式算法。

的序列。D 满足以下属性：

(1) HB-完全性

在每个正确进程 p，系统中每个故障进程 q 的心跳序列都是有界的。

(2) HB-精确性

1) 在每个进程 p，系统中每个进程 q 的心跳序列都是非递减的。

2) 在每个正确进程 p，系统中每个正确进程 q 的心跳序列都是无界的。

下面，我们仍用 HB 表示上述定义的扩展后的心跳故障检测器。

2.2 扩展 HB 的实现

现在给出扩展心跳故障检测器的具体实现，如表 1 所示。

表 1　扩展 HB 的实现

```
对每个进程 p:
initialization:
  for all q∈Π do D_p[q]:=0        //Π代表系统中的所有进程
cobegin
  ||Task 1:
    repeat periodically
      D_p[p]:=D_p[p]+1
      for all q∈neighbor(p) do send_{p,q}(HEARTBEAT,p)
  ||Task 2:
    upon receive_{p,q}(HEARTBEAT,path) do
      for all q∈Π such that q appears after p in path do
        D_p[q]=D_p[q]+1
      path=path.p
      for all q∈neighbor(p) and q appears at most once in path
do
        send_{p,q}(HEARTBEAT,path)
coend
```

可以看到，每个进程 p 执行两个并发任务。在任务 1 中，进程 p 周期性地增加自己的心跳值，并向它的所有邻居发送消息(HEARTBEAT,p)。在任务 2 中，进程 p 处理接收到的形如(HEARTBEAT,path)的消息。当进程 p 从进程 q 收到这个消息时，p 增加所有在 path 中出现在 p 之后的进程的心跳值然后 p 把自己添加到 path 的末尾，并把消息(HEARTBEAT,path)转发给在 path 中至多出现一次的所有邻居。

3 用 HB 实现无阻塞弱原子提交问题

3.1 问题描述

原子提交(AC,Atomic Commitment)问题要求对某个事务，所有的参与进程都有共同的判定结果，即或者全部输出提交(commit)，或者全部输出中止(abort)，我们称之为一个进程 AC-判定提交或 AC-判定中止。一个进程的 AC-判定结果依赖于所有进程提供的投票值(yes 或 no)。一个进程能 AC-判定提交，仅当所有的进程投票 yes。为了排除所有进程总是判定中止的情况，一般要求在所有进程投票值为 yes 且没有进程故障的情况下必须判定提交。

当系统中有故障发生时，考虑无阻塞原子提交(NB-AC,Non-Blocking Atomic Commitment)问题，其中，即使某些进程发生故障，正确进程仍能 AC-判定。NB-AC 问题要求满足下列条件：

(1) 一致同意（Uniform-Agreement）所有发生判定的进程判定相同的值。

(2) 一致有效性（Uniform-Validity） 如果一个进程 AC-判定提交，那么所有的进程都已投票 yes。

(3) 终止性(Termination) 每个正确进程最终都 AC-判定。

(4) 非平凡性（Nontriviallity） 如果所有的进程都投票 yes, 并且没有进程发生故障，那么每个正确进程最终 AC-判定提交。

已经证明，在有故障发生的情况下，使用不可靠的故障检测器无法解决 NB-AC 问题。这是因为 NB-AC 问题的非平凡性要求进程知道关于故障的精确信息，而不可靠的故障检测器显然无法给相应进程提供这样的信息。

然而，如果我们减弱 NB-AC 问题的非平凡性条件，从而得到一个较 NB-AC 弱的问题 NB-WAC(Non-Blocking Weak Atomic Commitment)，那么该问题可以用不可靠的故障检测器解决，并且在实际的事务处理系统中，这种减弱的非平凡性条件已经足够了。NB-WAC 问题除了要求满足 NB-AC 问题的一致同意、一致有效性、终止性外，还要求满足如下的非平凡性：如果所有的进程都投票 yes，并且没有进程曾被怀疑过，那么每个正确进程最终 AC-判定提交。

3.2 用 HB 求解 NB-WAC 问题

为了使用 HB 求解 NB-WAC 问题，我们先用 HB 将 NB-WAC 问题归约到一致合意问题，再用 HB 求解一致合意问题。实现的算法不仅可以容忍进程故障，还可以容忍链路故障。

在合意问题(Consensus Problem)中，每个参与进程提出一个输入值，要求正确进程最终判定某个相同的输出值。合意问题要求满足以下条件：

(1) 一致有效性(Uniform-Validity) 如果一个进程判定 v, 那么 v 必为某个进程的输入。

(2) 终止性(Termination) 所有正确进程最终都判定。

(3) 同意(Agreement) 所有发生判定的正确进程判定相同的值。

一致合意问题(Uniform Consensus Problem)除了满足合意问题的一致有效性和终止性外，还必须满足如下的一致同意条件(Uniform-Agreement)：所有发生判定的进程判定相同的值。

(4) 用 HB 将 NB-WAC 问题归约到一致合意问题

算法如表 2 所示。进程要么发生故障，要么调用函数 NBWAC(votei)。用 votei 表示进程 pi 的投票值，用函数 uniformConsensus() 表示一致合意算法。

表 2　用 HB 将 NB-WAC 问题归约到一致合意问题

```
Function NBWAC(vote_i)
  broadcast (vote_i);
  for j: =1 to n
    wait until [received(p_j,vote_j) or D_i[p_j] stop increasing];
    if D_i[p_j] stop increasing or vote_j =no then
      outcome_i:=uniformConsensus (abort);
      return outcome_i;
outcome_i:=uniformConsensus (commit);
return outcome_i;
```

定理 1　表 2 所示算法用 HB 将 NB-WAC 问题归约到一致合意问题。

证明：我们分别证明表 2 所示的算法满足 NB-WAC 的四个条件。

1) 一致同意。可以看出，所有发生 AC-判定 outcome 的进程都已通过调用函数 uniformConsensus() 判定 outcome，根据一致合意问题的一致同意性质，可知所有发生判定的进程判定相同的值。

2) 一致有效性。可以看出，所有发生 AC-判定 outcome 的进程都已通过调用函数 uniformConsensus()判定 outcome，根据

一致合意问题的一致有效性性质，如果一个进程判定 commit，那么 commit 必为某个进程的输入，即 p_i 会调用 uniformConsensus (commit)，而算法能运行到这一步，说明 p_i 已经从所有进程收到投票值 yes。

3) 终止性。对每个正确进程 p，考虑 3 种情况。

i) p 从所有进程收到投票 yes。那么 p 将调用 uniformConsensus(commit)。

ii) p 收到 k(k>0)个值为 no 的投票。那么 p 将调用 uniformConsensus(abort)。

iii)否则，由于每个正确进程都会广播自己的投票值并且 broadcast (votei)是可靠的，因此，p 最终将停止增加某些故障进程在 p 的计数器的值，从而调用 uniformConsensus(abort)。总之，每个正确进程最终都将调用函数 uniformConsensus()，根据一致合意问题的终止性性质，每个正确进程最终都 AC-判定。

4) 非平凡性。如果所有的进程都投票 yes,并且没有进程曾被怀疑过，那么每个进程都会调用函数 uniformConsensus (commit)，根据一致合意问题的一致有效性和终止性性质，每个正确进程最终 AC-判定提交。

证毕。

(5) 用 HB 求解一致合意问题

Fischer、Lynch 和 Paterson 已经证明，在异步系统中，即使只有一个进程发生故障并且网络是完全连通的，合意问题也是不可解的。为了解决这个问题，Chandra 和 Toueg 引入了不可靠故障检测器。现在，我们采用 HB 来求解一致合意问题。算法如表 3 所示。Vp[q]是 p 目前对 q 的提议值 vq 的估计，算法由 3 个阶段组成，在第一阶段，进程执行 n-1 个异步轮(rp 表示 p 的当前轮号)。进程先执行一个广播 broadcast(rp,△p,p)，然后等待，直到收到每个未发生故障的进程的第 rp 轮的消息。然后，如果 p 在第 rp 轮第一次收到 q 的提议值 vq，就把△p[k]赋值为 vq。在第二阶段，正确进程最终基于每个进程的提议值达成一个一致的向量 V，向量的第 i 个元素要么是 pi 的提议值，要么是△。在第三阶段，每个正确进程对该向量的第一个非△元素发生判定。

定理 2 表 3 所示算法用 HB 求解一致合意问题。

证明： 分别证明表 3 所示算法满足一致有效性、终止性和一致同意即可。定理的证明与采用其它不可靠故障检测器求解一致合意问题的证明类似。可以参照参考文献[2]中的用 S 类故障检测器求解一致性问题的证明。但要注意此处采用由 HB 实现的可靠广播来发送消息。

证毕。

表 3 用 HB 求解一致合意问题

```
每个进程 p 如下执行：
procedure propose(vp)
Vp :=(△, △, …, △)
Vp[p]:= vp
△p:=Vp
Phase1: {1≤rp≤n-1}
for rp:=1 to n-1
broadcast(rp, △p, p)
wait until [∀ q: received(rp, △q, q) or Dp[q] stop increasing]
msgsp[rp]:={(rp, △q, q) | received(rp, △q, q)}
    △p:=(△, △, …, △)
for k:=1 to n
      if Vp[k]=△ and ∃(rp, △q, q)∈msgsp[rp] with △q[k]≠△
then
          Vp[k]:=△q[k]
          △p[k]:=△q[k]
Phase2: broadcast (Vp)
  wait until [∀ q: received Vq or Dp[q] stop increasing]
lastmsgsp:={ Vq | received Vq}
for k=1 to n
if ∃Vq∈lastmsgsp  with Vq[k]=△ then Vp[k]:=△
Phase3: decide (first non-△ component of Vp)
```

4 结 论

在发生故障的情况下，使用不可靠的故障检测器无法解决 NB-AC 问题，所以我们减弱它的非平凡性条件，得到 NB-WAC 问题，然后用 HB 将它归约到一致合意问题，再用 HB 求解一致合意问题，从而完成对 NB-WAC 问题的求解。使用 HB 进行求解，能够在包含进程故障和链路故障的异步消息传递系统中解决 NB-WAC 问题，而目前所知道的其它传统的故障检测器只能解决仅包含进程故障的 NB-WAC 问题，这是本文的主要贡献。另外，本文的方法适用于每对不同的正确进程都通过一条公平通道连接的情况，如何在分块网络中解决类似的问题，是我们下一步的工作重点。

参 考 文 献

[1] M.K.Aguilera, W.Chen and S.Toueg, Heartbeat: a timeout-free failure detector for quiescent reliable communication, In: Proceedings of the 11th International Workshop on Distributed Algorithms, Berlin: Springer, 1997, 126~140.

[2] T.D.Chandra and S.Toueg, Unreliable failure detectors for reliable distributed systems, Journal of the ACM, 1996, 43(2):225~267.

[3] R.Guerraoui, M.Larrea and A.Schiper, Non-blocking atomic commitment with an unreliable failure detector, In: Proceedings of the 14th IEEE symposium on Reliable Distributed Systems, Washington: IEEE Computer Society, 1995.

[4] R.Guerraoui, Revisiting the relationship between non-blocking atomic commitment and consensus, In: Proceedings of the 9th International Workshop on Distributed Algorithms, Lemont: Springer, 1995, 87~100.

[5] B.Coan and J.Welch, Transaction commit in a realistic timing model, Distributed Computing, 1990, 4(2): 87~103.

[6] M.J.Fischer, N.A.Lynch, and M.S.Paterson, Impossibility of distributed consensus with one faulty process, Journal of the ACM, 1985, 32(2):374~382.