学校编码：10384　　分类号_____密级_
学　　号：200340005　　UDC _____

厦门大学

硕　士　学　位　论　文

基于心跳故障检测器的原子提交问题解决方案

Solution to Atomic Commitment Problem Based on Heartbeat Failure Detector

蔡　强

指导教师姓名：　赵致琢　教授
专　业　名　称：　计　算　机　应　用
论文提交日期：　2006　年　4　月
论文答辩时间：　2006　年　　月
学位授予日期：　2006　年　　月

答辩委员会主席：_____
评　　阅　　人：_____

2006 年 4 月

厦门大学学位论文原创性声明

兹呈交的学位论文，是本人在导师指导下独立完成的研究成果。本人在论文写作中参考的其他个人或集体的研究成果，均在文中以明确方式标明。本人依法享有和承担由此论文产生的权利和责任。

声明人（签名）：

年　　月　　日

厦门大学学位论文著作权使用声明

本人完全了解厦门大学有关保留、使用学位论文的规定。厦门大学有权保留并向国家主管部门或其指定机构送交论文的纸质版和电子版，有权将学位论文用于非赢利目的的少量复制并允许论文进入学校图书馆被查阅，有权将学位论文的内容编入有关数据库进行检索，有权将学位论文的标题和摘要汇编出版。保密的学位论文在解密后适用本规定。

本学位论文属于

1、保密（　），在　　年解密后适用本授权书。

2、不保密（√）

（请在以上相应括号内打“√”）

作者签名：　　日期：　　年　月　日

导师签名：　　日期：　　年　月　日

摘 要

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。不像通常采用超时技术的传统故障检测器，心跳故障检测器不使用超时技术，而采用计数器。

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。原因是无阻塞原子提交问题的非平凡性要求进程必须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，因为不可靠的故障检测器可能提供错误的信息。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最好的方法是使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且具有可靠链路的异步系统中实现，但所得到的算法不是静止的。

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分别在包含进程故障和链路故障的一般网络和分块网络中得到解决无阻塞弱原子提交问题的静止算法。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实现静止可靠通信，再用静止可靠通信求解一致合意问题，最后把无阻塞弱原子提交问题归约到一致合意问题，从而完成对无阻塞弱原子提交问题的求解。

关键词：心跳；故障检测；原子提交

Abstract

Heartbeat failure detector (HB) can be used to solve quiescent reliable communication problems in asynchronous message-passing systems with process and link failures. Unlike traditional failure detectors which use timeouts, HB does not use timeouts, but use counters.

Atomic Commitment (AC) problem requires the participants to agree on an outcome for the transaction: commit or abort. In real-world we commonly use Non-blocking Atomic Commitment (NB-AC), which requires that every correct participant eventually reach an outcome despite the failure of other participants. But R.Guerraoui had proved that when failure occurs, it is impossible to solve NB-AC with unreliable failure detectors. The difficulty in solving NB-AC is that its Non-triviality requires precise knowledge about failures, but unreliable failure detectors cannot provide it, because they may provide error messages. Nevertheless, with a weaker non-triviality, we define a problem weaker than NB-AC, called NB-WAC, which is solvable with unreliable failure detectors, and in fact it is adequate in real-world transactional systems.

Up to now, the best way to solve NB-WAC is using traditional unreliable failure detectors, in synchronous systems or in asynchronous systems with process failures but reliable links, and, the algorithms are not quiescent.

In asynchronous systems with process and link failures, we use HB to get quiescent solutions for NB-WAC in general and partitionable networks. To do so, first, we extend the definition of HB, then we use it to implement quiescent reliable communication, and then we use quiescent reliable communication to solve uniform consensus problem. Finally, we reduce the NB-WAC problem to uniform consensus problem and finish.

Key words：HeartBeat；Failure Detect；Atomic Commitment

目 录

Contents

第一章　引 言

合意问题（Consensus Problem）是指如何在系统中使没有故障的进程形成一个一致的结果，以维护系统的性能和完整性。合意问题是许多分布式计算的核心问题。然而，Fischer、Lynch和Paterson指出：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[1]。FLP不可能性结果导致了对问题的更弱表述和更强模型，前者的例子有弱协同和随机化，后者包括引入同步或使用故障检测机制。现在普遍认为使用故障检测机制是最好的办法，因为它最简单并且最具一般性。

1.1 故障检测简介

故障检测是 1991 年由康奈尔大学的Chandra和Toueg研究小组最先提出的，并表述为一种抽象机制[2]。在故障检测模型中，每个进程都有一个自己的局部故障检测器模块，负责监视系统中的进程，并且维护一个它怀疑已经发生故障的进程列表。故障检测器可以看作是一种分布式谕示，它为每个进程提供关于进程故障的提示，尽管这种提示有时候可能是不正确的。

下面，我们简单介绍故障检测中的一些基本概念和基础知识。

1.1.1 故障模型

实际的分布式系统总是会出现各种类型的故障。比如说进程失灵、网络互连中断、消息丢失与复制，甚至是故障进程的恶意行为。我们希望系统在出现故障时仍然能正确执行算法，这就要求设计的算法具有容错计算的能力。但是，如果希望算法能容忍任何数量和任何类型的故障，那对算法的要求就太苛刻了，我们需要的是能容忍指定数量的某些类型故障的算法。

如果一个进程总是按照算法的要求执行操作，则称这个进程正确地执行。一个进程是正确的，如果它正确地执行并且一直可以这样进行下去，除非执行终止。一般地说，有四类在破坏性方面逐次递增的故障模型[2]（Failure Models）。

（1）失灵模型（Crash Model）：进程正确地执行，但是，在某个时刻停止，并且停止之后不会重新启动。

（2）遗漏模型（Omission Model）：在发送消息的情况下，没有把某条消息放入接受方的缓冲区中；在接收的情况下，接收者没有把缓冲区中所有的消息都取走。遗漏也可以解释为跳过算法中的某一个应该执行而没有执行的步骤。

（3）带身份验证的拜占庭模型（Authenticated Byzantine Model）：进程的一种任意性的行为。但是，消息可以附上发送进程的标识（签名），而且这个标识其他进程是不能更改的。

（4）拜占庭模型（Byzantine Model）：进程的一种任意性行为。没有签名机制，但可以假定消息的接收者知道发送者的标识。

1.1.2 故障检测器模型

我们考虑没有定时假设的异步消息传递的分布式系统。特别地，我们不考虑消息传递的时间延迟和进程的相对处理速度。进程通过在网络中传递消息与其它进程进行通信，系统可能发生进程失灵故障和链路丢失消息的故障。用有向图 $G=(\Pi,\wedge)$表示网络，其中$\Pi=\{1,\ldots,n\}$表示进程集合，$\wedge \subseteq \Pi \times \Pi$表示链路集合。为了表述方便，假定存在一个全局时钟，这个时钟是虚构的，进程无需访问它。假设时钟的值域Γ为自然数集。

（1）进程故障模式

进程故障模式是一个函数F_P: $\Gamma \rightarrow 2^{\Pi}$（$2^{\Pi}$表示进程的幂集），$F_P(t)$表示在时刻t失灵的进程的集合，$t \in \Gamma$。进程一旦失灵就不能恢复。因此，$t_1 \leqslant t_2$蕴含$F_P(t_1) \subseteq F_P(t_2)$。我们用$Crashed(F_P)=\cup_{t \in \Gamma} F_P(t)$和$Correct(F_P)=\Pi - Crashed(F_P)$分别表示失灵进程集合和正确进程集合。如果$p \in Crashed(F_P)$,就称p在$F_p$中失灵。如果$p \in Correct(F_P)$,就称p在$F_p$中是正确的。

（2）链路和链路故障模式

我们用 $p \rightarrow q$ 表示从进程 p 到进程 q 之间存在一条链路，如果 $p \neq q$,那么 q 称为 p 的一个邻居。p 的邻居集合用 neighbor(p)表示。

对应每条链路$p \rightarrow q$有两个原语: $send_{p,q}(m)$和$receive_{q,p}(m)$。如果进程p调用$send_{p,q}(m)$，我们称进程p发送消息m给进程q。如果p是正确的，那么它最终会从该调用返回。允许进程通过同一链路多次发送同一条消息。如果q从$receive_{q,p}(m)$的执行中返回，我们称进程q从进程p收到消息m。

网络中的每条链路都满足完整性，即：对所有的 $k \geqslant 1$,如果进程 q 从进程 p

接收消息 m 的次数为 k，那么，之前进程 p 发送消息 m 给进程 q 的次数至少是 k。也就是说，完整性假设保证了网络中的链路不可能自己产生消息。

称一条链路$p \rightarrow q$是公平的，如果$send_{p,q}()$和$receive_{q,p}()$除了满足完整性之外，还满足公平性，即: 如果进程q是正确的并且进程p发送消息m给进程q的次数是无穷多次，那么进程q从进程p接收消息m的次数也是无穷多次[2]。公平性保证了公平链路最终能传送任何经由该链路重复发送的消息。

称一条通道$(p_1,\ldots,p_k)$是公平的，如果进程$p_1,\ldots,p_k$都是正确的并且链路$p_1 \rightarrow p_2,\ldots,p_{k-1} \rightarrow p_k$都是公平的。

链路故障模式是一个函数F_L：$\Gamma \rightarrow 2^{\wedge}$（$2^{\wedge}$表示链路的幂集），$F_L(t)$表示在时刻t失灵的链路的集合，$t \in \Gamma$。链路一旦失灵就不能恢复。类似地，我们用$Crashed(F_L)=\cup_{t \in \Gamma} F_L(t)$和$Correct(F_L)=\Pi - Crashed(F_L)$分别表示失灵链路集合和正确链路集合。我们假定网络中的链路或者是公平的，或者发生失灵故障，即停止传送消息。如果$p \rightarrow q \in Crashed(F_L)$，就称$p \rightarrow q$在$F_L$中是失灵的，此时进程q只能从进程p接收到有限次消息。如果$p \rightarrow q \notin Crashed(F_L)$，就称$p \rightarrow q$在$F_L$中是公平的。

下面，我们用故障模式$F=(F_P,F_L)$表示进程故障模式和链路故障模式。

(3) 故障检测器和故障检测器类

故障检测器（Failure Detectors）是一种模块，每个进程都可以在任何时刻访问本地的故障检测器模块以获得关于执行中发生的故障模式的信息（可能是不正确的）。故障检测器历史是一个函数H：$\Pi \times \Gamma \rightarrow 2^{\Pi}$，H(p,t)是进程p的故障检测器模块在时刻t的输出值，表示p在时刻t怀疑的进程的集合。不同进程的故障检测器模块在时刻t的输出值可能是不同的，即：如果$p \neq q$，那么$H(p,t) \neq H(q,t)$是可能的。非形式地说，一个故障检测器D提供了发生在一次执行中的关于故障模式F的信息（可能是不正确的）。形式上，一个故障检测器是一个函数，它将故障模式F映射到故障检测器历史的一个集合D(F)。因此，D(F)表示对故障模式F，故障检测器D所允许的可能的故障检测器历史的集合。

为使故障检测器可用，在它的输出（故障检测器历史）和输入（故障模式）之间必定存在一种关系。通常的要求总是具有两种类型，即完全性（故障检测器将怀疑失灵进程）和精确性（故障检测器将不怀疑正确进程）。完全性（Completeness）限制了受到怀疑的进程集合的下界，精确性（Accuracy）则限

制了受到怀疑的进程集合的上界。

有两种形式的完全性：

1）强完全性（Strong Completeness）

如果每个失灵进程最终受到每个正确进程的怀疑，则称故障检测器 D 是强完全的。形式地，D 满足强完全性，如果：

$$\forall F, \forall H \in D(F), \exists t \in \Gamma, \forall p \in Crashed(F), \forall q \in Correct(F), \forall t' \geq t : p \in H(q, t')$$

2）弱完全性（Weak Completeness）

如果每个失灵进程最终受到某个正确进程的怀疑，则称故障检测器 D 是弱完全的。形式地，D 满足弱完全性，如果：

$$\forall F, \forall H \in D(F), \exists t \in \Gamma, \forall p \in Crashed(F), \exists q \in Correct(F), \forall t' \geq t : p \in H(q, t')$$

但是，系统如果只满足完全性性质并没有太大的意义，因为它不一定能提供有关故障的任何信息。例如，考虑如下的故障检测器：系统中的每个进程永远怀疑所有的其他进程。显然，这样的故障检测器满足强完全性性质，但是它没能提供关于故障的任何信息，因此并没有多大用处。为使故障检测器有用，还必须满足某些限制其犯错行为的精确性性质。有四种形式的精确性：

1）强精确性（Strong Accuracy）

如果进程没有失灵，就没有进程曾经受到过怀疑，则称故障检测器 D 是强精确的。形式地，D 满足强精确性，如果：

$$\forall F, \forall H \in D(F), \forall t \in \Gamma, \forall p, q \in \Pi - F(t) : p \notin H(q, t)$$

2）弱精确性（Weak Accuracy）

如果存在某个正确进程从未受到过怀疑，则称故障检测器 D 是弱精确的。形式地，D 满足弱精确性，如果：

$$\forall F, \forall H \in D(F), \exists p \in Correct(F), \forall t \in \Gamma, \forall q \in \Pi - F(t) : p \notin H(q, t)$$

3）最终强精确性（Eventual Strong Accuracy）

如果存在一个时刻，之后，没有正确进程受到任何正确进程的怀疑，则称故障检测器 D 是最终强精确的。形式地，D 满足最终强精确性，如果：

$$\forall F, \forall H \in D(F), \exists t \in \Gamma, \forall t' \geq t, \forall p, q \in Correct(F) : p \notin H(q, t')$$

4）最终弱精确性（Eventual Weak Accuracy）

如果存在一个时刻，之后，某个正确进程不受任何正确进程的怀疑，则称故障检测器 D 是最终弱精确的。形式地，D 满足最终弱精确性，如果：