使用 π-演算验证两阶段提交协议*)

张 帆[1] 李舟军[2] 孙 云[1]

(国防科技大学计算机学院 长沙 410073)[1] (北京航空航天大学计算机学院 北京 100083)[2]

摘 要 两阶段提交协议是最简单且最常用的原子提交协议,该协议使分布式事务的提交具有原子性和持久性。在本文中,我们使用 π-演算对两阶段提交协议进行描述,并对其正确性进行了证明,进一步体现了 π-演算对于描述进程通信及并行性的独特优势。

关键词 两阶段提交协议,形式化,异步 π-演算,互模拟

The Two Phase Commit Protocol in π-Calculus

ZHANG Fan[1] LI Zhou-Jun[2] SUN Yun[1]

(School of Computer, National University of Defence Technology, Changsha 410073)[1]

(School of Computer Science & Engineering, Beihang University, Beijing 100083)[2]

Abstract The two phase commit protocol is the simplest and most popular atomic commitment protocol (ACP). It makes the transaction commitment in the distributed systems have atomicity and durability. In this paper, we use the π-calculus to describe the two phase commit protocol, and prove the correctness, and also incarnate the unique predominance of π-calculus in description of process communication and parallelism.

Keywords 2PCP, Formalization, Asynchronous π-calculus, Bisimulation

1 引言

形式化方法[1]被公认为是一种行之有效的减少设计错误、提高系统可信度的重要途径。以通信系统演算(CCS)[2,3]为代表的进程代数方法,在并发系统的规约、分析、设计和验证等方面获得了广泛应用。π-演算[4]是在 CCS 基础上发展起来的,其将通道、常元、变元统一看作名字作为消息值域,从而能对通信拓扑结构可变的并发系统做出自然的描述。异步 π-演算[5~7]是 π-演算的一个子集,它没有选择和输出前缀,但是其进程间的交互模型与 π-演算是一致的。正是由于没有输出前缀,异步 π-演算的输出动作与其他动作只能并发执行,因此不能控制其具体执行时间。

原子提交协议(ACP)[8]是分布式事务处理中一个用于确保一致性的算法,即协调者和所有的参与者要么都提交事务,要么都撤销。两阶段提交协议(2PCP)是最简单且最常用的原子提交协议,该协议使分布式事务的提交具有原子性和持久性,并且允许每个事务管理器有权单方面决定中止还没有准备提交的事务。两阶段提交主要是用于协调在一个事物内各个独立的资源管理器的工作,这对处理被延迟到事务完成时(阶段 1)的完整性检查,以及延迟到事务最终被提交时(阶段 2)的有关工作特别有用。

在本文中,我们使用 π-演算对基本的两阶段提交协议进行描述,并证明了其实现与规约之间是观察同余的。在描述过程中,我们主要参考了 Berger[9]的相关工作。

本文内容安排如下:在第 2 节,对异步 π-演算进行了简单介绍;在第 3 节,对两阶段提交协议进行简单介绍;在第 4 节,我们用异步 π-演算对两阶段提交协议进行描述;在第 5 节,对其进行互模拟证明;最后对全文进行总结。

2 异步 π-演算

设 N 是名字的可数无穷集,其元素用 $a,b,\cdots,x,y,\cdots$ 表示。异步 π-演算的语法可用如下的 BNF 语法给出:

$$P::=0\mid\bar{x}\langle\vec{y}\rangle\mid x(\vec{y})\cdot P\mid !\ x(\vec{y})\cdot P\mid[x=y]P\mid P\mid P\mid(\upsilon x)P$$

0 代表一个没有任何动作的进程;$\bar{x}\langle\vec{y}\rangle$ 表示一个输出动作,在通道 x 输出 n 元向量 $\vec{y}$($\vec{y}$ 是 $y_1y_2\cdots y_n$ 的简写),异步 π-演算中输出动作是没有后继的;$x(\vec{y}).P$ 表示进程有一个输入前缀,在通道 x 等待一个 n 元向量 $\vec{y}$ 的输入;$P\mid P$ 表示两个进程的并发合成;$!\ x(\vec{y}).P$ 表示可数无穷多个 $x(\vec{y}).P$ 的并发合成;$[x=y]P$ 表示先对 x 和 y 进行比较,若相等则演化为 P,否则为 0;$(\upsilon x)P$ 把名字 x 的作用域限制在 P 中。

该语言有两个约束构造:限名 $(\upsilon x)P$ 和输入 $x(\vec{y}).P$ 分别将 x 和 $\vec{y}$ 约束在 P 中。P 的约束名字集和自由名字集分别用 bn(P)和 fn(P)表示。仅在约束名字上不同的进程称为 α-等价的,用 $\equiv_\alpha$ 表示。我们将不区分 α-等价的进程。替换是从 N 到 N 的部分映射,用 σ 表示。替换是后缀算子,其结合力强于语言中其它的任何算子。

定义 1(结构同余) 结构同余≡是由表 1 中的规则生成的进程上的最小同余关系。

引理 1 若 $x\notin fn(P)$ 则 $(\upsilon x)P\equiv P$

证明:$(\upsilon x)P\equiv(\upsilon x)(P\mid 0)\equiv P\mid(\upsilon x)0\equiv P\mid 0\equiv P$

*)国家自然科学基金项目(90104026,60473057)。**张 帆** 硕士研究生,主要研究方向为灾难备份与恢复;**李舟军** 博士,教授,博士生导师,主要研究方向为进程代数理论、安全协议的形式化验证、灾难备份与恢复、数据仓库与数据挖掘;**孙 云** 博士研究生,主要研究方向为进程代数理论。

下面给出异步 π-演算的迁移语义：

设动作 l 形如 $\bar{x}\langle\vec{z}\rangle, x(\vec{z}), \tau$

表 1　结构同余

Alpha $P\equiv Q$ if $P\equiv_\alpha Q$	P0 $P\mid 0\equiv P$
P1 $P\mid Q\equiv Q\mid P$	P2 $P\mid(Q\mid P)\equiv(P\mid Q)\mid R$
R0 $(\upsilon x)0\equiv 0$	R1 $(\upsilon x)(P\mid Q)\equiv P\mid(\upsilon x)Q$ if $x\notin fn(P)$
R2 $(\upsilon x)(\upsilon y)P\equiv(\upsilon y)(\upsilon x)P$	R3 $(\upsilon x)(\alpha\cdot P)\equiv\alpha\cdot(\upsilon x)P$ if $x\notin n(\alpha)$
R4 $(\upsilon x)(\alpha\cdot P)\equiv 0$ if x is the subject of α	
M0 $[x=y]P\equiv 0$ if x and y are distinct	M1 $[x=y]P\equiv P$

表 2　异步 π-演算的迁移语义

$$\text{OUT}\ \frac{}{\bar{x}\langle\vec{y}\rangle\xrightarrow{\bar{x}\langle\vec{y}\rangle}0} \qquad \text{IN}\ \frac{}{x(\vec{v})\cdot P\xrightarrow{x(\vec{z})}P\{\vec{z}/\vec{v}\}}$$

$$\text{MATCH}\ \frac{P\xrightarrow{l}P'}{[x=x]P\xrightarrow{l}P'} \qquad \text{PAR}\ \frac{P\xrightarrow{l}P'\ bn(l)\cap fn(Q)=\Phi}{P\mid Q\xrightarrow{l}P'\mid Q}$$

$$\text{COM}\ \frac{P\xrightarrow{\bar{x}\langle\vec{z}\rangle}P'\quad Q\xrightarrow{x(\vec{z})}Q'}{P\mid Q\xrightarrow{\tau}P'\mid Q'} \qquad \text{REP}\ \frac{}{\bar{x}\langle\vec{y}\rangle\mid!\,x(\vec{v})\cdot Q\xrightarrow{\tau}Q\{\vec{y}/\vec{v}\}\mid!\,x(\vec{v})\cdot Q}$$

$$\text{RES}\ \frac{P\xrightarrow{l}Q\quad x\notin fn(l)\cup bn(l)}{(\upsilon x)P\xrightarrow{l}(\upsilon x)Q} \qquad \text{OPEN}\ \frac{P\xrightarrow{\vec{x}\langle\vec{z}\rangle}Q\quad v\neq x, v\in\{\vec{z}\}}{(\upsilon x)P\xrightarrow{\vec{x}\langle(\upsilon x)\vec{z}\rangle}Q}$$

$$\text{ALPHA}\ \frac{P'\xrightarrow{l}Q\quad P\equiv_\alpha P'}{P\xrightarrow{l}Q} \qquad \text{CLOSE}\ \frac{P\xrightarrow{\bar{x}\langle(\upsilon x)\vec{z}\rangle}P'\quad Q\xrightarrow{x(\vec{z})}Q'}{P\mid Q\xrightarrow{\tau}(\upsilon v)(P'\mid Q')}$$

在 π-演算中，用基于互模拟的等价关系来刻画进程之间的行为等价关系。最重要的互模拟等价关系有强互模拟等价、弱互模拟等价和观察同余。关于 π-演算的强、弱互模拟等价和观察同余的定义请参考文[3,7]。

3　两阶段提交协议

对于每一个分布式事务 T，在执行 T 的每一个站点都有一个进程。这些进程关于 T 执行原子提交协议。在 T 的主站点(即 T 的发起站点)的进程称为 T 的协调者，其余进程则称为 T 的参与者。协调者拥有所有参与者的标识，因此它可以给所有的参与者发送消息。而每个参与者也有协调者的标识，但不一定有其他参与者的标识。

原子提交协议(ACP)是一个确保一致性的算法，即协调者和所有的参与者要么都提交事务，要么都撤销。每个进程只有一张投票：Yes 或者 No，并且最后的决定也只有一个：Commit 或者 Abort。

一个 ACP 算法必须满足：

AC1：收到决定的所有进程，其决定必须是一致的。

AC2：进程收到决定之后是不能对其进行更改的。

AC3：若决定是提交，则所有进程的投票一定是 Yes。

AC4：若没有任何失效发生，且所有的进程都投 Yes 票，则决定将会是提交。

AC5：考虑执行中发生的失效仅是算法可以允许的失效，则在执行过程中，当所有的失效都修复之后，且在一定时间内没有新的失效发生时，所有的进程最终一定会有一个决定。

AC1 是对事务终止时一致性的要求。但需要注意的是它并不要求所有的进程都收到决定，因为有的进程可能会发生失效并且永远不能恢复。而且它也不要求所有运行的进程都收到决定，因为当有通讯失效和全局失效发生时，进程可能会阻塞。然而，AC5 要求一旦恢复完成，所有的进程都能够收到一个决定。

AC2 要求事务在某个站点的结束是一个不可取消的操作。即，事务不能在提交之后又撤销，也不能在撤销之后又提交。

AC3 要求事务只有在执行其的所有站点都同意提交之后才能提交。AC4 比 AC3 的逆命题较弱一点。因为任何 ACP 都不能保证发生失效的进程能够独立恢复，所以有可能所有的进程都投了 Yes 票，而最后的决定是撤销。从 AC3 可以看到，只要进程没有投 Yes 票，就可以随时单方面地决定撤销。但一旦进程投了 Yes 票，就不能有单方面的动作了。在进程投 Yes 票至收到决定期间，称为进程的非确定阶段。处在非确定阶段的进程称为不确定的。不确定的进程是不知道其最后是要撤销还是要提交，因此是不可以单方面决定撤销的。

两阶段提交协议是最简单且最常用的原子提交协议(ACP)。

在不考虑失效发生的情况下，2PCP 的描述如下：

(1)协调者向每个参与者发送 VOTE-REQ 消息。

(2)当参与者收到 VOTE-REQ，就给协调者发送其投票：YES 或者 NO。若参与者的投票是 No，则其立即决定撤销并终止。若参与者的投票是 Yes，则等待协调者的消息。

(3)协调者收集所有参与者的投票。若都是 Yes 且协调者自己的投票也是 Yes，则协调者决定提交并向每个参与者发送 COMMIT 消息。只要有一张投票是 No，则协调者决定撤销并向每个投 Yes 票的参与者发送 ABORT 消息。两种情况下，协调者在发送消息后都将终止。

(4)投 Yes 票的参与者收到协调者的 COMMIT 或 ABORT 消息后，则做出相应决定并终止。

前两步是 2PCP 的投票阶段，后两步则是 2PCP 的决定阶段。一个参与者的非确定阶段是从给协调者发送 Yes 票直至收到协调者的 COMMIT 或 ABORT 消息。协调者是没有不确定阶段的，因为协调者在其投票之后立即有了最终的决定。

4　两阶段提交协议的形式化描述

在我们的描述中，2PCP 由 $n+1$ 个进程组成：一个协调者 C 和 n 个参与者 $P_1,\cdots,P_n$。协调者和各个参与者之间通过单独的通道 c_i 和 d_i 进行通讯。

$2PCP\equiv(\upsilon\vec{cd})(C\mid P_1\mid\cdots\mid P_n)$

协调者本身又由多个子进程组成：

$C\equiv(\upsilon\vec{ab})(C^{req}\mid C^{wait}\mid C^{and}\mid C^{or})$

$C^{req}\equiv\prod_{i=1}^{n}fC_i^{req}$　　$C_i^{req}\equiv\vec{c_i}\langle\text{REQ}\rangle$

$C^{wait}\equiv\prod_{i=1}^{n}C_i^{wait}$　　$C_i^{wait}\equiv d_i(x)\cdot([x=\text{NO}]\bar{a}\mid[x=\text{YES}]\bar{b_i})\ x\in\{\text{YES},\text{NO}\}$

$C^{and}\equiv b_1\cdot b_2\cdots b_n\cdot C^{vote}$　　$C^{vote}\equiv C^{abort}\oplus C^{commit}$

$C^{or}\equiv a\cdot C^{abort}$

$C^{abort}\equiv\prod_{i=1}^{n}C_i^{abort}$　　$C_i^{abort}\equiv\vec{c_i}\langle\text{ABO}\rangle$

$C^{commit}\equiv\prod_{i=1}^{n}C_i^{commit}$　　$C_i^{commit}\equiv\vec{c_i}\langle\text{COM}\rangle$

C^{req} 进程表示协议初始时协调者向每个参与者发送 VOTE-REQ 消息；C^{wait} 进程表示协调者等待参与者的投票；

C^{cnd} 进程收齐所有参与者的 YES 投票之后，协调者最终做一个决定是提交还是撤销，若提交则向每个参与者发送 COMMIT 消息，若撤销则发送 ABORT 消息；如果有一个参与者投了 NO 票，则 C^{cr} 进程将会告知所有的参与者撤销事务。为了模拟参与者和协调者的投票，用 $P \oplus Q$ 表示内部选择，它是 $(\upsilon x)(\bar{x}|x \cdot P|x \cdot Q)(x \notin n(P,Q))$ 的简写形式。

$P_i \equiv c_i(x) \cdot [x=\mathrm{REQ}]P_i^{vote}$

$P_i^{vote} \equiv P_i^{abort} \oplus P_i^{commit}$

$P_i^{abort} \equiv \bar{d}_i\langle \mathrm{NO}\rangle|\overline{abort_i}$　　$P_i^{commit} \equiv \bar{d}_i\langle \mathrm{YES}\rangle|P_i^{wait}$

$P_i^{wait} \equiv c_i(x) \cdot ([x=\mathrm{ABO}]\overline{abort_i}|[x=\mathrm{COM}]\overline{commit_i})$

$x \in \{\mathrm{COM},\mathrm{ABO}\}$

P_i 表示第 i 个参与者。初始时，参与者等待从协调者发来的消息，若该消息是 VOTE-REQ，则 P_i^{vote} 进程准备向协调者发送投票，若投票是 NO，则立即决定撤销；若投票是 YES，则 P_i^{wait} 进程等待协调者的消息。

5 两阶段提交协议的证明

首先，要确保一致性，即参与事务的协调者和所有的参与者要么都提交事务，要么都撤销。即：2PCP≈Abort⊕Commit(Abort≡$\prod_{i=1}^{n}\overline{abort_i}$，Comit≡$\prod_{i=1}^{n}\overline{commit_i}$)。

再者，需要证明各个投票与结果之间的关系。当所有进程的投票都是 Yes 时，决定提交，即：若 $\forall i \in n$ 都满足 $P_i^{vote}=P_i^{commit}$ 且 $C^{vote}=C^{commit}$，则 2PCP≈Commit。当有一个进程投否定票时，决定撤销，即：若 $\exists i \in n$ 满足 $P_i^{vote}=P_i^{abort}$ 或 $C^{vote}=C^{abort}$，则 2PCP≈Abort。

在无失效的情况下，$n+1$ 个内部选择是 2PCP 的唯一的不确定因素，也可称之为信息源。每个信息源都有各自的通道与协调者联系，互不干扰。而协调者综合各个参与者的信息，最后作一个决定(提交或者撤销)。然后这个决定又通过各自的通道传回参与者。

初始时，协调者向 n 个参与者发送 REQ(请求投票)消息，参与者接收这些消息：

$C \xrightarrow{\bar{c}_1\langle REQ\rangle} \cdots \xrightarrow{\bar{c}_n\langle REQ\rangle} (\upsilon \vec{ba})(\prod_{i=1}^{n}C_i^{wait}|b_1 \cdot b_2 \cdots b_n \cdot (C^{abort} \oplus C^{commit})|a \cdot C^{abort})$

$C_i^{wait} \equiv d_i(x) \cdot ([x=\mathrm{NO}]\bar{a}|[x=\mathrm{YES}]\bar{b_i})x \in \{\mathrm{YES},\mathrm{NO}\}$

$C^{abort} \equiv \prod_{i=1}^{n}\bar{c}_i\langle \mathrm{ABO}\rangle$　　$C^{commit} \equiv \prod_{i=1}^{n}\bar{c}_i\langle \mathrm{COM}\rangle$

$P_i \xrightarrow{c_i(REQ)} (\bar{d}_i\langle \mathrm{NO}\rangle|\overline{abort_i}) \oplus (\bar{d}_i\langle \mathrm{YES}\rangle|P_i^{wait})$

$P_i^{wait} \equiv c_i(x) \cdot ([x=\mathrm{ABO}]\overline{abort_i}|[x=\mathrm{COM}]\overline{commit_i})$

$x \in \{\mathrm{COM},\mathrm{ABO}\}$

为描述方便，令 $S,T \subseteq \{1,\cdots,n\}$，$S \cap T=\Phi$，$i,k \in S$；$i \geqslant k$；$\forall j \in S, k \leqslant j$；$\forall j \in T, k<j$

$D_{S,T,k}^{s} \equiv (\upsilon \vec{ba})(\prod_{j\in S}C_j^{wait}|\prod_{j\in T}\bar{b_j}|b_k \cdots b_n \cdot (C^{abort} \oplus C^{commit})|a \cdot C^{abort})|\prod_{j\in T\cup\{1,\cdots,k-1\}}P_j^{wait}$

在这里，S 表示还没有投票的参与者的下标集合，T 表示已经投了 yes 票，但是在其下标序号之前还有没投票的参与者，k 表示下标为 1 到 k−1 的参与者已经投了 yes 票。

$D_{S,T}^{a} \equiv \prod_{j\in S}d_j(x)|\prod_{j\in S}C_j^{abort}|\prod_{j\in T}\overline{abort_j}$

在这里，S 表示还没有投票的参与者的下标集合，T 表示已经投票的参与者的下标集合，并且其已经收到了协调者的撤销消息。

命题 1　$D_{S,T,k}^{s}|P_i^{abort} \approx D_{S\setminus\{i\},T\cup\{1\cdots k-1,i\}}^{a}|C_i^{abort}$

证明：

$D_{S,T,k}^{s}|P_i^{abort}$

$\equiv (\upsilon \vec{ba})(\prod_{j\in S}C_j^{wait}|\prod_{j\in T}\bar{b_j}|b_k \cdots b_n \cdot (C^{abort} \oplus C^{commit}))|a \cdot C^{abort})|\prod_{j\in T\cup\{1,\cdots,k-1\}}P_j^{wait}|\bar{d}_i\langle \mathrm{NO}\rangle|\overline{abort_i}$

$\xrightarrow{\tau} (\upsilon \vec{ba})(\bar{a}|[\mathrm{NO}=\mathrm{YES}]\bar{b_i}|\prod_{j\in S\setminus\{i\}}C_j^{wait}|\prod_{j\in T}\bar{b_j}|b_k \cdots b_n \cdot (C^{abort} \oplus C^{commit})|a \cdot C^{abort})|\prod_{j\in T\cup\{1,\cdots k-1\}}P_j^{wait}|\overline{abort_i}$

根据规则 M0，$[\mathrm{NO}=\mathrm{YES}]\bar{b_i} \equiv 0$，即通道 b_i 不会有输出信息。

$(\upsilon \vec{ba})(\bar{a}|\prod_{j\in S\setminus\{i\}}C_j^{wait}|\prod_{j\in T}\bar{b_j}|b_k \cdots b_n \cdot (C^{abort} \oplus C^{commit})|a \cdot C^{abort})\prod_{j\in T\cup\{1,\cdots k-1\}}P_j^{wait}|\overline{abort_i} \xrightarrow{\tau} (\upsilon \vec{ba})(\prod_{j\in S\setminus\{i\}}C_j^{wait}|\prod_{j\in T}\bar{b_j}|b_k \cdots b_n \cdot (C^{abort} \oplus C^{commit})|C^{abort})|\prod_{j\in T\cup\{1,\cdots k-1\}}P_j^{wait}|\overline{abort_i} \xrightarrow{\tau} \cdots \xrightarrow[|T|+k-1]{\tau} (\upsilon \vec{ba})(\prod_{j\in S\setminus\{i\}}C_j^{wait}|\prod_{j\in T}\bar{b_j}|b_k \cdots b_n \cdot (C^{abort} \oplus C^{commit})|\prod_{j\in S}C_j^{abort})|\prod_{j\in T\cup\{1,\cdots k-1,i\}}\overline{abort_i}$

$\equiv (\upsilon \vec{ba})(\prod_{j\in S\setminus\{i\}}C_j^{wait}|\prod_{j\in T}\bar{b_j}|b_k \cdots b_n \cdot (C^{abort} \oplus C^{commit}))|\prod_{j\in S}C_j^{abort}|\prod_{j\in T\cup\{1,\cdots k-1,i\}}\overline{abort_j}$

$\equiv (\upsilon b_1 \cdots b_{i-1}b_{i+1} \cdots b_n a)(\prod_{j\in S\setminus\{i\}}C_j^{wait}|\prod_{j\in T}\bar{b_j}|b_k \cdots b_{i-1} \cdot (\upsilon b_i)(b_i \cdots b_n \cdot (C^{abort} \oplus C^{commit})))|\prod_{j\in S}C_j^{abort}|\prod_{j\in T\cup\{1,\cdots k-1,i\}}\overline{abort_j}$

$\equiv (\upsilon b_1 \cdots b_{i-1}b_{i+1} \cdots b_n a)(\prod_{j\in S\setminus\{i\}}C_j^{wait}|\prod_{j\in T}\bar{b_j}|b_k \cdots b_{i-1})|\prod_{j\in S}C_j^{abort}|\prod_{j\in T\cup\{1,\cdots,k-1,i\}}\overline{abort_j}$

又 $(\upsilon b_1 \cdots b_{i-1}b_{i+1} \cdots b_n a)(\prod_{j\in S\setminus\{i\}}C_j^{wait}|\prod_{j\in T}\bar{b_j}|b_k \cdots b_{i-1}) \approx \prod_{j\in S\setminus\{i\}}c_j(x)$

因为通道 $a,b_j(j \in T\cup\{k,\cdots,i-1)$ 都被限制了，且所有输出都是自由输出，即通道 $a,b_j(j \in T\cup\{k,\cdots,i-1\})$ 的作用域不会扩大，所以等式左边除了 τ 动作就只有 $\prod_{j\in S\setminus\{i\}}d_j(x)$ 了。

$(\upsilon b_1 \cdots b_{i-1}b_{i+1} \cdots b_n a)(\prod_{j\in S\setminus\{i\}}C_j^{wait}|\prod_{j\in T}\bar{b_j}|b_k \cdots b_{i-1})|\prod_{j\in S}C_j^{abort}|\prod_{j\in T\cup\{1,\cdots k-1,i\}}\overline{abort_j}$

$\approx \prod_{j\in S\setminus\{i\}}d_j(x)|\prod_{j\in S}C_j^{abort}|\prod_{j\in T\cup\{1,\cdots,k-1,i\}}\overline{abort_j}$

$\equiv \prod_{j\in S\setminus\{i\}}d_j(x)|\prod_{j\in S\setminus\{i\}}C_j^{abort}|\prod_{j\in T\cup\{1,\cdots k-1,i\}}\overline{abort_j}|C_i^{abort}$

$\equiv D_{S\setminus\{i\},T\cup\{1,\cdots k-1,i\}}^{a}|C_i^{abort}$

综上所述，$D_{S,T,k}^{s}|P_i^{abort} \approx D_{S\setminus\{i\},T\cup\{1,\cdots k-1,i\}}^{a}|C_i^{abort}$

命题 2　$D_{S,T,k}^{s}|P_i^{abort} \approx D_{S\setminus\{i\},wait_k(T\cup\{i\}),boundary_k(T\cup\{i\})}^{a}$

在此

$$wait_k(S) \equiv \begin{cases} \Phi & S=\Phi \\ wait_{k+1}(S\setminus\min(S)) & S\neq\Phi,\min(S)=k \\ \{\min(S)\}\cup wait_k(S\setminus\min(S)) & S\neq\Phi,\min(S)>k \end{cases}$$

$$boundary_k(S) \equiv \begin{cases} k & S=\Phi \\ boundary_{k+1}(S\setminus\min(S)) & S\neq\Phi,\min(S)=k \\ k & S\neq\Phi,\min(S)>k \end{cases}$$

证明：

$D_{S,T,k}^{s}|P_i^{commit}$

$\equiv (\upsilon \vec{ba})(\prod_{j\in S}C_j^{wait}|\prod_{j\in T}\bar{b_j}|b_k \cdots b_n \cdot (C^{abort} \oplus C^{commit})|a \cdot C^{abort})|\prod_{j\in T\cup\{1,\cdots k-1\}}P_j^{wait}|\bar{d}_i\langle \mathrm{YES}\rangle|P_i^{wait}$

$\xrightarrow{\tau} (\upsilon \vec{ba})(\bar{b_i}|\prod_{j\in S\setminus\{i\}}C_j^{wait}|\prod_{j\in T}\bar{b_j}|b_k \cdots b_n \cdot (C^{abort} \oplus C^{commit})|a \cdot C^{abort})|\prod_{j\in T\cup\{1,\cdots k-1,i\}}P_j^{wait}$

若 $i>k$，则

$(\upsilon \vec{ba})(\bar{b_i}|\prod_{j\in S\setminus\{i\}}C_j^{wait}|\prod_{j\in T}\bar{b_j}|b_k \cdots b_n \cdot (C^{abort} \oplus C^{commit})|a \cdot C^{abort})|\prod_{j\in T\cup\{1,\cdots k-1,i\}}P_j^{wait} \equiv (\upsilon \vec{ba})(\bar{b_i}|\prod_{j\in S\setminus\{i\}}C_j^{wait}|\prod_{j\in T\cup\{i\}}\bar{b_j}|b_k \cdots b_n \cdot (C^{abort} \oplus C^{commit})|a \cdot C^{abort})|\prod_{j\in(T\cup\{i\})\cup\{1,\cdots k-1\}}P_j^{wait}$

$\equiv D_{S\setminus\{i\},T\cup\{i\},k}^{s}$

$\equiv D^{c}_{S\setminus\{i\},wait_k(TU\{i\}),boundary_k(TU\{i\})}$

若 $i=k$，则

$(\upsilon\vec{ba})(\bar{b_i}\mid\prod_{j\in S\setminus\{i\}}C_j^{wait}\mid\prod_{j\in T}\bar{b_j}\mid b_k\cdots b_n\cdot(C^{abort}\oplus C^{commit})\mid a\cdot C^{abort})\mid\prod_{j\in TU\{1,\cdots k-1,i\}}P_j^{wait}$

$\xrightarrow{\tau}(\upsilon\vec{ba})(\prod_{j\in S\setminus\{i\}}C_j^{wait}\mid\prod_{j\in T}\bar{b_j}\mid b_{k+1}\cdots b_n\cdot(C^{abort}\oplus C^{commit})\mid a\cdot C^{abort})\mid\prod_{j\in TU\{1,\cdots,k\}}P_j^{wait}$

$\xrightarrow{\hat{\tau}}D^{c}_{S\setminus\{i\},wait_k(TU\{i\}),boundary_k(TU\{i\})}$

综上所述，$D^{c}_{S,T,k}\mid P_i^{commit}\approx D^{a}_{S\setminus\{i\},wait_k(TU\{i\}),boundary_k(TU\{i\})}$

命题 3 $D^{c}_{S,T}\mid P_i^{abort}\approx D^{a}_{S\setminus\{i\},TU\{i\}}\mid C_i^{abort}$

证明：

$D^{c}_{S,T}\mid P_i^{abort}$

$\equiv\prod_{j\in S}d_j(x)\mid\prod_{j\in S}C_j^{abort}\mid\prod_{j\in T}\overline{abort_j}\mid\bar{d_i}\langle NO\rangle\mid\overline{abort_i}$

$\xrightarrow{\tau}\prod_{j\in S\setminus\{i\}}d_j(x)\mid\prod_{j\in S}C_j^{abort}\mid\prod_{j\in TU\{i\}}\overline{abort_j}$

$\xrightarrow{\tau}\prod_{j\in S\setminus\{i\}}d_j(x)\mid\prod_{j\in S\setminus\{i\}}C_j^{abort}\mid\prod_{j\in TU\{i\}}\overline{abort_j}\mid C_i^{abort}$

$\equiv D^{a}_{S\setminus\{i\},TU\{i\}}\mid C_i^{abort}$

$\therefore D^{c}_{S,T}\mid P_i^{abort}\approx D^{a}_{S\setminus\{i\},TU\{i\}}\mid C_i^{abort}$

命题 4 $D^{c}_{S,T}\mid P_i^{commit}\approx D^{a}_{S\setminus\{i\},TU\{i\}}$

证明：

$D^{c}_{S,T}\mid P_i^{commit}$

$\equiv\prod_{j\in S}d_j(x)\mid\prod_{j\in S}C_j^{abort}\mid\prod_{j\in T}\overline{abort_j}\mid\bar{d_i}\langle YES\rangle\mid P_i^{wait}$

$\xrightarrow{\tau}\prod_{j\in S\setminus\{i\}}d_j(x)\mid\prod_{j\in S\setminus\{i\}}C_j^{abort}\mid\prod_{j\in TU\{i\}}\overline{abort_j}$

$\equiv D^{a}_{S\setminus\{i\},TU\{i\}}$

$\therefore D^{c}_{S,T}\mid P_i^{commit}\approx D^{a}_{S\setminus\{i\},TU\{i\}}$

当所有参与者都投票之后，即 $S=\Phi$，则

$D^{c}_{\Phi,\Phi,n+1}\equiv(\upsilon\vec{ba})((C^{abort}\oplus C^{commit})\mid a\cdot C^{abort})\mid\prod_{j\in\{1,\cdots,n\}}P_j^{wait}$

$D^{a}_{\Phi,\{1,\cdots,n\}}\equiv\prod_{j\in\{1,\cdots,n\}}\overline{abort_j}$

若协调者在 n 个参与者投票提交之后也决定提交，则

$D^{c}_{\Phi,\Phi,n+1}\xrightarrow{\tau}(\upsilon\vec{b})(C^{commit}\mid a\cdot C^{abort})\mid\prod_{j\in\{1,\cdots,n\}}P_j^{wait}$

$\equiv(\upsilon\vec{ba})(a\cdot C^{abort})\mid C^{commit}\mid\prod_{j\in\{1,\cdots,n\}}P_j^{wait}\equiv C^{commit}\mid\prod_{j\in\{1,\cdots,n\}}P_j^{wait}$

$\underbrace{\xrightarrow{\tau}\cdots\xrightarrow{\tau}}_{n}\prod_{j\in\{1,\cdots,n\}}\overline{commit_j}$

若协调者在 n 个参与者投票提交之后决定撤销，则

$D^{c}_{\Phi,\Phi,n+1}\xrightarrow{\tau}(\upsilon\vec{ba})(C^{abort}\mid a\cdot C^{abort})\mid\prod_{j\in\{1,\cdots,n\}}P_j^{wait}$

$\equiv(\upsilon\vec{ba})(a\cdot C^{abort})\mid C^{abort}\mid\prod_{j\in\{1,\cdots,n\}}P_j^{wait}$

$\equiv C^{abort}\mid\prod_{j\in\{1,\cdots,n\}}P_j^{wait}$

$\underbrace{\xrightarrow{\tau}\cdots\xrightarrow{\tau}}_{n}\prod_{j\in\{1,\cdots,n\}}\overline{abort_j}$

当所有进程的投票都是 Yes 时，$2PCP\approx(\upsilon\vec{c})(\prod_{j\in\{1,\cdots,n\}}\overline{commit_j})\equiv Commit$；当有一个进程投 No 票时，$2PCP\approx(\upsilon\vec{c})(\prod_{j\in\{1,\cdots,n\}}\overline{abort_j}\mid\prod_{i\in A}C_i^{abort})\equiv$ Abort，A 表示投 No 票的参与者的下标集合。因此，$2PCP\simeq$ Abort$\oplus$Commit。

结束语 本文的主要工作是使用 π-演算对两阶段提交协议进行了形式化的描述，并进行了验证。两阶段提交协议是一个被广泛使用的例子，但对它的正确性证明并不多，使用进程代数方法的证明尤为少见。Bernstain 在文[8]中对两阶段提交协议进行了详细的文字描述和讨论，但并没有对其正确性进行证明。Berger 在文[9]中使用了进程代数方法对两阶段提交协议进行了形式化的描述，但其使用的语言并不是标准的 π-演算语言，并且在其描述过程中对两阶段的理解有误，如事务的协调者应该是在收到所有参与者的投票之后自己才会投票，即协调者是没有不确定阶段的。Berger 在文中对两阶段提交协议的正确性也进行了证明，但是他只证明了两阶段提交协议的实现与规约是弱互模拟等价的，并且其证明过程并不严格。

我们对两阶段提交协议的描述参考了 Bernstain 和 Berger 的工作。与他们不同的是：1）我们使用了标准的异步 π-演算进行描述。2）我们给出了详细而准确的描述与证明过程。3）证明了两阶段提交协议的实现和规约是观察同余的。

本文只是无失效情况下对 2PCP 的描述，还没有考虑超时动作和失效恢复情况的描述。下一步的工作是能给出一个包括失效和恢复等情况在内的 2PCP 的描述和验证。

参 考 文 献

1 Clarke E M, Wing J M. Formal Methods: State of the Art and Future Directions. ACM Computing Surveys (CSUR), 1996, 28(4): 626～643

2 Milner R. Communication and Concurrency. Prentice Hall, 1989

3 李舟军. 传值 CCS 和 π-演算的验证理论和算法:[博士论文]. 国防科技大学, 1999

4 Milner R, Parrow J, Walker D. A Calculus of Mobile Processes, Part I, II. Information and Computation, 1992, 100(1): 1～77

5 Boudol G. Asynchrony and the π-Calculus: [Research Report]. 1702. INRIA, Sphia-Antipolis, 1992

6 Honda K, Tokoro M. On Asynchronous Communication Semantics. Object-based concurrent computing, Springer Lect Notes in Comp Sci 612, 1992

7 Amadio R M, Castellani I, Sangiorgi D. On bisimulations for the asynchronous π-calculus. Theoretical Computer Science, 1998

8 Bernstain P A. Concurrency Control and Recovery in Database Systems. Addison-Wesley Longman Publishing Co Inc Boston, MA. 1987

9 Berger M. Towards Abstractions for Distributed Systems: [PhD thesis]. the University of London, Revised Version, 2004

（上接第 262 页）

8 van Ommering R, van der Linden F, Kramer J, et al. The koala component model for consumer electronics software. IEEE Computer, 2000

9 Stewart D B, Volpe R A, Khosla P K. Design of dynamically reconfigurable real-time software using port-based objects. IEEE Trans Software Eng, 1997, 23(12): 759～776

10 Winter M, Genßer T, Christoph A, et al. Components for embedded software-The PECOS approach. In: Proc. Second International Workshop on Composition Languages, 2002

11 Nierstrasz O, Arvalo G, Ducasse S, et al. A component model for field devices. IFIP/ACM Conference on Component Deployment, Berlin, Germany, June 2002

12 Urting D, Van Baelen S, Holvoet T, et al. Embedded software development: components and contracts. In: Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems. ACTA Press, 2001. 685～690

13 Urting D, Berbers Y, Van Baelen S, et al. A tool for component based design of embedded software. In: Proceedings of the Fortieth International Confernece on Tools Pacific: Objects for internet, mobile and embedded applications. February 2002, 10

14 Yen I-Ling, Goluguri J, Bastani F, et al. A Component-based Approach for Embedded Software Development. In: Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC. 02)

15 de Lucena V F Jr. Facet-Based Classification Scheme for Industrial Automation Software Components

16 NATO Communications and Information Systems Agency. NATO Standard for Management of A Reusable Software Component Library. 1991, 2: 50～54

17 Yu Zhi-wen, Zhou Xing-she. Study of Development Techniques Based on Embedded Softwae Components. Application research of computers, 2003(4): 12～14