计算机科学 1996 Vol.23 No.5

# 分布式计算中的因果性研究:回顾与展望

贺乐天 孙永强
(上海交通大学计算机科学与工程系 上海 200030)

**摘 要** In the execution of a distributed program, causal relationships between events are fatal for us to understand and analysis the behavior of the program. This paper systematically reviews the development of causality in the last ten years. We discuss the important role of causality in many distributed applications. We also present our understanding of current research and point out some future directions.

**关键词** Distributed computation, Causal relationship, Logical time, Partial order relationship, Predicate detection.

## 1 引 言

分布式系统是一组在空间上分隔的顺序进程,进程间仅仅通过消息传递来交换信息,由于缺乏全局时钟,基于事件观察的事件交叉线性序列往往不适合于我们对分布式程序行为的理解,在实际应用中带来很多问题。现在已经认识到,将事件之间的关系理解成基于事件相互影响的偏序关系是比较合适的[13,18],这种相互影响的关系也被称为因果关系。

本文较系统地综述了十多年来因果性问题研究的发展情况,并讨论了因果性在分布式计算中很多重要问题中的应用,阐述了对研究现状的认识,并指出了若干研究方向。

## 2 系统模型和因果性问题

一个分布式系统包含 $N$ 个顺序进程 $P_1,\cdots,P_N$,进程间以消息传递的方式通讯,系统中没有全局时钟或精确同步的局部时钟,没有共享存储器。我们假定消息传递的延迟是任意的;并假定进程间的通讯是点对点的,通信信道是可靠的。这一模型也常被称为基于消息传递的分布计算模型。

我们首先定义分布式计算中的事件集合。

**定义 2.1** $N$ 个顺序进程 $P_1,\cdots,P_N$ 构成的分布计算中,用 $E_i$ 表示进程 $P_i$ 中发生的所有局部事件的集合,$E=E_1\cup\cdots\cup E_N$ 表示分布计算中发生的所有事件的集合。对 $E_i$ 中发生的局部事件按其发生的局部时间先后次序排序,即 $E_i=\{e_{i,1},e_{i,2},e_{i,3},\cdots\}$。

我们假设事件都是原子事件,并将所有事件区分为三类:发送消息的事件,接收消息的事件和内部事件。

下面我们定义事件间的因果关系。

**定义 2.2** 事件集合 $E$ 上的因果关系:$\rightarrow\subseteq E\times E$ 是满足如下条件的最小传递关系:①若 $e_{i,j},e_{i,k}\in E_i$ 是进程 $P_i$ 的局部事件,且 $j<k$,则 $e_{i,j}\rightarrow e_{i,k}$;②若 $s\in E_i$ 是消息发送事件,其相应的消息接收事件为 $r\in E_j(i\neq j)$,则 $s\rightarrow r$。

这一关系与 Lamport 的"happened before"关系[13]是相同的。显然,因果关系$\rightarrow$是反自反、反对称、传递的,因而是一严格的偏序关系。

用因果关系的概念可以定义事件间的并发关系。

**定义 2.3** 事件集合 $E$ 上的并发关系:$\|\subseteq E\times E$ 被定义为:$\forall e,e'\in E, e\|e'$ $iff$ $(e\not\rightarrow e')\wedge(e'\not\rightarrow e)$。容易看到,并发关系是非传递的,即由 $e\|e',e'\|e''$,并不能得到 $e\|e''$。

分布式计算中事件间的关系是一个偏序关系,这个偏序关系就是因果关系。因果关系或因果性是分布式计算中很多问题的基础,我们通过列举因果性的应用来说明其重要性。

由于没有全局时钟,不能得到全局状态,因此全

贺乐天 博士生,主要研究兴趣为分布式计算系统; 孙永强 教授,博士生导师,主要研究兴趣为计算理论、新型语言。

局快照[4]的概念被提出来作为分布式计算的"全局状态"。全局快照是各个进程上局部快照的集合，全局快照并不总是有意义的。一致全局快照指的是那些正确地反映了因果关系的全局快照：若 $e$ 在全局快照中，且 $e' \rightarrow e$，那么 $e'$ 也必然在该快照中。这样，我们便获得了一个一致性的概念，其本质上就是正确地反映因果性，我们称之为因果一致性。

因果一致性有很多重要的应用[7,10-13,15,17-19,21]。如在分布式数据库中用来确定一个一致的恢复点；分布式死锁和终止检测中，计算状态的因果一致视图可以用来检测伪死锁和假终止；在分布式调试中，检测全局谓词也需要一个因果一致的系统状态；在竞争条件和其他同步错误的检测中，分析事件间的因果关系能将注意力集中到那些可能的并发事件上，从而大大地减少了分析的工作量；在分布式监控和调试中，常常需要重演当前和历史活动，此时因果性决定了需处理的事件，大大地减少了需存储的信息，提高了处理效率。

在分布式算法的设计中，对计算中事件因果关系的分析可得到一个算法的抽象并发性的度量[5]，从而揭示出计算的内在并发性并进行相应的优化。

在分布式协议中，减少不必要的同步限制会得到较高的并发性，而对同步的最小需求就是保证事件间的因果关系。现在，一些系统(如 ISIS[3])中已经实现了因果次序的通信原语，使用因果序，而不是完全序或同步序，对开发异步系统的性能也有重要的作用。因果共享存储就是运用了这一思想而实现的一种弱形式的共享虚拟存储。

在证明异步系统的性质[14,18,22]中，用因果关系来取代传统使用的时态关系，现已被认为是一种正确的抽象方法。实际上这一思想早已被 Petri 网所采用。

可以看到，因果性是分布式计算中的一个基本特性。今天，分布式系统普遍地存在于我们的社会中，但是这些系统的实现手段和效率并不令人满意，我们迫切需要一些适合于分布式计算的编程工具和环境，克服由于分布而引起的问题，并开发潜在的分布并发能力，如提高速度、可用性和可靠性。十几年来，这方面所取得的研究成果不尽如人意。我们认为一个至关重要的原因是忽视了因果性这一本质问题；原因之二还在于因果结构本身的复杂性，大多数的研究者宁愿采取从传统集中式系统过渡的研究方法(即假设全局时钟、全局状态、寻找同步时钟等)。但是，分布式计算中使用传统的牛顿模型，亦即基于绝对全局时间概念的模型，是不适宜于反映异步、物理上分布和不确定的消息延迟和进程速度的这种相对性的[25]，因果结构上的偏序语义是理解分布式计算行为的正确方法。

## 3 因果性问题研究的发展情况

分布式计算中因果性的研究较明显地异军突起。下面大致按时间和问题类型的线索来综述十几年来的发展情况。

### 3.1 偏序语义

1978 年，Lamport 用"happened before"[13]关系很自然地描述了计算中事件的相互影响，揭示了分布式计算的偏序语义，它完全不同于传统上假设了一个事件全序的交叉语义，这篇论文促成了分布式计算理论的主要进步。

在理论上，计算机科学家和逻辑学家就偏序语义还是交叉语义作了大量的研究。这一问题本质上与计算所基于的时间的基本结构是线性结构还是偏序结构是同一问题。传统的线性时间方法将时间考虑成一线性序列，分支时间方法[2]采用树结构的时间，允许在某一时刻有不止一个后继，偏序时间方法[17]则采用偏序结构的时间。交叉集方法[11]结合了前三种逻辑方法的特性，其语义模型是基于一组交叉系列的集合，它从计算模型的底层语义上反映了偏序。这些逻辑方法(文[12]除外)在表达能力上是不可比较的，具体选择哪种方法取决于我们试图形式化和研究的程序类型和程序性质类型。例如，在刻画一个程序产生的所有执行系列的集合和研究在这些系列中都一致保持的性质(如完全正确性)时，基于线性时间的方法是可行的；但对于异步的分布式系统，基于偏序语义的方法才是正确的。

1986 年，Pratt[16]强烈地反对全序假设，其论证的主要依据是：在存在不可区分的事件时，偏序并不等同于其所有线性化全序的集合。1985 年，Lamport[14]在事件原子性的基础上反对全序假设，他引入了事件上的两个关系：precede 和 affect，并说明了这两个关系对于在不同事件粒度下论证一个计算是合适的，这篇文章的主要贡献在于它提出的两个关系完全独立于全局时间的概念。文[1]对文[14]的公理化理论提出了基于全局时间的模态语义，将一系

统执行(由 precede 和 affect 关系定义的)等同于一组全局时间解释,并以此作为规范和验证异步分布式系统的一个合理性基础。然而,有趣的是,文[10]中提出了一种原子寄存器的结构,并说明了当基于偏序语义来证明时,这种结构是正确的;但若证明中假设了全局时间,并且每个偏序等同于其所有的线性化全序,这种结构是错误的。这篇文章说明了基于全局时间的语义来规范和论证分布式系统是有缺点的。

### 3.2 逻辑时钟与因果性刻画

Lamport 在文[13]中同时还提出了一个反映因果关系的逻辑时钟算法。

**定义 3.1** Lamport 逻辑时钟是事件集合 $E$ 到自然数集合 $N$ 的映射 $L:E\to N$,递归定义如下:①$e$ 是内部事件或发送事件,若 $e$ 没有局部的直接前趋事件,则 $L(e)=1$;否则前趋事件为 $e'$,$L(e)=L(e')+1$。②$e$ 是一接收事件,$s$ 是其相应的发送事件,若 $e$ 没有局部的直接前趋事件,则 $L(e)=L(s)+1$;否则前趋事件为 $e'$,$L(e)=\max(L(s),L(e'))+1$。

尔后,各种各样的逻辑时钟和同步时钟方案被提出来(直到今天,这一研究仍然长盛不衰),但这些时钟大多不能够完全刻画因果性,因而不能用于算法的证明,这一困难是导致因果性未能引起足够重视的原因之一。

逻辑时钟的研究在 80 年代末期达到了高潮。Fidge[9]和 Mattern[16]先后独立地提出了向量时钟的概念,并证明了它刻画了因果性。

**定义 3.2** $n$ 个进程 $P_1,\cdots,P_n$ 构成的分布式计算,$E$ 为所有事件的集合,任一进程 $P_i$ 保持一个 $n$ 维的向量 $V_i$,各进程遵循如下的协议:

①初始时,$V_i[k]:=0,1\leqslant k\leqslant n$;

②对每一内部事件,$V_i[i]:=V_i[i]+1$;

③对每一发送事件,$V_i[i]:=V_i[i]+1$,且让发送出去的消息带上修改后的向量 $V_i$;

④对每一接收事件,$V_i[i]:=V_i[i]+1$,假设消息所携带的向量为 $V(m)$,则 $V_i[j]:=\max(V_i[j],V(m)[j]),1\leqslant j\leqslant n$。任一事件发生时所在进程上的向量即为该事件的向量时钟。

向量时钟刻画了事件间的因果关系,即:

$\forall e,e',e\to e' iff\ V(e)\geqslant V(e')$;

$\forall e,e',e\parallel e' iff\to(V(e)\geqslant V(e'))\wedge\to(V(e')\geqslant V(e))$。

向量时钟的这种能力,使得我们可以较为定量地描述因果一致性,很多分布式算法使用这一概念可以描述得更简洁。向量时钟第一次精确地揭示出了因果性的复杂内涵。但是,使用向量时钟的主要问题是,它需要 $n$ 维大小的向量,在通信中将占用很大的通信带宽,尤其是对于进程数很大的分布式计算和那些在计算中进程数动态变化的计算。

此后,降低向量时钟复杂度的方案在各种不同的应用实践中被提出来,特别是压缩向量时钟的算法和相应的重构算法。但是,这些算法不能够适应于普遍模型中的分布式计算且带来了很大的计算复杂度,这方面较有代表性的是文[23]。

1991 年,Charron-Bost[6]构造了一个分布式计算的实例,说明了为了刻画因果性,n 维大小的向量是必要的。她还从偏序集理论的角度对这一结论给出了精辟的数学解释。值得一提的,Charron-Bost 利用向量时钟,在并发进程并发性的抽象测量上也作出了开创性的工作[5]。由于因果结构本身的复杂性,在实际编程工具的设计中,完全刻画因果性是相当低效的。看来,这一复杂性也是因果性未能被广泛接受的重要原因之一。

八十年代末期和最近的几年,因果性刻画的研究成果还是激励了研究团体。除了前面提到的各种向量时钟方案的提出之外,全局快照问题,谓词检测问题都有了新的理论上的算法和应用研究,这些成果给分布式程序的调试和监控带来了新的思路。此外,因果次序的通讯也在一些系统中被实现,基于因果性的容错和安全研究也有了一些进展。

### 3.3 全局快照

1985 年 Chandy 和 Lamport 在文[4]中讨论了如何使用"happened before"关系来确定全局快照的问题,并以此决定一个因果一致的全局状态,文中还讨论了在一致全局状态上的谓词计算问题。

使用向量时钟可以很容易地确定两个局部快照是否因果相关,但这只是一个必要条件(即局部快照不能因果相关)。最近,文[20]中提出和证明了一组局部快照是否属于一个一致全局快照的精确条件。他们用一个"曲折路径"(zigzag path)的概念扩充了 Lamport 的"happened before"关系,他们的这一成果可以减少数据库中独立检查点的失败回滚扩散(所谓的"多米诺"效应);也可以在分析全局状态时,减少记录因果信息和检查点的空间开销。

### 3.4 分布式谓词检测

分布式全局谓词是一定义在分布式计算中一致全局状态上的性质。为了使得谓词的计算有意义，首先需要获得因果一致的全局状态。与此相关的一类问题是检测计算的行为模式，即特定的非局部事件或事件系列的发生(可看作另一种形式的谓词)。全局谓词从其特性上可分为稳定谓词和不稳定谓词。稳定谓词是那些一旦为真将永远为真的谓词。不稳定谓词是那些在系统的某些状态上为真的谓词。从观察者角度可将全局谓词分为确定谓词和可能谓词。确定谓词是在所有的观察中都检测为真的谓词；可能谓词是一个观察中能检测到为真，而在另一个观察上检测到为假的谓词。此外，对谓词的定义和形式描述也决定着所能检测到的谓词的形式和范围。该领域中的研究和所提出的算法在功能、复杂度、对程序的干扰、正确性以及所采取的策略上都有所不同。以下按大致的时间顺序介绍这些研究成果。

1985 年，Chandy 和 Lamport[4]在提出分布式全局快照概念时，也给出了一个稳定谓词的检测算法。该算法取系统的一致全局快照并检测该快照是否满足全局性质。问题是隔多长的时间取一次快照才能使得谓词被检测为真后系统的现时状态对我们是有价值的，此外，这一算法不能够检测不稳定谓词。

早期的其他工作大都致力于开发描述计算行为的语言机制和实现具体的监控系统，如 EDL(事件描述语言)或单步执行的调试系统。但由于当时对分布式计算行为复杂性理解有限，这些方法都有相当的局限性，或是歪曲了正常的计算行为(如单步执行监控)，或者不是在一个正确的系统状态上进行计算。

1988 年，Miller 和 Choi[17]提出了 Linked Predicate 的概念，其定义如下：

Simple Predicate(SP)::=局部进程上的简单谓词，如 $p_i \cdot x < p_i \cdot y$；

Disjunctive Predicate(DP)::=SP[∪SP]*；([]表示选择、★表示重复)

Linked Predicate(LP)::=DP[→DP]*；

Conjunctive Predicate(CP)::=SP[∩SP]*。

该文中讨论了 Linked Predicate 的检查问题，提出了一个检测到谓词为真后的程序暂停算法，但未讨论 Conjunctive Predicate 的检测。文中没有区别用于检测的消息和程序本身的消息，因而获得的一致状态是不准确的，会导致在有调试器时可能会检查到某谓词为真，而在没有调试器时该谓词可能为假。在上面的定义中，我们可以看到，Simple Predicate 与一般顺序程调试器中的谓词没有什么区别，是局部的，其检测比较简单；Disjunctive Predicate 的满足取决于任一局部谓词的满足；Linked Predicate 是一种行为模式，可以顺序地检测。

1988年，Spezialetti 和 Kearns[24]提出了一个监视分布式计算事件发生的策略方法。这一方法包括分析所识别事件的特性、事件发生的环境特性和识别的需求(时间延迟、系统状态保持与否等)。事件的单调性是文中的一个重要概念，单调事件亦即稳定谓词。这篇文章启发了我们可能找不到一个适应于普遍情形下的谓词检测方案。

1988年，Haban 和 Weigel[11]，1991年 Cooper 和 Marzullo[8]都提出了检测一般性谓词的算法。文[8]的思路是将系统中的全局状态集合考虑成一个格，将系统的执行考虑成格中的一条路径。格的思想是有启发意义的，但需考虑的全局状态将达到 $O(K^N)$ (K 为事件数，N 为进程数)。文[11]的算法除了也有组合爆炸问题之外，它将谓词的满足时间看作是谓词中最后事件的匹配时间，这会导致不一致的现象。

1994年，Gary 和 Waldecker[10]用逻辑方法形式地刻画了一大类谓词。他们考虑那些由局部谓词的合取形式构成的谓词的检测，在检测时，这些谓词可以在局部进程上分解检测。他们的方法可检测顺序和合取谓词，但不能检测两者的混合，这一算法是集中式的，而且需要考虑所有的事件。1994年，Chiou 和 Korfhage[7]提出了事件范式的概念，扩展了 Gary 等的工作，可检测混合谓词，且算法是分布的，考虑的事件也比 Gary 中少一些。事件范式谓词的定义如下：

Primitive Event Predicate(ep)::=单一事件表达式；

Concurrent Event String(CES)::=$ep_1 \wedge ep_2 \wedge \cdots \wedge ep_k$；

Sequential Event String(SES)::=$CES_1 \rightarrow CES_2 \rightarrow \cdots \rightarrow CES_k$；(→表示因果序)

Event Normal Form(ENF)::=$SES_1 \mid SES_2 \mid \cdots \mid SES_n$。(|表示选择)

ENF 的实现需要复杂的语言机制，较之以前的工作，它在形式化地描述谓词上迈出了一步。但这些方法都不能用来检测普遍意义上的谓词，如 $p_1 \cdot x > p_2 \cdot y$，$p_1 \cdot x + p_2 \cdot y = p_3 \cdot z$ 等。

分布式死锁、终止和面向对象程序设计语言中的分布式垃圾问题都是稳定的全局谓词。最近，文[15,21]中都形式地讨论了这类性质。两文中讨论的是一类特殊的稳定性质，在文[15]中被称为 locally stable predicate，在文[21]中被称为 strong stable predicate，这两种谓词是否等价还不清楚，但其优点是一致的，即都可以在非一致快照的一致部分上(一致投影)正确地计算，从而不需要为取得一致快照而付出系统开销。文[21]中证明了分布式终止和死锁是这类性质(即强稳定性质)，而分布式垃圾不是(即弱稳定性质)。

### 3.5 因果序通信

因果序通信这几年在分布式操作系统的学术团体中引起了很大的争论。因果序的消息传递顺序是在多个发送者和多个接收者情形下 FIFO 传递顺序的扩充，以保证发给同一进程的任意两条消息 $m_1$ 和 $m_2$，若 $m_1 \rightarrow m_2$，则 $m_1$ 必然先于 $m_2$ 被接收到。这里列举几个因果序发送的例子来阐述这一问题。

在分布式数据库中，需要管理很多的数据副本。通常是用一个令牌来控制各个副本互斥地修改数据，任一副本的数据被修改后向其他所有副本广播该修改，但是通讯的不确定延迟可能会导致数据不一致。现有的方法是各副本在修改数据后须等待其他所有副本的应答信息之后才能释放令牌给其他要修改数据的副本。若有因果序的发送，这一等待便是没有必要的，它可保证所有副本的修改都是按相同次序(令牌的传递次序)进行。如图1所示：$W_1 \rightarrow A$、$A \rightarrow B$、$B \rightarrow W_2$，故 $W_1 \rightarrow W_2$。对任何的副本，其必然先接收到 $W_1$，然后再是 $W_2$，保证了数据一致性。

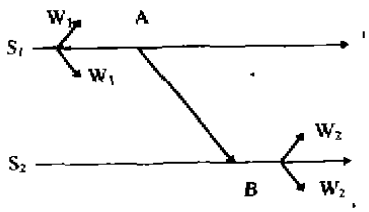


图1 分布式数据副本的因果序更新

在资源分配中，$R_1$ 和 $R_2$ 是两个对同一资源的请求，若 $R_1 \rightarrow R_2$，那么 $R_1$ 应先被授权使用资源。现有的资源分配方案做不到这一点，有了因果序的发送之后，这一点自然地得到了保证，这可防止因资源申请的顺序不一致而产生的死锁现象。

有了因果序的消息传递次序，不同的观察者观察到的系统状态将是完全一致的，亦即此时的一致全局状态可以比较容易地得到，这对分布式计算中的许多问题都会带来简化。因果序的消息传递次序对改进事务系统的性能，简化分布式程序设计语言，提高分布式共享存储的效率和实现容错系统等都有较好的应用。但因果序通信的主要缺点是对通信带宽和通信缓冲的大小要求太高。现已有了很多实现因果序通讯的系统，较有代表性的是 ISIS[3] 中的进程组通讯。

### 3.6 因果性与分布式系统安全

1993年，Reiter[19] 提出正确地检测事件间的因果性对于分布系统的安全性也是重要的。文中从敌对环境中因果性的否认和伪造这两个角度形式地描述了对因果性的攻击，也形式地描述了为阻止类似情况发生应制订怎样的安全制度。文中的思想非常有启发性，列举的事例亦非常有趣。这里举例说明因果性否认和伪造是怎样的一种安全性破坏(见图2)。

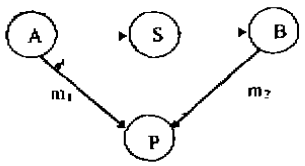


图2 因果性否认与伪造

假设 $A$，$B$ 为证券商，$P$ 为交易中心，现 $A$ 向 $P$ 发一委托请求购入某股票(消息 $m_1$)，$B$ 知道了该请求(直接或间接的通讯或窃听，虚线所示)，此时 $B$ 也向 $P$ 发一委托请求卖出股票(消息 $m_2$)。若 $P$ 不能识别 $m_1$ 与 $m_2$ 之间的因果关系，认为 $m_1$ 与 $m_2$ 是并发的请求，那么 $P$ 可能先接收 $m_2$，后接收 $m_1$，从而可能造成 $A$ 受损，$B$ 受益。这种情形被称之为 $B$ 否认了 $m_1$ 对自己的影响，即否认因果性。

同一图例中，假设 $A$ 是一上市公司，它宣布将与另一家公司合并(其股价将上涨)；$B$ 是一证券商，若 $B$ 通过一内部消息在 $A$ 宣布之前得知了这一消息，并大量购入 $A$ 的股票，但为了避免被怀疑，$B$ 使得自己的购入请求 $m_2$ 看起来像是 $m_1$ 宣布之后生成的。若 $P$ 不能识别此情形，$B$ 将非法受益。这里 $B$ 伪造了 $m_1 \rightarrow m_2$ 这一关系，称之为伪造因果性。这类问题值得进一步的研究，但需要寻找合适的解决方法。

## 4 对因果性研究的总体认识

分布式程序是难于开发和分析的，这是由于其固有的特性引起的：并发性、非确定性、没有全局状态和全局时钟。这些概念还没有被我们所完全掌握，我们还没有找到处理分布式计算中复杂因果结构的合适方法，这是导致缺乏分布式程序设计和分析工具的一个原因。从理论角度看，对分布式计算中因果性的理解已经比较明朗了，近年来的研究成果已经大大地丰富了我们对分布式问题的认识，但是这些成果仍然没有为我们最终解决这些问题并开发相对成熟、有效的分布式程序设计工具和环境带来太大的希望，我们认为主要有以下的原因：

首先，对于论证分布式计算中的因果性，目前尚无完整、一致公认的形式化方法，各种文献中的系统模型参差不齐，缺乏共同的基础，这种混乱非常不利于知识和经验的交流，甚至往往难以评价所取得的成果。很多的文献中存在着概念不清的问题，如往往没有考虑到不同的观察者会观察到不同的系统状态；状态和状态转换，原子和非原子行为也常常混为一谈。

其次，因果结构的内在复杂性也导致了难以有效地利用现有的理论成果，难以开发出有效的工具，像向量时钟的使用还仅限于学术交流中。

最后，也可能是最重要的原因，就是理论上还没有取得令人鼓舞的突破。

总之，分布式计算的复杂性远远地超出了我们的想像。分布式程序设计还停留在实验室中，还没有成为一门实用的技术。

## 5 对因果性研究的展望

基于以上的认识，我们认为因果性的研究目前正处于一个徘徊阶段。我们觉得目前的工作可以对解决这类问题的严格性适当放松一些要求，这是一个很好的发展方向，具体地说，可以有如下两个方面。

其一是放松因果关系。应该指出的是目前所讨论的因果性并不是语义上的因果性，而是一种潜在的因果性。例如发生在同一进程上的事件是因果关系上的一个完全序，但这些事件未必真的因果相关。现在绝大多数的研究只考虑了这种潜在的因果性。原因之一是因为单一进程中的事件序是由局部的程序控制器所唯一决定的(PC→PC+1)，因而虽然没有强加上因果性，但对于任何观察者而言都会观察到这一序。原因之二是，从技术角度看，在因果性跟踪和确定两个事件是否因果相关上，使用实际的因果性无论从语义分析上，还是从处理的方便上来说都较之潜在的因果性要累赘得多。因此，可以想像，当涉及到实际的因果性研究时，我们所要面临的问题较之现在的问题要复杂得多。当然，使用实际因果性的好处在于允许我们开发分布式程序的更高并发性，异步能力和允许我们更自然地处理事件、行为和行为模式，发现处理实际因果性的分析方法和更复杂的描述结构是一个有意义的挑战。

在实际应用中，应用层的业务因果性是一种能较好地定义和操作的性质，充分利用因果性研究的成果应能获得具有可用性更高的业务系统。

另一个考虑是试图发现一种时间戳方案，其产生的偏序事件较之向量时间严格，但较之 Lamport 的全序逻辑时钟要松一些，这样便可以降低时间戳的复杂度，换句话说，就是用准确性换取计算的方便。但是这需要重新考虑分布式计算中的事件定义，现有定义下的理论结论[6]已经证明了，不可能有比向量时钟更简单的刻画因果性的方案。从另一角度考虑时间戳方案。若保持现有的向量时钟，充分地利用现有的计算资源(如实时时钟，通讯处理层，底层硬件)，也可以降低在应用程序层次上维持因果性刻画的实际复杂度，我们已提出了一种基于实时时钟的向量时钟，有一定的实际应用价值。

再从另一个方面考虑因果复杂性。我们也可以放松现有计算模型的通用性，通过限制系统的特性和问题范围来达到有效地计算因果性的目的。如现已证明，加上 FIFO 的通信信道可以降低向量时间占用的带宽(只发送上次发送之后的改变量)。也许我们可以研究一套策略方法，针对不同的系统特性和问题范围来处理因果性。

总之，现有的研究还没有找到一种能够成熟有效地作为分析因果性的通用机制。预测分布式程序的行为，给一个一般的谓词一个有意义的语义以及发现检测它们的有效算法，仍然是一个巨大的挑战。从直觉上来看，这一问题是很难解决的，但是我们可以在开发用于这一目标的工具时结合有限的自动检测能力与人的直觉与灵活性。

(参考文献共25篇略)

并发系统模型 软件开发 数学模型

计算机科学1996Vol.23№.5

# 并发系统模型研究

贾国平 郑国梁
(南京大学计算机科学系 南京210093)

**摘 要** In this paper, we present discussion and analysis of various existing models of concurrent systems, such as finite-state machine, hierarchy of functions, Statecharts, Petri-Net, graph model of computation and fair transition system model. As a result of discussion, we know that different properties are expressed by different models of requirements. Our work is very helpful for software developers, especially for systems analysts to choose proper models in their problems and it provides the foundation for further applications of various models.

**关键词** Model, Concurrent system, Finite-state machine, Hierarchy of functions, Statecharts, Petri-Net, Graph model of computation, Fair transition system.

## 1. 引 言

一个模型的作用是对想要构造或分析的系统性质给出严格定义，同时也为验证这些性质提供一个基础。现有的用于软件开发的每一种形式化描述和分析方法其有效性范围均是相对于程序执行的某一数学模型而言，并不是实际的系统。尽管这些方法对于其模型是完全有效的，但它们对于程序实际的应用却只能由其对应模型表示实际程序执行的可信度来决定。通过一种形式化方法所得到的关于程序执行的结论，在某种程度上，只有当其基本的数学模型是可靠的，它才是可靠的。因此在开发形式化方法过程中，很重要的一点是它不仅应该包括构造新的模型和用于分析此模型的技术，而且还应该包括不断对模型之间以及它们与实际系统之间进行比较和评价。

我们对不同的并发系统模型进行了分析和讨论。不同于顺序系统模型，并发系统模型除了表示顺序程序的所有特征之外，很重要的一点是它还必须很好地表达并发性。文中我们就目前普遍用于说明并发系统性质的不同模型，如有穷状态机、函数层次模型、Statecharts、Petri-网、计算图模型及公平转换系统模型等的优点及不足进行了分析。由此分析和讨论，我们知道不同的性质可以由不同的需求模型表示。为了表达并发系统的特殊性质，我们必须要选择合适的模型加以表示。本文的工作有利于软件开发人员，特别是系统分析人员选择合适的模型以便于更好地对不同问题进行说明描述。它为不同模型的进一步应用提供了基础。

## 2. 并发系统模型

### 2.1 有穷状态机模型

有穷状态机模型(FSM)是计算机科学中的一个最基本的模型，已经广泛应用于通信协议的规范及验证之中，同时也是几乎所有证明技术及系统分析技术的核心[1]。

一般，一个 FSM 模型由这样几个部分组成：①Input：输入集合。②Output：输出集合。③S：状态集合。④$s_0$：初始状态。⑤$T_{state}$：状态转换函数：S×Input→S。⑥$T_{transf}$：变换函数：Input→Output。

这一模型非常适用于对数据处理问题的描述。数据处理的输入及输出分别对应于 FSM 中的 Input 和 Output。处理器内存及寄存器内容与 FSM 中状态相联，而执行代码对应 FSM 中的变换函数。

然而，FSM 用于说明并行处理时有两个主要的缺陷：其一，FSM 模型本质上将所有基本的并发进行了顺序化处理。模型显式地假设在下一输入到达之前，前一输入点上的所有处理都已完成。因此，当此模型用于并发系统的规范时，其关键问题是它忽略了冲突状态，即两个输入同时到达的状态。这也就导致了一个不合适模型上的正确性证明。在这方面，现在已出现了许多工作，它们主要通过采用多个并

贾国平 博士生。 郑国梁 教授、博士生导师。

行FSM对FSM模型进行扩充用以对并发性进行描述。其二，FSM模型本身并无一个机制用于处理复杂性。此模型是“平坦”的，并无一个层次，当所要描述的进程的状态个数过大时，如果没有一个辅助结构，FSM描述将很难理解。即使对于小型问题，FSM中对引起状态改变的状态转换函数及变换函数的描述也非常复杂。因此，一些机制需要引入FSM模型中以便将两个函数分解为更小、更易于理解的单元，易于实际构造。

### 2.2 函数层次模型

函数层次模型是数据处理中最常用的一种说明方法，在说明系统时，首先要说明一个系统函数，此函数有一个由所有可能输入数据组成的输入论域和一个由所有可能输出数据组成的输出论域。然后，此系统函数分解成一个相互交互的函数集合，它们之间具有输入/输出关系，而每一交互函数又可逐个被进一步分解为更低一级相互交互的函数。分解过程中，数据和函数都同时被分解。函数层次模型是许多软件规范技术(如SADT和PSL/PSA)的基础。

函数层次模型最大的优点在于它对大量数据的组织能力以及它对高级函数同它分解而成的众多低级交互函数之间相容性进行检查的能力。这一过程可以重复下去直到得到一个任意级的函数层次。这就解决了FSM模型中无法表达的并发及复杂性问题。

然而，函数层次模型也有其不足，主要是它只表示了数据流，而对控制流并未进行表示，特别是，执行顺序和条件信息并未在此模型中说明并且也不可能用此模型进行验证。这就限制了函数层次模型的作用，同时缺乏对并发性更细致的表示能力。

### 2.3 Statecharts模型

Statecharts模型是由D. Harel[1987]提出的用于描述复杂并发系统(及反应系统)的一种可视化形式。它通过状态、事件和条件对系统行为进行描述。而后两者的组合形成转换。从而导致状态之间的转换。

Statecharts模型本质上可看成是对传统的FSM模型及状态图(State Diagram)模型的推广，是Moore有穷状态自动机和Mealy有穷状态自动机的某种组合。此模型中的允许序列对应于由自动机所接受的语言。另外，Statecharts模型完全遵循了标准状态图的可视化表示。

具体地说，Statecharts模型从三个方面对传统的FSM模型及状态图模型进行了扩充。首先，此模型中采用“与/或”(AND/OR)状态分解/聚集机制。状态可以反复使用“与/或”状态聚集机制而被组合成高级状态。同样，高级状态可以通过反复使用“与/或”分解机制而被精化为低级状态。模型中的转换并不受状态级的限制，它可以从任何聚集级状态到其它任何聚集级状态。事实上，转换一般是从一个格局(Configuration)到另一个格局，因此模型获得了层次描述能力。其次，Statecharts模型采用“与”状态分解机制。如果系统处于某一状态，则通过使用“与”状态分解机制后，系统必须处于所有分解得到的“与”部件状态之中。此机制获得了并发系统中部件之间的并发性描述。最后，Statecharts模型中对并发部件之间采用广播通信机制，更加易于实际描述。这几方面的扩充使得Statecharts模型成为一种高度结构化、简洁且非常适用于复杂并发系统的模型，目前已经得到了较广泛的应用，尤其适用于复杂反应系统的规范及验证。

### 2.4 Petri-网模型

Petri-网模型[2,3]提供了一种形式表示顺序和并发以及标识和解决歧义性的方法。它是通信协议的规范及验证中的一个有效工具。

在Petri-网模型中，我们区分两类节点：位置节点和网转换节点。形式上，一个Petri-网模型是一个三元组：(P,T,F)，其中：P是位置节点集合，每一位置节点定义有向图中标记可以进驻的位置。T是网转换节点集合，每一网转换节点控制前一位置节点中的标记向下一位置节点的转换。如果网转换节点的条件满足，则此网转换就被执行。$F \subseteq (P \times T) \cup (T \times P)$是连接边集合。对一个简单的Petri-网中的网转换，当其前置条件满足时，此网转换被称为点火(Firing)，上一位置节点中的标记迁移至下一位置节点。

Petri-网模型能够用于精确地说明顺序、并发和同步等概念。另外，通过定义一个Petri-网处理器。我们可以利用Petri-网进行自动模拟。这个处理器检查所有网转换的状态，选择一满足输入条件的位置节点，然后根据后置条件移动标记。此步骤可以重复下去，直到不存在满足前置条件的网转换为止。系统分析人员通过观察标记的活动来对所描述进程的行为进行分析。并发系统中许多有趣的性质，如正确性终止、无死锁性、互斥等问题都可以用Petri-网得到定义和解决。

尽管Petri-网模型有上述优点，但用于描述并发系统存在两个缺陷。首先，它与FSM模型一样是平坦的，即Petri-网本质上只有一层，并无一个层次结构。许多研究工作中已经提出了Petri-子网的概念。子网层次的概念可以较好地处理大型复杂问题。这一方法使用得合适，并不会影响Petri-网本身的

语义。其次，Petri-网模型只说明系统的控制流而对数据流并未予以说明。即使网转换条件由数据值来表示，改变和存储数据变量值的语义仍然并不属于Petri-网内在的一部分。对此问题，目前也有不少的研究工作，如文[4]。

### 2.5 计算图模型

计算图模型是一个带节点及弧的有向图，事实上，它是Petri-网模型的一种扩充，其中转换和位置被有效地组合成某一节点类型，模型中每一节点表示一个处理步。它可以有一个或多个输入弧，也可以有一个或多个输出弧。为了进行转换，一个输入函数用来定义说明是否所有或者只有一个输入弧是能行的。在转换完成点，一个结束函数说明是否所有或者只有一个输出弧是能行的。它包含类似于Petri-网模型中关于转换的说明。然而，不同于Petri-网模型的是在计算图模型中每一节点都有一个从某一论域中定义的输入和输出，它用于说明节点之间的数据流。因此，计算图模型不仅说明了控制流，同时它还说明了数据流，它为两者的相容性检测提供了基础。在计算图模型中，由于其控制流是用与Petri-网模型相同的形式表示的，因此许多性质，如正确性终止、无死锁性等也能够在此模型中进行分析。另外，Petri-网模型中子网的概念同样也可适用于计算图模型，它为计算图模型提供了一个层次分解机制。

计算图模型最大的缺陷就是：虽然它可以很好地用于描述数学函数的分解，但是本质上并不适用于对时间序列转换的描述，因为在此模型中并无状态的概念。因此计算图模型并不非常适用并发系统的描述。

### 2.6 公平转换系统模型

公平转换系统模型(FTS)最初由Z. Manna和A. Pnueli[1983]给出。它一方面可以作为并发系统的抽象计算模型，另一方面也可以作为一种抽象的实现语言。此模型包含了许多具有不同语法表示及通信机制的并发系统，它与许多具体程序设计语言之间的对应关系可参见文[5]。

一个公平转换系统S是一个六元组$(V,\Sigma,\Theta,T,WF,ST)$，其中，$V$：状态变量的有穷集合。$\Sigma$：状态集合。$\Theta$：初始条件。对一个状态$s\in\Sigma$，如果它满足$\Theta$，即$s\models\Theta$，则我们称$s$为初始状态。$T$：有穷转换集合。每一转换$\tau\in T$是一函数：$\Sigma\to 2^{\Sigma}$。$WF\subseteq T$：弱公平性转换集。$SF\subseteq T$：强公平性转换集。

一个转换在状态$s$下是能行的如果$\tau(s)\neq\varnothing$，否则$\tau$在此状态下是不能行的。

$\Sigma$中的无穷状态序列$\sigma:s_0,s_1,\cdots$被称为是公平转换系统S的一个计算(Computation)，如果$\sigma$满足：①初始化条件：$s_0$是初始状态，即$s_0\models\Theta$。②连续性：对每一$j=0,1,\cdots$，状态$s_{j+1}$是状态$s_j$的$\tau$-后继状态，即存在转换$\tau\in T$，使$s_{j+1}\in\tau(s_j)$。③弱公平性：对每一转换$\tau\in WF$，不会发生$\tau$在$\sigma$中某一位置以后一直能行，但只有有限多次被执行。④强公平性：对每一转换$\tau\in SF$，不会发生$\tau$在$\sigma$中无穷多次能行，但只有有限多次被执行。

由上可知，FTS模型本质上也是FSM模型的扩充，其中很重要的一点就是此模型中并发性是通过交替执行(Interleaving)来表示的。两个并行进程永远不会在同一时间执行它们的语句，只会交替执行其原子转换。形式上，当一个并行进程正在执行其原子转换时，其它进程中的原子转换都处于非激活状态。FTS模型中通过对转换进行不同的公平性限制而确保并发程序执行中，所有进程都参与了执行，避免了某些计算中只有一些进程被执行，而其它进程都没有机会被执行的不公平情形，从而解决了实际并发程序执行中进程间独立执行问题。在这样一个并发性交替表示模型中，并发系统规范及验证的理论就非常简单。另外很重要的一点是FTS模型为时序逻辑很好地应用于并发系统的规范及验证提供了基础。

然而，上述FTS模型中有一个很重要的干扰(Interference)问题，FTS模型本质上是一个交替模型，要求在一个转换执行过程中无干扰发生。为了使FTS模型能够完全描述并发系统的实际执行，必须要对FTS模型中原子转换及其粒度划分施加限制。文[6]中，通过对程序原子执行语句(即原子转换)进行限制临界引用约束，即LCR条件，而对干扰问题进行了处理。由此约束条件可以得到结论：FTS模型是可信的，它能够完全对并发系统的实际执行进行描述。

## 主要参考文献


[1] T. F. Piatkowski, The State of the Art in Protocol Engineering, In: Proc. ACM Sigcomm.' 86 Symp., Stowe, VT, 1986

[2] J. Peterson, Petri Net Theory and the Modeling of Systems, Prentice Hall, 1981

[3] N. Reisig, Petri Nets, EATCS Monographs, 1985

[4] S. Yau, M. Caglayan, Distributed software Design Representation Using Modified Petri Nets. Trans. on Software Engineering, 1983

[5] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent System: Specification, New York: Springer-Verlag, 1991

[6] 贾国平、郑国梁，论公平转换系统模型的可信性，《计算机科学》1996年，第23卷，第2期