分布式系统中的虚拟时间与全局状态 *

Friedemann Mattern †
计算机科学系，凯撒斯劳滕大学
德国 6750 凯撒斯劳滕

2025 年 10 月 15 日

摘要

一个分布式系统可以由以下两个特征来表征：全局状态是分散的，且不存在共同的时间基准。然而，时间的概念在我们去中心化的“现实世界”日常生活中至关重要，它有助于解决诸如如何获得一致的人口普查数据或确定事件之间潜在因果关系等问题。我们认为，线性有序的时间结构对于分布式系统来说并不总是足够，因此我们提出了一种广义的非标准时间模型，该模型由时钟向量构成。这些时钟向量是部分有序的，并形成一个格结构。通过使用时间戳和一个简单的时钟更新机制，可以以同构的方式表示因果关系的结构。这种新的时间模型与闵可夫斯基的相对论时空具有密切的类比关系，并为全局状态问题提供了有趣的刻画。最后，我们提出了一种新的算法，用于计算分布式系统的一致全局快照，即使消息可能以非顺序方式接收。

1 引言

一个异步分布式系统由若干没有共享内存的过程组成，它们仅通过具有不可预测（但非零）传输延迟的消息进行通信。在这样的系统中，全局时间与全局状态的概念起着重要作用，但它们的实现十分困难，甚至其定义本身也并不清晰。由于一般情况下系统中的任何过程都无法立即且完整地获取所有过程的状态，因此一个过程只能近似地获得一个理想化外部观察者（该观察者能够立即访问所有过程）的全局视图。

系统中事先不存在任何过程对全局状态的一致视图，也不存在共同的时间基准，这正是分布式系统中大多数典型问题的根源。操作系统和数据库系统中的控制任务，如互斥、死锁检测和并发控制，在分布式环境中比在传统的集中式环境中要困难得多，而且大量针对这些问题的分布式控制算法被发现是错误的。在分布式系统中还出现了在集中式系统或具有共享内存的并行系统中不存在的新问题。其中最突出的问题包括分布式一致性、分布式终止检测以及对称性打破或选举问题。这些问题的解决方案具有极大的多样性——其中一些方案甚至非常优美而巧妙——这令人惊叹，充分体现了分布式计算在应对全局状态和时间缺失方面的诸多原则。

由于异步系统中算法的设计、验证和分析具有困难且易出错的特点，可以尝试以下几种方法：

1. 在给定的异步系统上模拟一个同步分布式系统，2. 模拟全局时间（即一个共同的时钟），3. 模拟全局状态（即共享内存），

并利用这些模拟特性来设计更简单的算法。第一种方法通过所谓的同步器 [1] 实现，其方式是模拟时钟脉冲，使得消息仅在时钟脉冲时生成，并在下一个脉冲前被接收。同步器旨在作为异步系统之上的一个透明的软件附加层，以便执行同步算法。然而，这种机制的消息开销相当高。第二种方法不需要额外的消息，系统在本质上仍保持异步，即消息的传输延迟是不可预测的。传输延迟。Lamport [9] 提出了这种方法。他展示了如何通过逻辑时钟实现的虚拟时间来简化分布式互斥算法的设计。Morgan [11] 和 Neiger 与 Toueg [12] 进一步发展了这一思想。Chandy 和 Lamport 在他们的快照算法 [3] 中继续了这一方法，该算法是分布式计算中的一个基本范式。Panangaden 和 Taylor [14] 进一步阐述了这一思想，从而得出了“并发共同知识”的特征。

显然，全局时间与全局状态的概念密切相关。通过 Chandy 和 Lamport 的算法，一个进程可以在不“冻结”整个计算过程的情况下，计算出全局状态的“最佳可能近似”——即如果所有进程同时对本地状态进行快照，则可能发生的全局状态。尽管系统中没有任何一个进程能够确定快照状态是否真的发生，但“可能发生”对于稳定属性（即一旦变为真就保持为真的全局谓词）如终止检测和死锁检测来说已经足够，这表明快照算法是解决这些问题的通用方案。（然而，这并不削弱针对这些

问题的特定算法，这些算法通常更简单且更高效）。虽然在某种意义上，快照算法计算出了可达到的最佳全局状态近似，但 Lamport 的虚拟时间算法并非如此完美。事实上，通过将分布式计算中部分有序的事件映射到一个线性有序的整数集合上，该方法会丢失信息。那些可能同时发生的事件可能会被赋予不同的时间戳，仿佛它们按某种确定的顺序发生。对于某些应用（如 Lamport 本人在 [9] 中描述的互斥问题），这种缺陷并不明显。然而，对于其他用途（例如分布式调试），这种缺陷是重要的。

本文旨在改进 Lamport 的虚拟时间概念。我们认为，线性有序的时间结构并不总是适用于分布式系统，而由向量构成的、具有格结构的部分有序系统是分布式系统中时间的自然表示。这种非标准的时间模型在许多方面类似于闵可夫斯基的相对论时空。特别是，它具有更广的“同时性”范围——所有非因果相关的事件都是同时发生的，从而以同构的方式表示因果关系，且不丢失信息。新的时间概念结合一个广义的时钟同步算法，使得每个进程能够获得外部观察者的“理想化”时间的最佳近似值。（这种理想化的全局时间自然地定义为所有局部时钟向量的上确界）。时间的向量结构

将被证明与快照算法所构造的可能状态结构是同构的，从而提供了全局时间与全局状态的互补视角。

2 事件结构

在抽象设定中，一个进程可以被视为由一系列事件构成，其中事件是本地状态中不耗费时间的原子转换。因此，事件是发生在进程上的原子动作。通常，事件被分为三类：发送事件、接收事件和内部事件。内部事件仅导致状态的改变。发送事件导致消息被发送，接收事件导致消息被接收，并通过消息的值更新本地状态。然而，在所谓的消息驱动的分布式计算模型（例如，演员模型）中，只有一种类型的事件：接收到消息会触发一个原子动作的执行，该动作导致本地状态更新，并可能向其他进程发送任意数量的消息。由于其简洁性，消息驱动模型在抽象层面具有吸引力。

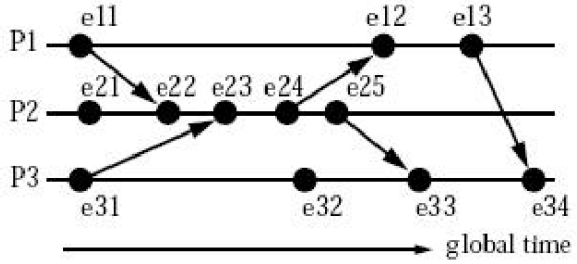

图 1. 事件图

事件之间存在关联：特定进程上发生的事件按照其本地发生顺序被完全排序，且每个接收事件都有一个对应的发送事件。这种关系是任何虚拟时间概念的核心。然而，核心概念似乎是因果关系，它决定了时间的基本特征，即未来不能影响过去。形式上，事件结构 [13] 是一个对 $(E, <)$，其中 E 是事件的集合，'$<$' 是 E 上的一个无自反的偏序关系，称为因果关系。事件结构以抽象方式表示分布式计算。对于给定的计算，若满足以下任一条件，则 $e < e'$ 成立：

(1) e 和 e' 是同一进程中的事件，且 e 在 e' 之前发生，

(2) e 是某消息的发送事件，e' 是对应的接收事件，

(3) $\exists e''$ 使得 $e < e''$ 且 $e'' < e'$。因果关系是满足这些条件的最小关系。

将此定义理解为一个图（图 1）是有帮助的。显然，$e < e'$ 表示事件 e 可以因果地影响事件 e'。图示上，这意味着可以在图中从事件 e 到事件 e' 跟踪一条“因果路径”(沿着箭头方向，从左到右沿过程线移动)。

可以将类似于图 1 的图视为一个实际计算的时间图，其中水平方向表示真实时间。那么，图中由点表示的事件在某个特定时间瞬间发生，这是理想化外部观察者所观测到的。消息由对角线箭头表示。(一个能够持续观察消息在前往目的地过程中的观察者，可以绘制出更精确的实际消息流动图。请注意，二维时空图表示了一维空间中物体相对于时间的位置轨迹)。此外，也可以将该图仅视为部分序 $(E, <)$ 的抽象偏序图。通常，偏序图是通过将元素 e 放在 e' 之上（当 $e' < e$ 时）来绘制的。(由于假设了传递性，只绘制直接相关元素之间的连接，省略冗余连接)。图 2 显示了图 1 计算的偏序图。显然，这两个图是同构的。然而，图 1 似乎隐含地指出了全局时间，描绘了一个特定计算，而图 2 仅显示了事件的逻辑关系，即计算的因果结构。

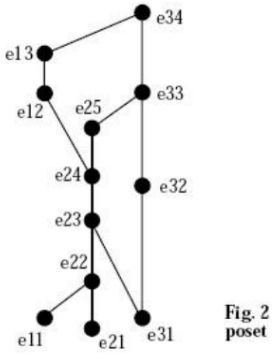

图 2. 偏序图

图 3 显示了一个与图 1 中所示图非常相似的图。请注意，部分事件顺序相同(即具有相同的偏序图)，但在此图中，事件 e_{12}, e_{25} 和 e_{32} 发生在同一全局时间瞬间。这类显示相同因果关系的图将被称为等价的。

显然，可以通过拉伸和压缩表示局部时间轴的过程线，将一个时间图转换为另一个等价图。通过假设过程线由理想化的弹性带组成，可以得到时间图等价变换

的操作视角。只要表示消息流动的箭头不反向（即从右向左）在时间上，任何由给定时间图 d 通过拉伸和压缩弹性带构造出的时间图 d' 都与 d 等价。（显然，消息在时间上反向流动的时间图并不表示任何可实现的情况。在任何“有效”的时间图中，当 $e < e'$ 时，事件 e 必须绘制在事件 e' 的左侧——这是从偏序图到实际时间图转换的唯一限制）。请注意，时间图上的“弹性带等价变换”正是那些保持因果关系不变的变换。

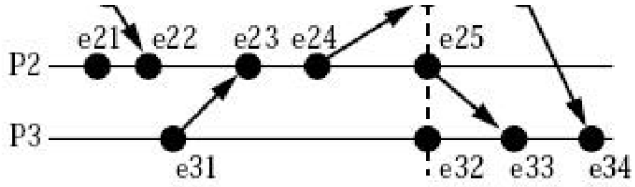

图 3. 一个等价图

3 一致切片

如果一个进程向所有其他进程发送消息以启动局部操作或获取某些分布式信息，这些消息通常会在不同的时间瞬间被接收。由于消息延迟不可预测，无法保证由这些消息触发的所有局部操作同时执行。这促使了“切片”这一概念的提出。

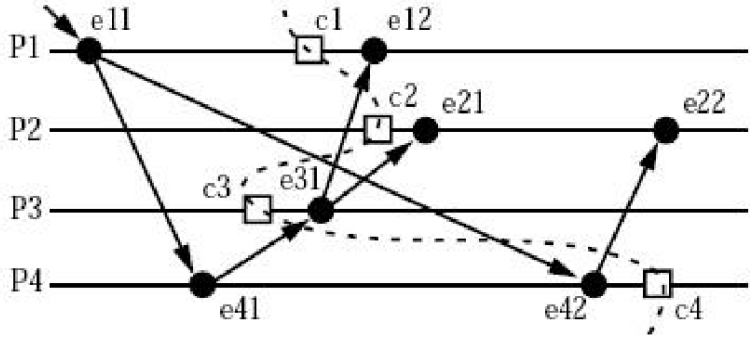

图 4. 一个切片

图示上，一个切片（或时间截面）是一条锯齿线，将时间图分为两部分——左部分和右部分（图 4）。形式上，我们将事件集合 E 扩展，为每个进程 P_i 增加一个称为切片事件 c_i 的新元素：$E' = E \cup \{c_1, \ldots, c_n\}$（$n$ 表示进程数量）。在图中连接这些事件得到切片线。一个切片将 E 分为两个集合：PAST（在切片之前发生的事件）和 FUTURE（在切片之后发生的事件）[2]。设 $<_l$ 表示局部事件顺序（即 $e <_l e'$ 当且仅当 $e < e'$ 且 e 和 e' 发生在同一进程上）。那么，我们可以形式化地将一个切片与其 PAST 集合对应起来，得到以下定义：

定义 1 一个事件集 E 的切 C 是一个有限子集 $C \subseteq E$，使得 $e \in C \& e' <_l e \rightarrow e' \in C$。

这里我们丢失了切事件 $c_1, \ldots, c_n$。然而，将由定义 1 所定义的“切”与给定的切事件集合相互关联是显而易见的。根据上下文可以清楚地判断，我们所说的“切”是指过去集（PAST）还是切事件。图 5 显示了图 4 所示切的偏序图，其中包含集合 PAST。

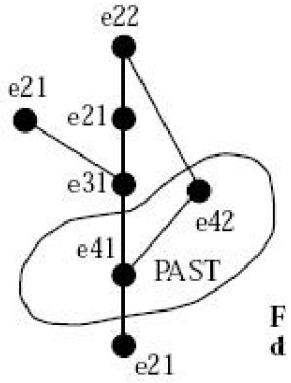

图 5. 图 4 的偏序图。

以下定义已经预示了时间的概念：

定义 2 若 $C_1 \supseteq C_2$，则切 C_1 比切 C_2 更晚。

在图中，一个切 C_1 比切 C_2 更晚的切线位于 C_2 切线的右侧。（在图 6 中，C_3 比 C_1 和 C_2 都晚。然而，C_1 并不比 C_2 晚，反之亦然）。注意，“更晚”是自反的，即一个切比自身更晚。“更晚”是切集合上的一个偏序关系。此外，它构成一个格：

定理 1 对于一个偏序事件集 E，其切集合在运算 $\cup$ 和 $\cap$ 下构成一个格。

证明是直接的。回忆一下，格是任意两个元素都有最大下界（inf）和最小上界（sup）的偏序集。显然，对于任意两个切 C_1, C_2，有 $\text{inf} = C_1 \cap C_2$ 且 $\text{sup} = C_1 \cup C_2$。

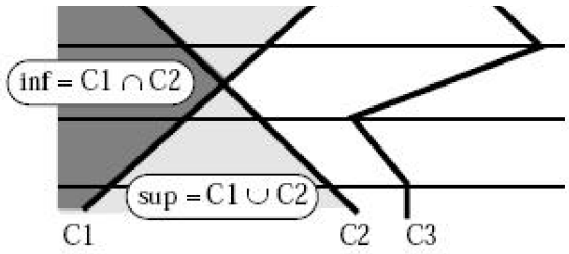

图 6. 切的格结构

如果没有相关规定，可能会出现切包含消息的接收事件，但不包含其发送事件的情况。这种情形是不可接受的，因为切被用来计算分布式系统沿切线的全局状态（第 4 节）。以下定义通过要求切在因果关系 $<$ 下是左闭的，排除了这种不一致的切：

定义 3 一个事件集 E 的一致切 C 是一个有限子集 $C \subseteq E$，使得 $e \in C \& e' < e \rightarrow e' \in C$。

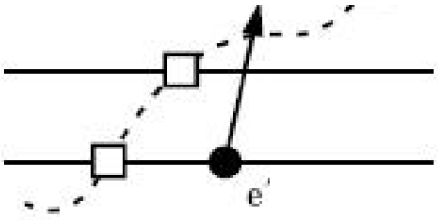

图 7. 一个不一致的切

如果每个接收到的消息都曾被发送过（但不一定反之），则称切是一致的。图 4 展示了一个一致的切。图 7 所示的切是不一致的：事件 e 属于该切，但其直接前驱事件 e' 不在其中。显然，由于 $e <_l e'$，一致切的集合是某个偏序集 E 的所有切的集合的一个子集。但这还不是全部：

定理 2 一致切的集合是所有切集合的子格。

一致切在 ∪(sup) 和 ∩(inf) 下封闭的简单证明留作读者练习。

一致切的格结构保证了对于任意两个一致切 C_1, C_2，总存在一个比它们都晚的切，以及一个比它们都早的切。这一点推广到任意有限个一致切：$\sup(C_1, \ldots, C_k) = C_1 \cup \ldots \cup C_k$ 比 $C_1, \ldots, C_k$ 都晚。

以下两个定理未给出证明（但参见 [14] 和 [3]）。其图形解释是显而易见的。假设切线也是一根橡皮筋。现在将其拉伸，使其变为一条竖直的直线。如果此时一条消息箭头从右向左穿过它，则该切是不一致的；否则是一致的。图 8 展示了不一致切的“橡皮筋一致性测试”（注意 $c_3 < e' < e < c_1$，因此 $c_3 < c_1$，从而违反了定理 3 的条件）。

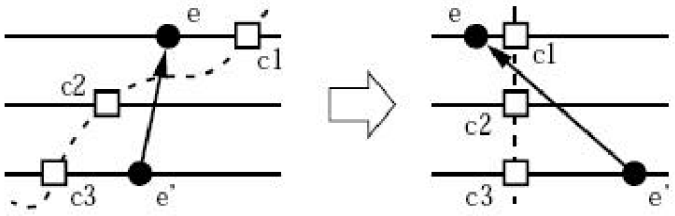

图 8. 橡皮筋测试

定理 3 对于由切事件 $c_1, \ldots, c_n$ 组成的一致切，任意一对切事件之间不存在因果关系。lated，即对所有切割事件 c_i, c_j 都有 $\neg(c_i < c_j)$&$\neg(c_j < c_i)$。

考虑到前一节末尾的说明，可以发现一致切割是“可能的”：

定理 4 对于任何包含切割事件 $c_1, \ldots, c_n$ 的一致切割的时间图，存在一个等价的时间图，其中 $c_1, \ldots, c_n$ 同时发生，即切割线形成一条竖直的直线。

4 一致切割的全局状态

如定理 4 所示，一致切割中的所有切割事件都可以同时发生，即存在一种分布式计算的潜在执行，在这种执行中，所有切割事件在“真实时间”上确实是同时发生的。在相同（真实）时间点对所有进程的局部状态进行快照，显然是一致的。因此，沿一致切割计算出的全局状态是“正确的”。

一致切割的全局状态包含每个进程在切割事件发生时刻的局部状态，以及所有已发送但尚未接收到的消息集合。在时间图中，这些消息由从左侧跨越切割线到右侧的箭头表示。

快照问题在于设计一个高效的协议，该协议仅产生一致切割，并收集局部状态信息。此外，以某种方式必须捕获跨越切割的消息。Chandy 和 Lamport 首次提出了这样的算法，假设消息传输是 FIFO 的 [3]。我们在第 11 节提出一个类似的算法，适用于非 FIFO 通道。

5 时间的概念

正如 Lamport 所指出的，时间的概念是我们思维方式的基础 [9]。事实上，“真实时间”有助于解决我们去中心化现实世界中的许多问题。例如，考虑获取一致的人口普查问题：大家约定一个共同的时间点（足够远的未来），然后在同一个时刻对所有人进行统计。时间也是一个有用的工具，用于考虑可能的因果关系。例如，如果某人因在犯罪发生时间附近某个时刻远离犯罪现场而有不在场证明，那么他或她就不可能是罪犯（图 9）。

这些例子之所以成立，是因为真实时间的特性。通过使用时钟，事件 e, e' 可以被标记时间戳 $t(e)$ 和 $t(e')$，使得当 e 在 e' 之前发生并可能因此因果影响 e' 时，有 $t(e) < t(e')$。根据 Lamport 的观点，能够构建一个独立运行的时钟系统来观察因果关系，是“宇宙中的一个谜”[9]。

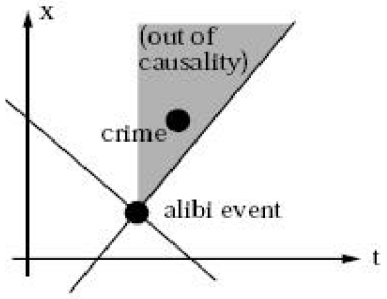

图 9. 时间与因果关系

在没有真实时间时钟的异步分布式系统中，不存在共同的时间基准。但如果我们能拥有一个具有真实时间大多数特征的近似时间基准，那么分布式算法的设计将大大简化。Morgan [11] 的想法是将分布式算法分解为两个独立的算法：

(1) 一个全局时间对所有进程都可用的算法。

(2) 一个实现虚拟时间的时钟同步算法，作为全局时间的合适近似。

这种关注分离有助于设计新的分布式算法：一旦想法被找到并加以完善，就应能够将这两个组件结合成一个优化的单一算法。

然而，这些陈述较为模糊，并引发了一些问题：虚拟时间究竟指什么？什么样的“最佳可能近似”可以视为全局（即真实）时间？如果一个算法是基于真实时间的存在而编写的，当使用虚拟时间时它是否仍然正确？真实时间的本质结构是什么？这些问题将在后续中进一步探讨。

时间的本质被哲学家、数学家、物理学家和逻辑学家所研究。对我们而言，感兴趣的是其在模型论意义上的形式结构，而普遍的数学图景是“标准时间”为一组“瞬间”构成的集合，具有一个时间先后顺序 $<$（“早于”），并满足某些明显条件 [17]：

(1) 传递性。

(2) 无自反性（注意传递性和无自反性蕴含不对称性）。(3) 线性性。

(4) 永恒性 $(\forall x \exists y : y < x, \forall x \exists y : x < y)$。

(5) 密度性 $(\forall x, y : x < y \rightarrow \exists z : x < z < y)$。

存在多个满足这些公理的非同构模型，最明显的例子是有理数集 $\mathbb{Q}$ 和实数集 $\mathbb{R}$。但在大多数使用真实时间和时钟的场景中，我们并不需要所有这些性质——例如，数字时钟显然不满足密度公理，但仍然在许多情况下非常有用。（当将密度性替换为

(5') 离散性时，整数集 $\mathbb{Z}$ 是时间的标准模型）。

这表明，通过硬件计数器或在计算机程序中使用实数或整数类型的变量来实现时钟确实是“正确”的。然而，偶尔我们会注意到，实际上公理 (4) 和 (5) 并未被完全实现：例如在大型仿真程序中，时钟变量可能会溢出（仿真时间不具备永恒性），或者由于舍入误差导致本不应同时发生的事件被判定为同时发生（仿真时间不具备密度性）。

主要问题是：能否设计一种逻辑时钟系统和用于异步分布式系统的同步机制，使其满足公理 (1) 到 (5)（或 (5')），而无需使用真实时间或物理时钟或其他类似机制？并且能否利用该逻辑时钟系统对事件进行时间戳标记，从而保持因果关系？事实证明，Lamport 的时钟同步机制 [9]（我们将在下一节中简要介绍）满足这些要求，并在许多方面具有实用性。尽管如此，它仍存在一个缺陷，即无法保持因果独立性。正如我们将看到的，通过使用 N^n, Z^n, Q^n 或 R^n（其中 n 表示进程的数量）作为“时间域”，是能够实现因果独立性的。

6 虚拟时间

虚拟时间与真实时间之间的主要区别似乎在于，虚拟时间只能通过事件的相继性来识别（因此是离散的）。虚拟时间不像真实时间那样自行流逝，我们无法逃

避或影响真实时间的流逝。仅仅什么都不做，等待虚拟时间过去是危险的——如果没有任何事件发生，虚拟时间将停滞不前，我们所等待的虚拟时间瞬间可能永远不会到来。分布式系统中虚拟时间的概念由 Lamport 在 1978 年 [9] 提出并广为传播。它被广泛应用于分布式控制算法（尽管并不总是明确指出），例如在互斥算法和并发控制算法中。Morgan 给出了一些虚拟时间的应用 [11]，Raynal 证明了不同进程中逻辑时钟之间的漂移可以被有界 [16]。

一个逻辑时钟 C 是某种抽象机制，它为任意事件 $e \in E$ 分配一个值 $C(e)$（事件 e 的时间戳），该值属于某个“时间域”T，并且满足某些条件。形式上，逻辑时钟是一个函数 $C : E \to \mathrm{T}$，其中 T 是一个偏序集，且满足时钟条件：

$$e < e' \rightarrow C(e) < C(e')$$

成立。在 T 上的非自反关系'<' 称为“早于”，其逆关系称为“晚于”。用语言描述，时钟条件可表述为：如果事件 e 能够因果地影响另一个事件 e'，那么 e 的时间戳早于（即更小）e' 的时间戳。注意，其逆蕴含关系并非必需。

$$e < e' \nrightarrow C(e) < C\,(e')$$

根据时钟条件，以下性质成立 [11]：

(1) 如果某个进程上事件 e 发生在事件 e' 之前，则事件 e 被赋予的逻辑时间早于事件 e' 被赋予的逻辑时间。

(2) 对于任意从一个进程发送到另一个进程的消息，发送事件的逻辑时间始终早于接收事件的逻辑时间。

通常，时间域 T 取整数集 $\mathbb{N}$，逻辑时钟通过一组计数器 C_i 实现，每个进程 P_i 对应一个计数器。为保证时钟条件，局部时钟必须遵循一个简单的协议：

(1) 当在进程 P_i 上执行内部事件或发送事件时，时钟 C_i “递增”：

$$C_i := C_i + d \qquad (d > 0).$$

(2) 每条消息包含一个时间戳，其值等于发送事件的时间。

$$C_i := C_i + d \quad (d > 0).$$

(3) 当在 P_i 上执行接收事件，且接收到一个时间戳为 t 的消息时，时钟被推进：

$$C_i := max(C_i, t) + d \qquad (d > 0).$$

我们假设事件的时间戳已经是本地时钟的新值（即，更新本地时钟发生在执行事件之前）。典型的 d 值为 $d = 1$，但每个“递增”可以有不同的 d 值，d 也可以是上一次“递增”以来真实时间差的近似值。

$$C_i := \max(C_i, t) + d \quad (d > 0).$$

两个事件 e, e' 互相独立（记为 $e \| e'$）当且仅当 $\neg(e < e')\&\neg(e' < e)$。我们能否对互相独立的事件的时间戳做出一些说明？事件？图 10 显示，因果上独立的事件可能获得相同的或不同的时间戳。（这里假设初始时 $C_i = 0$ 且 $d = 1$）。e_{12} 和 e_{22} 具有相同的时间戳，但 e_{11} 和 e_{22} 或 e_{13} 和 e_{22} 具有不同值。

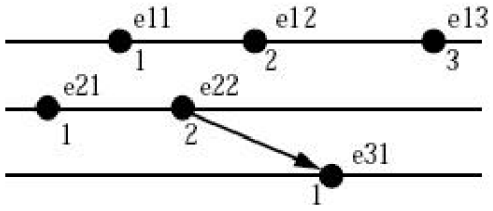

图 10. 时间-

图 11 展示了这种情况。通过观察它们的时间戳，我们可以得出结论：如果两个事件具有相同的时间戳，则它们是独立的。我们还可以得出结论：如果 $C(e) < C(e')$，则 $\neg(e' < e)$，即可以保证过去不会受到未来的影响。然而，如果 $C(e) < C(e')$，则无法判断 $e < e'$ 或 $e \| e'$，即无法判断事件之间是否存在因果关系。（请注意，因果关系 $e < e'$ 并不必然意味着 e 在某种模态上导致 e'，它仅仅意味着 e 可能导致 e'）。这是一个重要缺陷——仅通过观察事件的时间戳，无法断言某个事件不可能影响另一个事件。

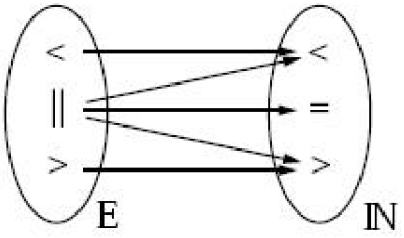

图 11. 结构丢失

该缺陷的原因是，C 是一个保持'$<$' 的序同构，但它通过将 E 映射到线性序而抹去了大量结构——它不保持否定（例如，"$\|$"）。我们所寻求的是一个更适合 T 的域集合，以及一个将 E 映射到 T 的同构。

7 向量时间

假设每个进程都有一个简单的时钟 C_i，每当发生一个事件时，该时钟就递增 1。一个理想化的外部观察者可以立即访问所有本地时钟，因此在任意时刻都知道所有进程的本地时间。存储这种全局时间知识的合适结构是一个向量，每个进程对应一个元素。图 12 所示的例子说明了这一思想。

我们的目标是构建一种机制，使得每个进程都能获得这种全局时间的最优近似值。为此，我们为每个进程 P_i 配备一个时钟 C_i，它是一个长度为 n 的向量，其中 n 是进程总数。（由于时间此后将被视为一组向量，因此时钟——即存储时间的设备——必须通过向量或“数组”来实现）。每当发生一个事件，本地时钟就会“滴答”一次。进程 P_i 通过递增其自身时钟的对应分量来“滴答”：

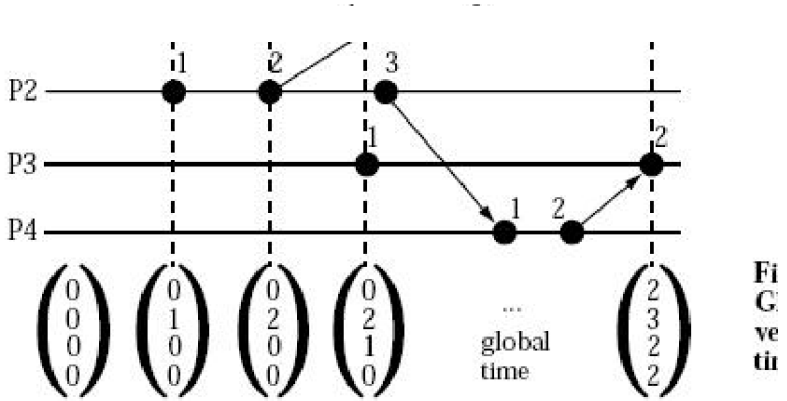

图 12. 全局向量时间

$$C_i[t] := C_i[t] + 1 \square$$

（任何实数值的增量 $d > 0$ 代替 1 都是可以接受的）。如前所述，滴答操作被认为发生在任何其他事件操作之前，事件 e 的时间戳 $C(e)$ 是滴答之后的时钟值。（然而，请注意，现在时间戳是向量）。正如 Lamport 原始方案中一样，每条消息都会附带一个时间戳，该时间戳是本地时钟的向量。当一个进程接收到一个带时间戳的消息时，它便获得了关于其他进程全局时间近似的部分知识。接收方通过以下方式简单地结合自身对全局时间的了解 (C_i) 和接收到的近似值 (t)：

$$C_i := sup(C_i, t)$$

其中 t 是消息的时间戳，sup 是逐分量最大操作，即 $\text{sup}(u, v) = w$，使得对每个分量 i，有 $w[i] = \max(u[i], v[i])$。由于该方案的传递性，一个进程也可能接收到关于非邻居进程时钟的时间更新。图 13 展示了时间传播方案的一个示例。

$$C_i := \text{sup}\,(C_i, t)$$

因果上独立的事件可以按任意顺序发生。此外，在图 13 中，全局时间是通过 $(0,0,0,0)$、$(0,1,0,0)$、$(0,1,0,1)$ 还是通过 $(0,0,0,0)$、$(0,0,0,1)$、$(0,1,0,1)$ 增加，这在某种程度上是任意的——两种序列都是可能的，它们反映了不同但等价的时间图谱中的时间流动。然而，对于图 13 所示的计算，时间 $(1,3,0,0)$ 永远无法被观察到，它是一个“不可能”的时钟值。

请注意，由于只有进程 P_i 能够推进全局时间的第 i 个分量，因此它总是拥有最对其自身局部时间有准确的了解。这得出以下定理：

定理 5 在任意真实时间瞬间，对所有 i, j：$C_i[i] \geq C_j[i]$。

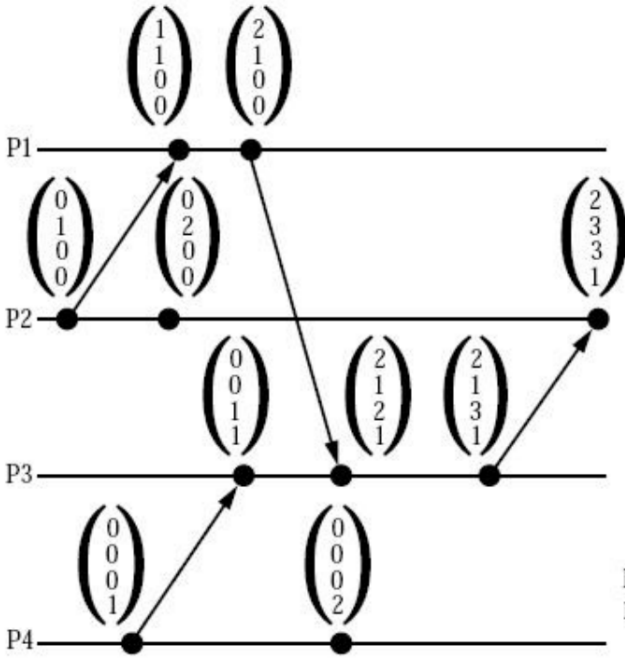

图 13. 时间传播

可以通过定义关系“$\leq$”、“$<$”和“$\|$”来比较时间向量：

定义 4 对于两个时间向量 u, v

$u \leq v$ 当且仅当 $\forall i : u[i] \leq v[i]$。

$u < v$ 当且仅当 $u \leq v \& u \neq v$，且

$u \| v$ 当且仅当 $\neg(u < v) \& \neg(v < u)$。

请注意，“$\leq$”和“$<$”是偏序关系。反射性和对称性的并发关系“$\|$”是标准时间中同时性关系的推广。在标准时间中，“现在”只是过去与未来的无延展交点，而在这里我们拥有一个更广的范围。然而，请注意，并发关系是不可传递的！

以下定义为一个切片赋予时间：

定义 5 设 X 为一个切片，c_i 表示进程 P_i 的切片事件（或若不存在特殊切片事件，则表示属于 X 的 P_i 的最大事件——回忆我们假设每个进程都存在一个初始事件）。则

$$t_X = sup(C(c_1), \ldots, C(c_n))$$

称为切片 X 的全局时间。（请注意，sup 是可结合的）。

$$t_X = \sup\left(C\left(c_1\right), \ldots, C\left(c_n\right)\right)$$

如图 14 所示，不同的切片可以具有相同的时间（例如中的 (2, 1, 2, 0)）。然而，对于一致切片，时间是唯一的，从而为切片提供了一个实用的一致性判据：

定理 6 设 X 和 c_i 如定义 5 所述。X 一致当且仅当 $t_X = (C(c_1)[1], \ldots, C(c_n)[n])$。

证明：

(1) 若 X 一致，则根据定理 4，我们可以假设 $c_1, \ldots, c_n$ 发生在真实时间的同一瞬间。则定理 5 成立，从而得出结论。

(2) 若 X 不一致，则存在某个进程 P_i 在 c_i 之后发送的一条消息，被另一个进程 P_j 在 c_j 之前接收。若 t 表示该消息的时间戳，则 $c_i[i] < t[i] \leq c_j[i]$。因此 $t_X = (C(c_1)[1], \ldots, C(c_n)[n])$。

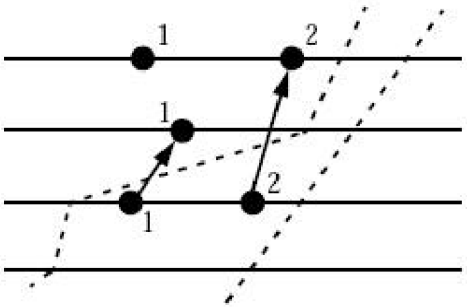

图 14. 切片的时间

8 向量时间的结构

非线性向量时间具有有趣的结构。首先我们有以下定理：

定理 7 对任意 $n > 0$，$(N^n, \leq)$ 是一个格。

证明是显然的。对于任意两个向量 $u, v \in N^n$，最小上界就是 sup(u, v)，最大下界是 inf(u, v)，其中 inf 类似于 sup 定义，只是将 max 替换为 min。（代替 N^n，我们也可以使用 Z^n, Q^n 或 R^n）。如果只考虑事件集 E 的可能时间向量，则情况变得有趣起来。类比于定理 2，可以证明

定理 8 事件集 E 的可能时间向量集合是 $(N^n, \leq)$ 的一个子格。

事实上，可以将一致切片 X 与它的时间向量 t_X “识别”在一起。可能时间向量与一致切片的识别导致以下主要定理：

定理 9 对于事件集 E，一致切片的格与可能时间向量的格是同构的。

这就是我们所寻找的同构！再次，我们把证明留给读者。定理 9 有一个优美且重要的推论：

定理 10$\forall e, e' \in E : e < e'$ 当且仅当 $C(e) < C(e')$，且 $e \| e'$ 当且仅当 $C(e) \| C(e')$。这为我们提供了一种非常简单的方法来判断两个事件 e, e' 是否存在因果关系：我们取它们的时间戳 C(e) 和 C (e′)，并检查是否满足 $C(e) < C(e')$ 或 $C(e') < C(e)$。如果测试成立，则这两个事件存在因果关系；否则它们是因果独立的。如果已知事件发生的进程，则可以简化该测试：

定理 11 若事件 e 发生在进程 P_i 上，则对于任意事件 $e' \neq e$，有 $e < e'$ 当且仅当 $C(e)[i] \leq C(e')[i]$。

从因果关系的图示意义可以看出，定理 10 和定理 11 的正确性是显而易见的。如果事件 e 能够因果影响事件 e'，那么在时间图中必须存在一条因果路径，将事件 e 的（局部）时间信息传播到事件 e'。沿着路径传播时间信息 t 只能使其增大。反之，如果事件 e' "知道"了事件 e 的局部时间，则必须存在从 e 到 e' 的因果路径。(参见 [6] 和 [5])。

偏序图中的任意路径对应于一种可能的交错——一个与因果关系一致的事件线性序列。因此，路径集合决定了全局虚拟时间的可能演化。然而，时间并非总能在所有维度上自由传播，消息对时间的演化施加了限制——当进程 P_i 在 $C_i[i] = s$ 时发送一条消息，并被进程 P_j 在 $C_j[j] = r$ 时接收时，会引入限制 $C[i] < s \rightharpoonup C[j] < r$，等价于 $\neg(C[i] < s \& C[j] \geq r)$。

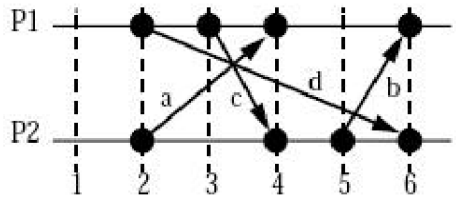

图 15. 包含四条消息的图示

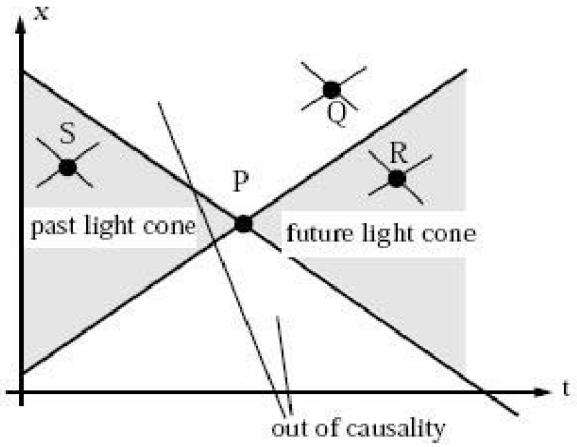

图 17. 光锥图

图 15 所示的时间图包含了四条消息，图 16 展示了格结构以及由四条限制 $\neg(C[2] < 2 \& C[1] \geq 4), \neg(C[2] < 5 \& C[1] \geq 6), \neg(C[1] < 3 \& C[2] \geq 4), \neg(C[1] < 2 \& C[2] \geq 6)$ 截断的区域。(三进程系统的格结构在视觉上可能更美观。)

9 刘维斯时空

标准时间模型满足第 5 节中的公理 (1)-(5)。然而，现实并非"标准"。事实上，由于光速是有限的，刘维斯著名的相对论时空模型可能比标准时间模型更准确地反

映现实。它与我们的时间向量模型有一些良好的类比。

在刘维斯模型中，将 n 维空间与 1 维时间结合，形成一个 $n+1$ 维的世界图景。虽然在物理现实中 $n=3$，我们仅考虑 $n=1$ 的情况，从而得到一个二维结构（即时间图），包含一个时间维度和一个空间维度。由于信号的传播速度是有限的，一个事件只能影响位于所谓未来光锥内部的其他事件，如图 17 所示。(注意这与图 9 的相似性，两者描述的是相同的现象)。事件 P 和 Q 是“因果无关”的，而事件 S 可以影响 P，P 可以影响 R。给定事件 e 和 e' 的两个坐标（以及光速），可以通过简单的算术运算判断这两个事件是否因果无关，或哪一个事件可以影响另一个事件。

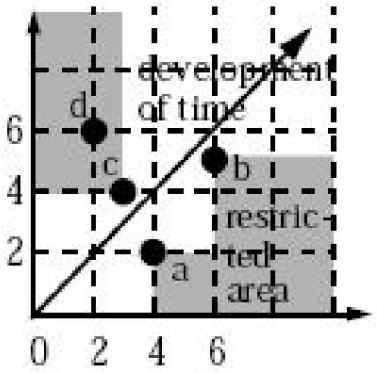

图 16. 限制条件

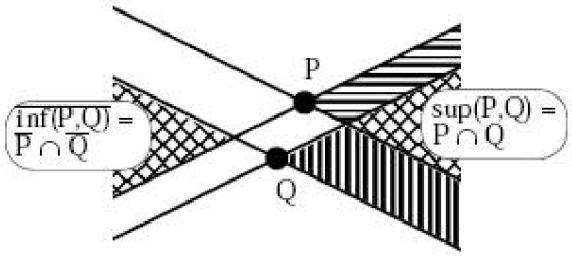

图 18. 光锥的格结构

注意，我们有一个事件的偏序集，每个事件都有一个“时空标记”(即其坐标)。保持因果关系的变换（对应于我们的“弹性带变换”）在刘维斯时空模型中也起着重要作用，这些是著名的洛伦兹变换，它们保持光锥不变。这种类比甚至更进一步——二维闵可夫斯基空间中的光锥形成一个格点。图 18 展示了这一构造。我们简单地将时空点与其光锥对应起来。未来光锥的交集定义了上确界，而过去光锥的交集定义了下确界。(请注意，任意两个光锥都会相交)。这种对应关系类似于我们对切片和时间向量的对应。

乍一看，闵可夫斯基时空与我们的向量时间之间存在着强烈的结构类比，这令人惊讶——它似乎表明向量时间是基于一个基本且非虚构的概念。然而，现实世界是分布式的，这种类比可能并不像表面看起来那样强烈：三维光锥并不构成一个格点！

10 向量时间的应用

向量概念的一个有趣应用是分布式调试领域。为了定位导致错误的 bug，程序员必须思考程序执行过程中事件之间的因果关系。然而，在分布式环境中进行调试比传统调试更困难，主要是因为复杂性增加以及无法重现的竞争条件。因此，事件的追踪非常重要，带时间戳的追踪数据可用于检测可能的竞争条件。显然，如果两个事件之间不存在因果关系，则可能存在竞争条件。这可以通过比较向量的时间戳来检测。时间向量还可以帮助证明某个事件不可能是另一个事件的原因，从而有助于定位错误。（然而请注意，如果 $e < e'$，则只有可能 e 影响 e'）。

我们的向量时间概念已在我们实验多机系统上的一个分布式调试系统中实现并集成 [6]。独立于我们，Fidge[5, 4] 最近提出了并讨论了使用逻辑时钟向量进行分布式调试的方法。与分布式调试类似的一个潜在应用是所谓的分布式系统追踪检查器，它们观察一系列事件，并检查该序列是否符合给定规范的可能事件序列 [7]。

为了进行性能分析，了解潜在的并发性是有用的。对于满足 $e \| e'$ 的两个事件 e, e'，它们可以并发执行。因此，分析一次执行的带时间戳的追踪数据有助于确定并行程度。

一种与我们时间向量非常相似的方案可用于在网络分区背景下检测多个文件副本之间的相互不一致。在 [15] 中，Parker 等人展示了如何使用向量来检测文件副本在不同分区中独立修改的情况。每个文件副本都有一个所谓的版本向量，其第 i 个分量表示在站点 i 处对该文件进行的更新次数。如果两个版本向量 u, v 满足 $u \| v$（且 $u \neq v$），则会触发版本冲突。两个版本向量为 u, v 且不发生冲突的文件副本，无需进一步操作即可被协调，协调后文件的版本向量为 $\sup(u, v)$。

在分布式仿真系统中，每个进程都有自己的逻辑时钟。然而，这些时钟并非完全无关。一个进程只有在保证不会接收到某个其他进程因时间落后而发送的消息时，才能安全地推进其时钟。事实上，情况稍微复杂一些，但关键点在于，为了获得高度的并行性，每个进程必须对其他进程的逻辑时钟有良好的近似值。在这里，向量时间的概念应用得非常自然，将时间向量附载在应用消息上可以提高加速比。目前正正在实现一个采用这一思想的分布式仿真系统。

向量时间的另一个用途是分布式算法的设计。在某种程度上，拥有对全局时间的最佳可能近似，应能简化分布式算法和协议的开发。我们将在下一节中介绍一个这样的应用。

11 在没有 FIFO 通道的系统中计算全局状态

在本节中，我们使用时间向量“重新发明”了 Chandy-Lamport 算法的一个变体，并展示如何在消息不一定按发送顺序接收的情况下计算快照。首先，我们集中于

进程的局部状态，消息在传输中的情况将在后续考虑。

显然，分布式系统的全局状态概念仅对一致切片有意义。定理 6 说明了如何判断给定切片是否一致。然而，我们所寻找的是一个能保证仅产生一致切片的方法。

在现实世界中，快照算法（例如获取一致的人口普查）很简单：

(1) 所有“进程”就某个“未来”时间 s 达成一致。

(2) 进程在时间 s 取其局部快照。

(3) 在时间 s 之后，收集局部快照以构建全局快照。如果我们要将该算法适应于没有公共时钟的分布式系统，那么进程必须就一个未来的虚拟时间 s 或一组相互并发且尚未发生的虚拟时间瞬态 $s_1, \ldots, s_n$ 达成一致。以下定理是很有帮助的。

定理 12 在时钟 C_i 跳动的时刻：$\neg\exists j : C_i < C_j$。

证明（概要）：使用定理 5，并回忆消息传输时间被假定为非零。

这意味着，请求“现在”快照永远不会太晚；在执行本地事件时，其他进程的时钟不可能晚于 P_i 自己的时钟！

为简化论述，我们假设 P_i 是唯一请求快照的发起者。第一个想法如下：

(1) P_i “跳动”一次，然后将其“下一个”时间 $s = C_i + (0, \ldots, 0, 1, 0, \ldots, 0)$ 设为共同快照时间。（“1”位于第 i 位）。

(2) P_i 将 s 广播给所有其他进程。

(3) P_i 在得知所有进程都已知晓 s（例如通过收到确认）之前，不执行任何事件。

(4) 然后 P_i 再次“跳动”（从而将 C_i 设置为 s），取一次本地快照，并向所有进程广播一个虚拟消息。这迫使所有进程将其时钟推进到一个值 $\geq s$。

(5) 每个进程在其本地时钟等于 s 或从某个小于 s 的值跳变到大于 s 的值时，取一次本地快照并将其发送给 P_i。

P_i 在成功发布共同快照时间 s 之前处于“冻结”状态的缺点，可以通过“发明”一个第 $n+1$ 个虚拟进程来克服，该虚拟进程的时钟由 P_i 管理。此时，P_i 可以像其他进程一样自由地使用其自身时钟，从而不再受限。但显然，前 n 个分量的向量并无实际用途——它们完全可以被省略！因为虚拟时钟 C_{n+1} 只在 P_i 启动快照算法的新一轮运行时才会跳动（我们假设这将在前一轮运行完成之后才发生），因此第一次广播阶段也是不必要的——进程们早已知道下一个快照时间！同时，也可以很容易看出，使用模 2 的计数器（即交替的布尔状态指示器）足以替代整数计数器。

如果我们把两种状态称为白色（“快照之前”）和红色（“快照之后”），那么我们最终得到一个简单的算法：每条消息要么是白色，要么是红色（指示该消息是在快照前还是快照后发送的），每个进程（初始为白色）一旦首次接收到红色消息，就会立即变为红色并取一次本地快照。白色进程只发送白色消息，红色进程只发送红色消息。发起进程会自发地变为红色，然后启动一个虚拟广播算法，以确保最终所有进程都变为红色。为实现这一点，它可能直接或间接地向所有进程发送（红色）虚拟消息（例如通过使用虚拟环），或者通过使用一种协议，当进程变为红色

时，向其所有邻居发送虚拟消息来淹没网络。

请注意，即使消息不是按发送顺序被接收，该算法仍然正确。很容易看出，快照算法所引起的切割（由所有“白色事件”组成）是一致的。不存在一个由红色进程发送、被白色进程接收的消息，因为这样的消息会在消息被接受之前立即把接收进程染成红色。

一致切割的完整全局状态不仅包括本地状态，还包括传输中的消息。幸运的是，这些消息很容易被识别——它们是白色的，但被红色进程接收。每当红色进程接收到这样的消息，它会简单地将该消息的副本发送给发起者。

唯一剩下的问题是终止性。发起者会收到所有传输中的消息的副本，但它并不知道何时收到最后一条消息。原则上，这个问题可以通过任何适用于非 FIFO 信道的分布式终止检测算法 [10] 来解决，只需忽略红色消息即可。然而，在这种情况下，缺陷计数方法尤为吸引人，因为可以在与本地状态快照同时进行时，对消息计数器进行一次一致的快照。

为此，每个进程配备一个计数器，作为进程状态的一部分，该计数器记录该进程已发送的消息数减去已接收的消息数。（然而，快照算法的消息不计入）。通过收集并累积这些计数器以及本地快照，发起进程可以知道有多少条白色消息处于传输中，从而确定快照算法的结束。（它知道将收到多少个副本）。因为算法终止后所有进程都变为红色，且没有白色消息处于传输中，因此后续运行可以无需重新初始化，只需简单地交换“白色”和“红色”角色即可。

有趣的是，可以将该算法与 Lai 和 Yang [8] 提出的快照算法进行比较，后者同样不需要 FIFO 信道。在他们的算法中，完整每个节点都保存了发送和接收消息的历史记录。由于这些消息历史被视为局部状态的一部分，因此发起者可以计算出差异，从而确定正在传输中的消息，而无需等待消息副本。尽管该算法“快速”，但它需要的存储空间远大于我们的方案。

最后，应指出，快照算法也可以用作分布式终止检测算法。此时，局部状态不再重要，只有消息计数器是相关的。如果累计的消息计数器等于零，则说明没有消息跨越切割线，因此系统已终止 [10]。

致谢

作者感谢 Michel Raynal 和 Gerard Tel 对本文初稿提出的有益意见。

参考文献

[1] B. Awerbuch. 网络同步的复杂性。Journal of the ACM, 32(4), 1985. 804-823.

[2] G. Bracha 和 S. Toueg. 分布式死锁检测。Distributed Computing, 2, 1987. 127-138.

[3] K. Mani Chandy 和 Leslie Lamport. 分布式快照：确定分布式系统的全局状态。ACM Transactions on Computer Systems, 3(1), 1985 年 2 月。63-75.

[4] C. J. Fidge. 并行调试中的偏序。在 Barton Miller 和 Thomas LeBlanc 编辑的《ACM SIGPLAN 和 SIGOPS 并行与分布式调试研讨会论文集》中，第 183-194 页，美国威斯康星州麦迪逊市威斯康星大学，邮编 53706，1988 年 5 月 5-6 日。

[5] C. J. Fidge. 保留偏序的基于消息传递系统的时标。在第 11 届澳大利亚计算机科学会议论文集中，第 56-66 页，1988 年 2 月。

[6] Dieter Haban 和 Wolfgang Weigel. 分布式系统中的全局事件与全局断点。在第 21 届年度夏威夷国际系统科学会议上，第 166-175 页，1988 年 1 月。

[7] C. Jard 和 O. Drissi. 分布式系统的轨迹检查器构造。IRISA，雷恩大学，法国，技术报告 347，1987 年。

[8] Ten H. Lai 和 Tao H. Yang. 关于分布式快照。Information Processing Letters, 25(3), 1987 年 5 月。153-158。

[9] Leslie Lamport. 时间、时钟和分布式系统中的事件排序。Communications of the ACM, 21(7), 1978 年 7 月。558-565。

[10] F. Mattern. 分布式终止检测算法。Distributed Computing, 2, 1987. 161-175。

[11] Carroll Morgan. 分布式算法中的全局时间与逻辑时间。Information Processing Letters, 20(4), 1985 年 5 月。189-194。

[12] G. Neiger 和 S. Toueg. 在分布式系统中用作真实时间与公共知识的替代。计算机科学系，康奈尔大学，技术报告 86-790，1986 年 11 月。1987 年 6 月修订版。另见发表于第 6 届 ACM 分布式计算原理研讨会论文集，1987 年，第 281-293 页。

[13] M. Nielsen, G. Plotkin, 和 G. Winskel. Petri 网、事件规范与域。第一部分。Theoretical Computer Science, 13, 1981. 85-108。

[14] P. Panangaden 和 K. Taylor. 并发公共知识：异步系统的协议新定义。计算机科学系，康奈尔大学，技术报告 TR 86-802，1986 年。修订版 TR 88-915，1988 年 5 月。另见 ACM 第五届分布式计算原理研讨会论文集，197-209，1988 年。

[15] D. S. Parker, Jr. 等。分布式系统中相互不一致的检测。IEEE Transactions on Software Engineering, 9(3), 1983. 240-247。

[16] Michel Raynal. 一种防止 n 个逻辑时钟之间相互漂移的分布式算法。Information Processing Letters, 24, 1987 年 2 月。第 199-202 页。

[17] J. F. A. K. Van Benthem. 时间的逻辑。D. Reidel 出版社，1983 年。