分布式系统中的时间、时钟和事件顺序

莱斯利·兰波特
马萨诸塞州计算机协会公司

检查了分布式系统中一个事件先于另一个事件发生的概念，并展示了定义事件的偏序。给出了一种用于同步逻辑时钟系统的分布式算法，该系统可用于对事件进行完全排序。
使用解决同步问题的方法说明了总排序的使用。然后该算法专门用于同步物理时钟，并推导出时钟可能不同步多远的界限。

关键词和短语：分布式系统，计算机网络，时钟同步，多进程系统
CR 类别：4.32、5.29

介绍

时间的概念是我们思维方式的基础。它源自事件发生顺序的更基本概念。如果发生在我们的时钟读取 3:15 之后和读取 3:16 之前，我们说某事发生在 3:15。事件时间顺序的概念贯穿了我们对系统的思考。例如，在航空公司预订系统中，我们指定如果在航班满员之前提出预订请求，则应予以批准。但是，我们将看到，在考虑分布式系统中的事件时，必须仔细重新审视这个概念。

这项工作得到了国防部高级研究计划局和罗马航空发展中心的支持。它由罗马航空发展中心根据合同编号 F 30602-76-C-0094 进行监控。

作者地址：SRI 国际计算机科学实验室，333 Ravenswood Ave.，Menlo Park CA 94025。

分布式系统由一系列不同的进程组成，这些进程在空间上分开，并通过交换消息相互通信。互连计算机的网络，例如 ARPA 网络，是一个分布式系统。单台计算机也可以看作是一个分布式系统，其中的中央控制单元、存储单元和输入输出通道是独立的进程。如果与单个进程中的事件之间的时间相比，消息传输延迟不可忽略，则系统是分布式的。

我们将主要关注空间分离的计算机系统。然而，我们的许多评论将更普遍地适用。特别是，单台计算机上的多处理系统涉及的问题与分布式系统的问题类似，因为某些事件可能发生的顺序不可预测。

在分布式系统中，有时不可能说两个事件之一首先发生。因此，“之前发生过”的关系只是系统中事件的部分排序。我们发现问题的出现往往是因为人们没有充分意识到这一事实及其影响。

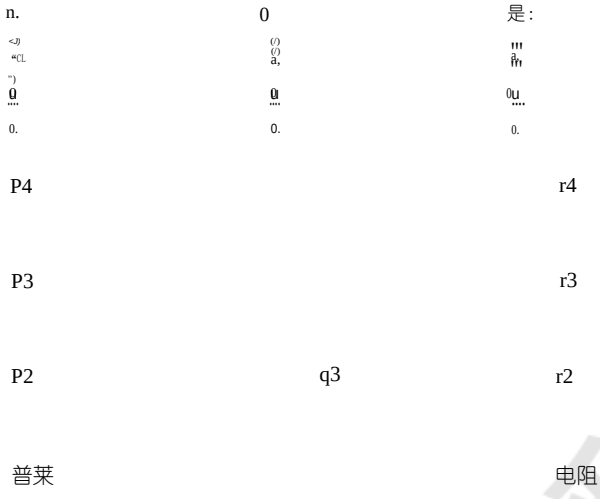
在本文中，我们讨论了由“发生在之前”关系定义的偏序，并给出了一种分布式算法，将其扩展到所有事件的一致全序。该算法可以为实现分布式系统提供有用的机制。我们用一个解决同步问题的简单方法来说明它的使用。如果此算法获得的排序与用户感知的排序不同，则可能会发生意外的异常行为。这可以通过引入真实的物理时钟来避免。我们描述了一种用于同步这些时钟的简单方法，并推导出它们可以偏离同步多远的上限。

部分排序

如果 a 发生的时间早于 b，大多数人可能会说事件 a 发生在事件 b 之前。他们可能会根据时间的物理理论来证明这个定义是正确的。但是，如果系统要正确满足规范，则必须根据系统内可观察到的事件给出规范。如果规范是根据物理时间，那么系统必须包含真实时钟。即使它确实包含真实的时钟，仍然存在这样的时钟不完全准确并且不能保持精确的物理时间的问题。因此，我们将在不使用物理时钟的情况下定义“之前发生过”的关系。

我们首先更精确地定义我们的系统。我们假设系统由一组进程组成。每个过程由一系列事件组成。根据应用程序，计算机上子程序的执行可能是一个事件，或者单个机器指令的执行可能是一个事件

图一。



事件。我们假设一个过程的事件形成一个序列，如果 a 发生在 b 之前，则在这个序列中 a 发生在 b 之前。换句话说，单个进程被定义为一组具有先验总排序的事件。这似乎是流程的一般含义。[1]扩展我们的定义以允许流程拆分为不同的子流程是微不足道的，但我们不会费心这样做。

我们假设发送或接收消息是流程中的一个事件。然后我们可以定义“之前发生过”的关系，用_ .. 表示，如下所示。

定义。系统事件集合上的关系“-”是满足以下三个条件的最小关系：（I）若a和b是同一过程中的事件，且a在b之前，则a-b。（2）如果a是一个进程发送消息，b是另一个进程接收同一消息，则a-b。（3）如果 a - b 和 b - c 则 a - c。如果a ti b 和b ti a ，则称两个不同的事件a 和b 是并发的。

我们假设 a ti a 对于任何事件 a。（事件可以在其自身之前发生的系统似乎在物理上没有意义。）这意味着 - 是系统中所有事件集合的非自反偏序。

用图一这样的“时空图”来看待这个定义是有帮助的。水平方向代表空间，垂直方向代表时间-后期时间高于早期时间。点表示事件，垂直线表示过程，波浪线表示消息。[2]很容易看出 a - b 表示可以从 a 到 b

图 2。

沿着流程和消息线在时间上向前移动来绘制图表。例如，我们在图 I 中有 p1 - r4。

查看定义的另一种方式是说a - b 表示事件 a 有可能对事件 b 产生因果影响。如果两个事件都不能对另一个产生因果影响，则两个事件是并发的。例如，图 1 的事件 p3 和 qa 是并发的。尽管我们已经绘制了图表来暗示 q3 发生在比 p3 更早的物理时间，但进程 P 无法知道进程 Q 在 qa 上做了什么，直到它在 p4 接收到消息。（在事件 P1 之前，p 最多可以知道 Q 计划在 qa 上做什么。）对于熟悉狭义相对论不变时空公式的读者来说，这个定义会显得很自然，例如在 [I] 或[2]的第一章。在相对论中，事件的顺序是根据可以发送的消息定义的。然而，我们采取了更务实的方法，只考虑实际发送的消息。我们应该能够通过只知道那些确实发生的事件来确定系统是否正确执行，而不需要知道可能发生了哪些事件。

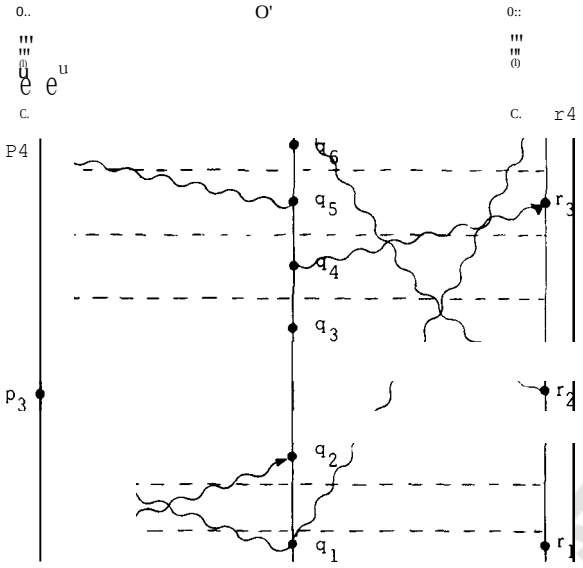
逻辑时钟

我们现在将时钟引入系统。我们从一个抽象的观点开始，其中时钟只是为事件分配数字的一种方式，其中数字被认为是事件发生的时间。更准确地说，我们定义了一个时钟 C；对于每个进程 P；是一个函数，它为该过程中的任何事件 a 分配一个数字 C；(a)。整个码头系统由分配给任何事件的函数 C 表示
b 数字 C(b)，其中 C(b) = CJ(b) 如果 bis 是一个事件
正在处理 P。现在，我们不知道数字 C;(a) 与物理时间的关系，所以我们可以想到时钟 C；作为逻辑时钟而不是物理时钟。它们可以由没有实际计时机制的计数器实现。

’选择什么构成事件会影响过程中事件的顺序。例如，消息的接收可能表示计算机中中断位的设置，或处理该中断的子程序的执行。由于中断不需要按照它们发生的顺序处理，这个选择将影响进程的消息接收事件的顺序。

[2]观察消息可能会被乱序接收。我们允许将多条消息的发送作为单个事件，但为方便起见，我们将假设单个消息的接收与任何其他消息的发送或接收不一致。

无花果。 3.



我们现在考虑这样一个时钟系统是正确的意味着什么。我们不能根据物理时间来定义正确性，因为这需要引入保持物理时间的时钟。我们的定义必须基于事件发生的顺序。最强的合理条件是，如果一个事件 a 发生在另一个事件 b 之前，那么 a 应该比 b 更早发生。我们更正式地陈述这个条件如下。

时钟条件。对于任何事件 a、b:

如果 a - b 则 C(a) < C(b)。

请注意，我们不能期望相反的条件也成立，因为这意味着任何两个并发事件必须同时发生。在图 1 中，p2 和 pa 都与 q:i 并发，所以这意味着它们必须与 q_3同时发生，这与时钟条件相矛盾，因为 p_2 ***-p;i.***

从我们对关系"-"的定义很容易看出，如果以下两个条件成立，则时钟条件成立。

氯。如果 a 和 b 是进程 P; 中的事件，并且 a 在 b 之前出现，则 C;(a) < C;(b)。

C2.如果a是进程P发送的消息; b 是进程 P_J收到该消息，然后 C;(a) < C_1(b)。

让我们根据时空图来考虑时钟。我们想象一个进程的时钟"滴答"通过每个数字，滴答发生在进程的事件。例如，如果a和b是进程P中的连续事件；如果 C;(a) = 4 和 C;(b) = 7，则时钟滴答 5、6 和 7 发生在两个事件之间。

我们在不同进程的所有类似编号的刻度上画一条虚线"刻度线"。图 1 的时空图可能会产生图 2 中的图片。条件 C1 意味着在过程线上的任何两个事件之间必须有一条刻度线，并且条件 C2 意味着每条消息线都必须穿过刻度线。从-的图形含义，很容易看出为什么这两个条件意味着时钟条件。

我们可以将刻度线视为某些笛卡尔坐标系在时空上的时间坐标线。我们可以重新绘制图 2 以拉直这些坐标线，从而获得图 3。图 3 是表示与图 2 相同的事件系统的有效替代方式。没有将物理时间的概念引入系统（这需要引入物理时间）时钟），没有办法决定这些图片中哪一个是更好的表示。

读者可能会发现可视化过程的二维空间网络会产生三维时空图很有帮助。进程和消息仍然用线表示，但刻度线变成了二维表面。

现在让我们假设流程是算法，而事件代表它们执行期间的某些动作。我们将展示如何将时钟引入满足时钟条件的进程。进程P;的时钟由寄存器C;表示，所以C;(a)就是C;包含的值。活动期间C的值；会在事件之间改变，所以改变 C；本身不构成事件。

为了保证时钟系统满足时钟条件，我们将确保它满足条件C1和C2。条件C1简单；进程只需要遵守以下实现规则：

红外线。每个进程P；增量 C;在任何两个连续事件之间。

为了满足条件 C2，我们要求每条消息 m 包含一个时间戳 Tm，它等于消息发送的时间。收到时间戳为 Tm 的消息后，进程必须提前其时钟以晚于 Tm•更准确地说，我们有以下规则。

IR2。 (a) 如果事件 a 是进程 P 发送消息 m;，则消息 m 包含时间戳 Tm=C;(a)。 (b) 收到消息 m 后，进程 P_I设置 C_I大于或等于其当前值且大于 Tm•

在 IR2(b) 中，我们考虑在 C_I设置之后发生的表示收到消息 m 的事件。 (这只是符号上的麻烦，与任何实际实现。) 显然，IR2 确保 C2 得到满足。因此，简单的实现规则 IR I 和 IR2 意味着满足时钟条件，因此它们保证了正确的逻辑时钟系统。

完全排序事件

我们可以使用满足时钟条件的时钟系统对所有系统事件的集合进行总排序。我们只是按时间排序事件

它们发生的地方。为了打破平局，我们使用任何任意的总排序 -< 进程。更准确地说，我们定义一个关系如下：如果 a 是一个正在处理的事件
P，并且 b 是过程 P 中的一个事件；那么 ab 当且仅当 (i) C;(a) < C;(b) 或 (ii) C,(a) = C;(b) and P ，
-<P;。很容易看出这定义了一个全序，并且时钟条件意味着如果 a -,, b 那么 a b。换句话说，关系是
完成“之前发生过”部分排序到总排序的方式。[3]

订购 取决于时钟系统 C;，并且不是唯一的。满足时钟条件的时钟的不同选择产生不同的关系给定任何总排序关系 它扩展 -,,, 有一个时钟系统满足产生这种关系的时钟条件。这只是部分排序
-,, 这是由事件系统唯一确定的。能够对事件进行完全排序对于实现分布式系统非常有用。事实上，实现正确的逻辑时钟系统的原因是为了获得这样的总排序。我们将通过解决以下版本的互斥问题来说明这种事件总排序的使用。考虑一个由共享单个资源的固定进程集合组成的系统。一次只有一个进程可以使用该资源，因此进程必须同步自己以避免冲突。我们希望找到一种将资源授予满足以下三个条件的进程的算法：（I）被授予资源的进程必须先释放它，然后才能将其授予另一个进程。（II）对资源的不同请求必须按照请求的先后顺序进行授权。（III）如果每个被授予资源的进程最终都释放了它，那么每个请求都是最终被授予。

我们假设资源最初只授予一个进程。

这些都是完全自然的要求。它们精确地指定了正确解意味着什么。[4]观察条件如何涉及事件的排序。条件 II 没有说明应该首先批准两个同时发出的请求中的哪一个。

重要的是要意识到这是一个不平凡的问题。除非做出额外的假设，否则使用按接收顺序授予请求的中央调度过程是行不通的。要看到这个，让
po 是调度进程。假设 P1 向 Po 发送请求，然后向 P2 发送消息。收到后一条消息后，P2 向 Po 发送请求。 P2 的请求有可能在 P1 的请求到达之前到达 Po。如果首先批准 P2 的请求，则违反条件 II。

为了解决这个问题，我们实施了一个系统具有规则 IR I 和 IR2 的时钟，并使用它们来定义所有事件的总排序。这提供了所有请求和释放操作的总排序。有了这个顺序，找到一个解决方案就变成了一个简单的练习。它只涉及确保每个流程了解所有其他流程的操作。

为了简化问题，我们做一些假设。它们不是必需的，但引入它们是为了避免分散实施细节。我们首先假设对于任意两个进程 P；和 P:，从 P 发送的消息；最佳；接收的顺序与发送的顺序相同。此外，我们假设每条消息甚至都被最终收到。 （这些假设可以通过引入消息编号和消息确认协议来避免。）我们还假设一个进程可以直接向其他每个进程发送消息。

每个进程都维护自己的请求队列，任何其他进程都不会看到该队列。我们假设请求队列最初包含单个消息 To:Po 请求资源，其中 P_0是最初授予资源的进程，To 小于任何时钟的初始值。

该算法然后由以下五个规则定义。为方便起见，假设每个规则定义的操作形成单个事件。

I. 请求资源，进程P；发送消息 Tm:P;向所有其他进程请求资源，并将该消息放入其请求队列，其中 Tm 是消息的时间戳。

2. 当进程 P;接收消息 Tm:P；请求资源，它将它放在它的请求队列中，并向 P; 发送一个（带时间戳的）确认消息。”

3. 释放资源，进程P；删除任何 Tm:P;从其请求队列中请求资源消息并发送（带时间戳的）P；向所有其他进程释放资源消息。

4. 当进程 P_j收到一个 P 时；释放资源消息，它删除任何 Tm:P；从其请求队列中请求资源消息。

5. 工艺P；当满足以下两个条件时被授予资源：（i）有一个 Tm:P；在其请求队列中请求资源消息，该消息在其队列中的任何其他请求之前按关系排序 -（为了定义消息的关系“”，我们用发送它的事件来标识消息。）（ii）P;已从时间戳晚于 Tm^6的每个其他进程收到一条消息

请注意，规则 5 的条件（i）和（ii）由 P，进行本地测试。

很容易验证这些规则定义的算法是否满足条件 1-111。首先，观察规则 5 的条件（ii），连同消息按顺序接收的假设，保证 P；已了解其当前请求之前的所有请求

〞排序 < 在进程之间建立优先级。如果需要“更公平”的方法，则 < 可以成为时钟的函数
价值。例如，如果 C,(a) = C_i(b) and)< i，那么我们可以让 a= b 如果 j < C; (a) mod N $ i，否则 b = a；其中 N 是进程总数。

[4]“最终”这个词应该准确一些，但这需要从我们的主要话题转移太长时间。

“如果 P1 已经向 P 发送了一条消息，则不需要发送此确认消息；时间戳晚于 T,。

一世;如果 P; < P_j，然后 P；只需要收到带时间戳的消息 2c T，来自 P_i。

要求。由于规则 3 和 4 是唯一从请求队列中删除消息的规则，因此很容易看出我持有的条件。条件 II 遵循以下事实：全序扩展了偏序
--规则2保证在P之后；请求资源，规则 5 的条件 (ii) 将最终成立。规则 3 和规则 4 意味着如果每个被授予资源的进程最终释放它，那么规则 5 的条件 (i) 最终将成立，从而证明条件 III。

这是一种分布式算法。每个进程独立地遵循这些规则，并且没有中央同步进程或中央存储。这种方法可以推广到为这种分布式多进程系统实现任何所需的同步。同步是根据状态机指定的，由一组可能的命令 C、一组 S
可能的状态，以及一个函数 e：ex s-S。关系
e(C, S) = S' 表示执行命令 C
处于状态 S 的机器使机器状态变为 S'。在我们的示例中，集合 C 包含所有命令 P；请求资源和P；释放资源，状态由等待请求命令的队列组成，队列头部的请求是当前授予的请求。执行请求命令将请求添加到队列尾部，执行释放命令则从队列中移除命令
• 他排队。[7]

每个进程使用所有进程发出的命令独立模拟状态机的执行。实现同步是因为所有进程根据它们的时间戳（使用关系）对命令进行排序，因此每个进程使用相同的命令序列。当一个进程获知所有其他进程发出的时间戳小于或等于 T 的所有命令时，它可以执行一个时间戳为 T 的命令。精确的算法很简单，我们不会费心去描述它。

这种方法允许在分布式系统中实现任何所需形式的多进程同步。然而，由此产生的算法需要所有进程的积极参与。一个进程必须知道其他进程发出的所有命令，这样单个进程的故障将导致任何其他进程无法执行状态机命令，从而使系统停止运行。

失败的问题是一个棘手的问题，详细讨论它超出了本文的范围。我们将观察到整个失败的概念仅在物理时间的背景下才有意义。没有物理时间，就无法将失败的过程与只是在事件之间暂停的过程区分开来。用户可以判断系统已经“崩溃”，只是因为他等待响应的时间太长了。在 (3) 中描述了一种尽管单个进程或通信线路出现故障仍能工作的方法。

[7]如果每个进程没有严格交替请求和释放命令，那么执行释放命令可能会从队列中删除零个、一个或多个请求。

异常行为

我们的资源调度算法根据总排序对请求进行排序——这允许以下类型的“异常行为”。考虑一个全国范围的互连计算机系统。假设一个人在计算机 A 上发出请求 A，然后打电话给另一个城市的朋友，让他在另一台计算机 B 上发出请求 B。请求 B 很可能收到较低的时间戳并在请求之前被排序A. 之所以会发生这种情况，是因为系统无法知道 A 实际上在 B 之前，因为该优先级信息是基于系统外部的消息。

让我们更仔细地检查问题的根源。设 Y 是所有系统事件的集合。让我们介绍一组事件，其中包含 ::I 中的事件以及所有其他相关的外部事件，例如
我们示例中的电话。让 - 表示“之前发生过”的关系 for!l 在我们的例子中，我们有 A
- B，但 A# B。很明显，没有算法完全基于 Y 中的事件，并且不会以任何方式将这些事件与 {!_ 中的其他事件相关联，可以保证请求 A 在请求 B 之前被排序。

有两种可能的方法可以避免这种异常行为。第一种方法是将有关订购的必要信息显式引入系统
-· 在我们的示例中，发出请求 A 的人可以从系统接收该请求的时间戳 TA。在发出请求 B 时，他的朋友可以指定给 B 一个迟于 TA·的时间戳。这让用户有责任避免异常行为。

第二种方法是构建一个满足以下条件的时钟系统。

强时钟条件。对于任何事件 a, bin::!:
如果 a - b 那么 C(a} < C(b}.

这比普通的时钟条件强是因为 - 是比 - 更强的关系。我们的逻辑时钟通常不能满足它。

让我们识别ff。在一些“真实”事件中
物理时空，让 - 成为狭义相对论定义的事件的偏序。宇宙的奥秘之一是，有可能构建一个物理时钟系统，该系统完全独立运行，满足强时钟条件。因此，我们可以使用物理时钟来消除异常行为。我们现在将注意力转向这样的时钟。

物理时钟

让我们在时空图中引入一个物理时间坐标，让 C;(t) 表示时钟 C 的读数；在物理时间 t.[8]用于数学计算

“我们将假设一个牛顿时空。如果时钟的相对运动或引力效应不可忽略，那么 C;(I) 必须通过从适当的时间转换到任意选择的时间坐标来从实际时钟读数中推导出来。

为方便起见，我们假设时钟是连续运行的，而不是在离散的“滴答声”中运行。 （离散时钟可以被认为是存在误差的连续时钟
在阅读时最多“打勾”。）更准确地说，我们
假设 C;(t) 是 t 的一个连续的、可微的函数，除了时钟被重置的孤立跳跃不连续性。然后 dC;(t)/dt 表示时钟在时间 t 运行的速率。

为了时钟C;要成为真正的物理时钟，它必须以近似正确的速率运行。也就是说，对于所有的 t，我们必须有 dC;(t)/dt:::::: 1。更准确地说，我们将假设满足以下条件：

PCI。存在一个常数 K « I
使得对于所有 i: IdC;(t)/ dt - 1I < K。

对于典型的晶体控制时钟，K :s 10-u。

时钟单独以近似正确的速率运行是不够的。它们必须同步，以便 C;(t)::::::: CJ(t) 对于所有 i、j 和 t。更准确地说，必须有一个足够小的常数 E 才能满足以下条件：

电脑2。对于所有 i,j: IC;(t) - CJ(t)I < E。

如果我们考虑图 2 中的垂直距离来表示物理时间，那么 PC2 表示单个刻度线的高度变化小于 E。

由于两个不同的时钟永远不会以完全相同的速率运行，因此它们会越来越远离。因此，我们必须设计一种算法来确保 PC2 始终成立。然而，首先让我们检查 K 和 E 必须有多小才能防止异常行为。我们必须确保相关物理事件系统满足强时钟条件。我们假设我们的时钟满足普通时钟条件，所以
我们只需要当 a 和 b 是 :f 中的事件时强时钟条件成立。与 b。因此，我们只需要考虑发生在不同进程中的事件。

令 μ 为一个数，如果事件 a 发生在物理时间 t 并且另一个过程中的事件 b 满足 a - b，则 b 发生的时间晚于物理时间 t + μ。换句话说，*μ* 小于进程间消息的最短传输时间。我们总是可以选择 *μ* 等于
到进程之间的最短距离除以光速。然而，根据消息 in!fare 的传输方式，*μ* 可能会大得多。

为避免异常行为，我们必须确保对于任何 i、j 和 t: C;(t + μ) - CJ(t) > 0。将其与 PC I 和 2 结合使我们能够将所需的 K 的小和 E 到 μ 的值如下。我们
假设当一个时钟被重置时，它总是向前设置，永远不会后退。 （将其回退可能会导致 CI 被违反。）然后 PC I 暗示 C;(t + μ) - C;(t) > (I
- K)μ。使用 PC2，很容易推导出 C;(t + μ) - CJ(t) > 0 如果以下不等式成立：

E/(1 - *K*):5 *μ*.

这种不等式连同 PC1 和 PC2 意味着异常行为是不可能的。

我们现在描述我们用于确保 PC2 成立的算法。令 m 是在物理时间 t 发送并在时间 t’ 接收的消息。我们定义 Pm = t’ - t 为
消息的总延迟 m。当然，接收 m 的进程不会知道这个延迟。然而，我们假设接收过程知道一些最小延迟 μm :::: 0 使得 μm :s P,。我们称 lm = Pm
- *μ，消息不可预测的延迟。*

我们现在将规则 IR I 和 2 专门用于我们的物理时钟，如下所示：

红外！’。对于每个 i，如果 P；在物理时间 t 没有收到消息，则 C1 在 t 和 dC 处可微；(t)/dt >0.

IR2’。 (a) 如果 P；在物理时间 t 发送消息 m，然后 m 包含时间戳 Tm = C,(t)。 (b) 在时间 t’ 收到消息 m 后，进程 Pi 设置 CJ(t’)
等于最大值 (CJ(t’ - 0), Tm + μ,)。[9]

虽然规则是根据物理时间参数正式指定的，但进程只需要知道自己的时钟读数和收到的消息的时间戳。为了数学上的方便，我们假设每个事件发生在物理时间的精确时刻，并且同一过程中的不同事件发生在不同的时间。这些规则是规则 IR1 和 IR2 的特化，因此我们的时钟系统满足时钟条件。真实事件具有有限持续时间这一事实不会导致算法的实现困难。实现中唯一真正关心的是确保离散时钟滴答足够频繁，以便维持 C1。

我们现在表明该时钟同步算法可用于满足条件 PC2。我们假设过程系统是由一个有向图描述的，其中有一个来自过程 P 的弧；处理 Pi 代表一条通信线路，通过该线路直接从 P 发送消息；到皮。我们说如果对于任何 t，P，每隔 -r 秒通过这条弧发送一条消息；发送至少一个
在物理时间 t 和 t + -r 之间向 Pi 发送消息。这有向图的直径是最小的数 d，使得对于任何一对不同的过程 Pi，Pk，从 Pi 到 Pk 的路径最多有 d 条弧。

除了建立 PC2 之外，以下定理还限制了系统首次启动时时钟同步所需的时间长度。

定理。假设有一个直径为 d 的过程的强连接图，它始终遵守规则 IR I’ 和 IR2’。假设对于任何消息 m，对于某些常数 μ，*μ*m :S μ，并且对于所有 t 2: 到： (a) PCI 成立。
(b) 存在常数 -r 和 f，使得每 -r 秒在每个弧上发送一个具有小于 f 的不可预测延迟的消息。然后 PC2 满足 E ::::: d(2K-r + f) 对于所有 t :::: 到 + -rd，其中近似假设
μ + f « *τ.*

这个定理的证明非常困难，在附录中给出。关于同步物理时钟的问题已经做了大量的工作。我们建议读者参考 [4] 的介绍

″ C (I’ - 0) = 1i̲m C (t’ - J8J)。
Jc'i-o

引出主题。文献中描述的方法可用于估计消息延迟 /J,m 和用于调整时钟频率 dC;/dt（用于允许此类调整的时钟）。然而，时钟永远不会倒退的要求似乎

将我们的情况与以前研究的情况区分开来，我们相信这个定理是一个新的结果。

结论

我们已经看到，“发生在之前”的概念定义了分布式多进程系统中事件的不变偏序。我们描述了一种将偏序扩展到某种任意全序的算法，并展示了如何使用这种全序或dering 来解决简单的同步问题。未来的一篇论文将展示如何扩展这种方法来解决任何同步问题。算法定义的总排序是任意的。如果它不同意系统用户感知的顺序，它就会产生异常行为。这可以通过使用正确同步的物理时钟来防止。我们的定理表明时钟可以紧密同步。

在分布式系统中，重要的是要意识到事件发生的顺序只是偏序。我们相信这个想法对于理解任何

多进程系统。它应该有助于理解
独立于用于解决它们的机制的多处理的基本问题。

附录

定理的证明

对于任何 i 和 t，让我们将 C/ 定义为设置为等于 C 的时钟；在时间 t 并以与 C 相同的速率运行；但永远不会重置。换句话说，

$$C/(t') = C_i(t) + \int_t^{t'} [dC_i(t)/dt]dt \qquad (1)$$

对于所有 t’ 2:: t。注意

C; (t’) 2: : C/ (t’) 对于 (2)
所有 t’ 2: : t。

假设进程 P1 在时间 t 向进程 P2 发送消息，该消息在时间 t₂ 接收，但具有不可预测的延迟 g，其中 tos 是 t2。那么对于所有 t 2: t2
我们有：

C (t) 2:: C (t2) + (1 - K)(t - tz) [由 (1) 和 PCl]
2:: C1(t1) + /J,m + (1 - K)(t - t2) [通过 IR2’ (b)]
= C1(t1) + (1 - K)(t - ti) - [(/2 - ti) - /J,m] + K(t2 - ti)
2: : C1 (ti) + (1 - K) (t - 星期二) - f

因此，根据这些假设，对于所有 t 2: /2 我们有：

C (t) 2:: C1(ti) + (1 - K)(t - ti) - f (3)

现在假设对于 i = 1, ... , n 我们有 t; tI < t;+i，到 s ti，并且在时间 tI 处理 P;向进程 P;+1 发送消息，该消息在时间 t;+1 接收到的不可预测延迟小于 f 然后重复应用不等式 (3) 产生以下结果

t 2: ln+l •

C n.;I(t) 2:: C1(t1') + (1 - K)(t - ti') - nf (4)

从PC1、IR1’和2’我们推导出C1(t1’)2::C1(t1)+(1-K)(ti’-ti)。

将此与 (4) 结合并使用 (2)，我们得到

中国+1 (吨) 2: : C1 (1- K) (t - ti) - ng (5)

对于 t 2: tn+l •

对于任意两个进程 P 和 P’，我们可以找到进程序列 P = Po, Pi, ... , Pn+1 = P’, n S d，每个 P 都有通信弧；到 P;+1。通过假设 (b) 我们可以找到时间 t;, tI 与 tI - t;和 t;+1 - tI s v，其中 v = μ, + f 因此，当 t 2 时，形式 (5) 的不等式对 n s d 成立：t1 + d(r + v)。对于任何 i,j 和任何 t, ti 与 ti 2:: to 和 t 2: /1 + d(r + v) 因此我们有：

$$C_i(t) \geq C_j(t_1) + (1 - \kappa)(t - t_1) - d\xi. \qquad (6)$$

现在让 m 是任何带有时间戳 Tm 的消息，并假设它在时间 t 发送并在时间 t’ 接收。我们假设 m 有一个时钟 Cm，它以常数运行

率使得 Cm(t) = tm 和 Cm(t’) = tm + /J,m- 然后
/J,m t’ - t 表示 dCm/dst 1. 规则 IR2’ (b)
将 CJ(t’) 设置为最大值 (C;(t’ - 0), Cm(t’))。因此，只有通过将时钟设置为与其他时钟相等才能重置时钟。

对于任何时间 tx 2: to + μ/(1 - K)，令 Cx 为在时间 Ix 具有最大值的时钟。由于所有时钟运行以小于 1 + K 的速率，我们有所有 i 和所有 t 2: tx:

$$C_i(t) \leq C_x(t_x) + (1 + \kappa)(t - t_x) \qquad (7)$$

我们现在考虑以下两种情况： (i) Cx 是进程 Pq 的时钟 Cq。(ii) Cx 是进程 Pq 在时间 ti 发送的消息的时钟 Cm。在情况 (i)， (7)
简单地变成

$$C_i(t) \leq C_q(t_x) + (1 + \kappa)(t - t_x) \qquad (8i)$$

在情况 (ii) 中，由于 Cm(ti) = Cq(t1) 和 dCm/ dt S 1，我们有

Cx(tx)S Cq(ti) + (tx - ti)。

因此， (7) 产生

C;(t) s Cq(ti) + (1 + K)(t - ti)。 (8ii)

由于 tx 2: to+ μ/(1 - K)，我们得到

Cq(tx - μ/(1 - K))s Cq(tx) - /J, [由 PC1]
S 厘米 (tx) - /J, [由m选择]
S 厘米 (1x) - (九 - 11) /J, [/J,毫秒/J, Ix - 11S Pm]
米/下午 [根据厘米的定义]
=公吨 [通过 IR2’(a)]。
= Cq(ti)

因此，Cq(tx - μ/(1 - K))S Cq(ti)，所以 tx - ti S μ/(1 - K) 从而 ti 2:: to。

让 t_1= Ix 在情况 (i) 中，我们可以结合 (8i) 和 (8ii) 推导出对于任何 t, Ix 与 t ::::: Ix ::::: to + μ,/ (I - K) 有一个过程 Pq 和时间 ti，其中 Ix - μ,/ (l - K)5 115 Ix 使得对于所有 i:

C;(t)5 Cq(ti) + (l + K)(t - ti)。 (9)

用 t::::: Ix+ d(-r + v) 选择 t 和 Ix，我们可以结合

(6) 和 (9) 得出结论，存在一个 ti 和一个过程 Pq 使得对于所有 i:

Cq (ti) - (I - K) (t- ti) - dt5 C; (t)

5 立方体 + (升 + 千) (吨 - 11)

令 t =Ix+ d(-r + v)，我们得到

d(-r + v)5 t - ti5 d(-r + v) + μ,/(1 - K)。

将此与 (10) 相结合，我们得到

Cq(t1) + *(t* - t1) - *Kd(T* + v) - *dg*5 C;(t)5 Cq(t1) (l l)

+ *(t* - !1) + *K[d(T* + v) + μ,/(I - K)]

使用 K « l 和 μ5 v « T 的假设，我们可以将 (11) 重写为以下近似不等式。

Cq(t1) + *(t* - ti) - *d(KT* + t) :5 C;(t) (12)

:5 Cq(ti) + (t - 11) + dKT。

由于这对所有 i 成立，我们得

到 IC;(t) - CJ(t)I :s d(2KT

+ t),

这适用于所有 t :=::: 到 + dT。 □

请注意，证明的关系式 (11) 产生了 I C;(t) - CJ(t) I 的精确上限，因为假设 μ, + g « T 是无效的。对证明的检查提出了一种简单的方法来快速初始化
时钟，或者如果它们因任何原因不同步，则重新同步它们。每个进程发送一条消息，该消息被中继到每个其他进程。该过程可以由任何进程启动，并且需要少于
2d(μ + t) 秒来实现同步，假设每个消息都有一个不可预测的延迟
比 f

确认。使用时间戳对操作进行排序，以及异常行为的概念是由 Paul Johnson 和 Robert Thomas 提出的。

1976 年 3 月收到； 1977 年 10 月修订

参考文献

1. 施瓦茨，J.T.插图中的相对论。纽约美国出版社，纽约，1962 年。

2. Taylor, E.F. 和 Wheeler, J.A.时空物理学，W.H.弗里曼，旧金山，1966 年。

3. Lamport, L. 可靠分布式多进程系统的实现。出现在计算机网络中。

4. Ellingson, C. 和 Kulpinski, R.J.系统时间的传播。*IEEE 翻译通讯。 Com-23, 5 (1973 年 5 月)，605-624。*

浅绑定 Lisp 1.5

小亨利·G·贝克
麻省理工学院

浅绑定是一种允许在有限的计算量中访问变量值的方案。提出了 Lisp 1.5 中浅层绑定的优雅模型，其中上下文切换是一种称为重新定位的环境树转换。Rerooting 是完全通用和可逆的，并且是可选的，因为 Lisp 1.5 解释器将
无论是否重新植根都可以正确操作
在每次上下文更改时调用。由于重根使 assoc (v, a) 保持不变，对于所有变量 v 和所有环境 a，程序员可以访问 a
重根原语，浅 ()，这使他可以动态控制访问是浅的还是深的，以及

这仅影响程序的执行速度，而不影响其语义。此外，多个进程可以在同一个环境结构中处于活动状态，只要reroot 是一个不可分割的操作。最后，rerooting 的概念将 Lisp 中浅绑定的概念与 Dijkstra 对 Algol 的显示相结合，因此是浅绑定的通用模型。

关键词和短语：Lisp 1.5、环境树、FUNARG、浅绑定、深度绑定、多道程序设计、Algol 显示

CR 类别：4.13、4.22、4.32

这项研究得到了高级研究项目的支持
国防部机构，并由海军研究办公室根据合同编号 N00014-75-C-0522 进行监控。

作者现在的地址：罗彻斯特大学计算机科学系，纽约州罗彻斯特市 14627。