分布式系统学习——Lamport时钟经典论文Paper翻译

2020-03-15 09:49:43

阅读笔记参考：[小强zju的读书笔记](https://www.jianshu.com/p/0c79d650d13f)

[分布式系统基础－Lamport Clock](https://www.jianshu.com/p/0d49b1811203)

[Lamport本人评述](https://www.microsoft.com/en-us/research/publication/time-clocks-ordering-events-distributed-system/)

# 分布式系统学习——Lamport时钟经典论文Paper翻译经典论文Paper翻译

## 摘要部分

本文研究了分布式系统中一个事件先于另一个事件发生的概念，并展示了如何定义事件的偏序，同时给出了一种可以对时间完全排序的分布式时钟同步算法。通过一种解决同步问题的方法来描述对全体事件的同步。

然后介绍专门用于同步物理时钟的情况，并推导出时钟不同步的界限。

## Introduction

时间概念是我们思维方式的基础，更基本的源于事件发生顺序，其贯穿我们对系统（system）的思考。然而在分布式系统中我们应**重新审视**这一基本概念。

A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process.

**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，那么该系统就是分布式的。** （无论是在单机系统还是多机系统上，上述情况都要考虑）

本文重点关注空间分离的系统，但是单机多处理器系统涉及一些相类似的问题，因为某些事件可能以不可预测的顺序发生。“之前发生”只是系统中事件的部分排序。（事实上整体事件呈现一个偏序集合 No happen before）。

## The Partial Ordering

We will therefore define the “happened before” relation without using physical clocks.

我们将不依赖物理时钟定义“happen before”规则。

因应用的不同，我们可以将一个子程序（或是一条机器指令的执行）当作某个process里的一个event。

我们假定某个process的所有events构成一个序列，这个序列是线序（或许应该说不包含相等的情况），用文中的话说是具有**先验总排序**。我们根据process中event的发生顺序定义“happen before”

定义“happen before”关系，用（→表示）

1. 如果a，b是相同进程的两个事件，且a出现在b之前，那么。
2. 如果a是某进程消息发送的一方，b是另一进程接收该消息的一方，则。
3. 该关系具备传递性，即 ，，则。如果，，那我们称a，b是并发（concurrent）的。
4. 另外补充，该关系是反自反的，即，对任意a，，（系统中一个时间可以在它自身之前发生没有实际意义）。

在图中即表示可以从a通过波浪线（消息传播途径），或者纵向的进程轴线（事件发生途径）走到b。

另一种方式解读该图像，我们也可以说，并发的进程是不会不经意间相互影响的。

Happened Before的理论和狭义相对论中的物理世界中的观念十分的类似，即event的先后是相对的，在实际世界中的不同惯性坐标系下，2个event的先后关系可能发生变化。不过Paper中没有深入的对比，我会在感想部分中深入讨论下

## Logical Clocks

现在考虑引入时钟，先以数学的方式为每个事件分配一个数字来代表其事件发生的时间。以时钟函数表示进程，其中表示该进程a事件的值。 函数表示整个系统的时钟，如果b是进程的事件之一，则。

由于当前没有将函数值和物理事件关联起来，所以我们将其认为是逻辑时间。

考虑该逻辑时钟的合理性，我们无法根据物理时间验证（这会要求基于物理时间的时钟引入），考虑一个强合理性的条件，如果事件a发生在另一个事件b之前，则a应该在比b更早的时间发生。 我们更正式地陈述这种情况如下。

Clock Condition

For any events a, b: if then

其逆命题不成立（否则会要求所有并发事件同时发生，易证否），即只具备充分性。

满足下面两条件时，时钟条件（Clock Condition）是满足的。

**C1.** 如果a和b是同一进程的两个事件且a发生在b之前，则。

**C2.** 如果a是i进程消息发送方，而b是j进程中该消息的接收方，则。

Condition）。

通过算法来为上述构建的时钟函数赋值，为满足C1和C2，分别设定两条规则：

**IR1.** 同一个process的相邻event，其时钟值是递增的。

**IR2.** i进程a活动发送的消息会携带一个时间戳，则接收方b进程活动b设定其时钟值为。

两条分别对应两条规则，所以时钟条件也得到满足，保证了具备逻辑时钟的合理系统。

## Ordering the Events Totally

在规定进程间一个顺序后，通过以下规则，我们从事件的偏序构建全序，我们便可得到新的“全序关系”（用）来表示：

1. 若对于a，b两事件，有，则
2. 若 且 ，则

可以说“”是将“happen before”偏序关系“”拓展成的全序关系。

该全序依赖与系统时钟的确定，是不唯一的，相反，唯一确定的是事件发生的偏序关系。

对事件进行全排序的有用性将通过解决下列情况下的互斥问题来说明，考虑一个共享单个资源的固定过程集合的系统：

Q1. 已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。 Q2. 必须按照它们的顺序授予对资源的不同请求。 Q3. 如果授予资源的每个进程最终都会释放该资源，那么每一个请求都后被授权。

通过一个中心化的进程通过请求抵达的时刻来安排资源是不可行的（后抵达该进程的消息可能先需求了资源）。

为了解决该问题，我们实行一个满足了IR1和IR2的系统，并且通过其定义出所有事件的全序，具备全局序后，问题就变得相当简单了，只需要确保每个进程清楚其他进程得操作即可。

### 分布式算法细节

每个进程维护自己的**请求队列**，规定请求队列初始包含单一的消息：请求资源，其中是初始占据资源的进程（根据假设，所有资源初始归同一进程占据），且小于任何时钟的初始值。

该算法由以下五条规则定义。为了方便起见，假定每个规则定义的操作构成一个单一事件。

1. 为了请求资源，请求方进程将消息“：请求资源”递送至所有其他进程，并把该消息加入自己的请求队列。表示该消息的时间戳。
2. 当收到消息“：请求资源”，将其加入自己的请求队列并向发送一个带时间戳的确认消息。
3. 为了释放资源，进程将所有“：请求资源”消息移出队列，并把一个带时间戳的释放消息递送给所有其他进程。
4. 当收到消息“释放资源”，将所有“：请求资源”消息移出队列。
5. 当满足下列两条件时，进程占据该资源：
   1. 存在一个“：请求资源”消息在其请求队列中并且序优先于其他的所有请求。
   2. 收到了所有其他进程发来的时间戳晚于的消息。

易于说明该算法满足该系统的三个要求。

这是一个分布式算法，每个进程独立遵循这些规则，没有集中的同步进程或是中央存储。

根据算法1->5构建可以证明Q1，Q2，Q3是满足要求的。

对于Q1，反证法：

假设Q1不成立，则意味着在资源分配给某进程后，假设该进程为，存在至少一个进程，假设为，未释放该资源。但是根据3、4点，这意味着，没有发出“释放”的消息，也就应导致着进程维护的队列中，仍存在“：请求资源”这一消息并未移出（第四点），且是最早的。现在我们审查第五点，上述未删除的消息存在队列，便不会使“：请求资源”这条满足最先（5.1点），那维护队列中不满足其占有资源的条件，矛盾。

对于Q2，时间戳是一个全序，由于5.2点要求收到所有其他进程法来的更晚的消息，所以可以保证先前消息是完全的，可以确定是最早的，根据全序顺序，可保证。

对于Q3，反证法：

假设在均释放的情况下，存在至少一个请求没有被授权（批准），不妨设最早的是的请求，则在其自身队列以及所有其他进程队列中都应在某一时间后都存在着这样一条“：请求资源”，由于始终未被授权，假设5.1不被满足，则应存在“：请求资源”始终存在，且但是我们设定是最早的不被授权的，所以前者必定在某个时间释放，请求资源的记录会随释放消息的到达被撤，不可能始终存在，不成立。若5.2始终不能被满足，在5.1成立后，由于没有fail的机器，必然会收到确认消息，不成立。所以5.1，5.2总会在一定时间后被满足，矛盾。

## Anomalous Behavior

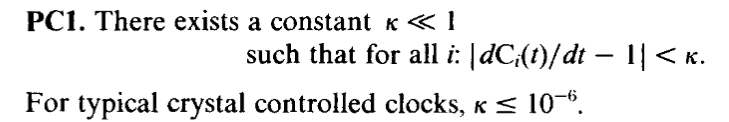
异常情况，上面定义的Logical Clocks有一些反常的行为，例如我们定义的Total Ordering是，但事实上a事件的操作员执行之后通过电话告诉b事件的操作员开始执行，那么Logical Clocks给出的全序就明显违背了客观情况。

这种情况的出现是由于系统不知道“电联”这一要求，因为这种先后顺序是在系统外定义的。 为避免这种反常，有2中解法：

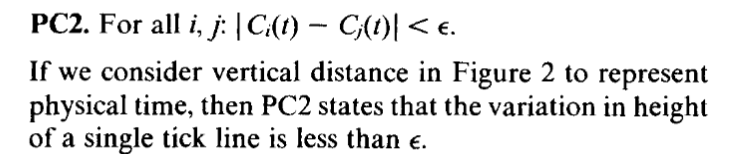
1. 将外部的happened before关系手动的引入到系统内（event b产生是强调依赖 event a）
2. 引入实际物理时钟

## Physical Clocks

对物理始终两条限制，



OS\_1\_1



OS\_1\_2

实际上时限制增速尽可能贴近1，以及各时钟的同步性。

后面保证物理时钟一致性以及满足C1，C2的算法暂时没有看懂，所以后面再补吧，数学功底差一些。