**Time, Clocks, and the Ordering of Events in a Distributed System**

**研读报告**

**1953871 邓泉**

1. **论文核心思想小结**

本文探讨了分布式系统中事件和时间的关系，阐述了通过使用算法在分布式系统中实现任意状态机来实现任何分布式系统的想法。Lamport通过引入逻辑时钟，提出了"happened before"的概念，给出了一个将偏序扩展到整个系统中的所有事件的全序的分布式算法。

针对分布式的时钟同步问题，文章审视了分布式系统中，某事件发生在另一事件之前这一概念，并展示了如何用它来定义事件间的偏序关系(partial order) "→"，引入逻辑时钟来描述事件发生的时间，这样就分离了事件的发生顺序关系与物理时钟。作者提出Clock Condition以及确保系统逻辑时钟满足该条件的Implementation Rule，继而给出了一个可以对具有逻辑时钟的系统进行同步的算法，通过逻辑时钟可以得到事件的全序关系(total ordering)。

该算法描述了一种解决特定共享资源问题的分布式协调方案：每个进程各自均有一个队列Request Queue记录资源申请的消息；当申请资源时在自己的队列中加入资源请求消息，并向其他进程广播；收到其他进程申请时，同样加入队列尾部，并返回带时间戳的ACK消息；释放资源的过程与请求时类似，唯一区别在于前者中进程获取到资源是最后结果，后者中释放出资源是起始事件（原因）。

该定义事件全序关系的算法可以用来实现任意的分布式系统。一个分布式系统可以描述为一个特殊的具有多个由网络互联的处理器的串行状态机。如果能够对输入请求进行全排序，就能够实现任何由网络互联的处理器组成的状态机，因此也就可以实现任意的分布式系统。

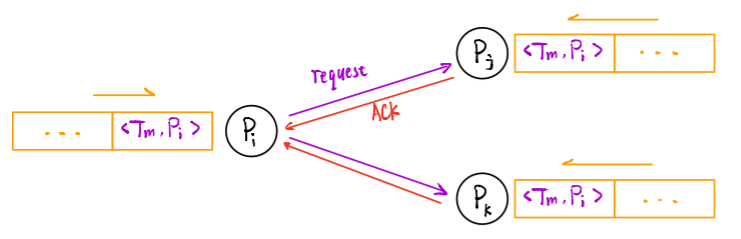
对于Anomalous Behavior异常事件，作者提出了两种解决方案，第一种方式是将关于->顺序的必要信息显式地引入到系统中；二是引入Physical Clocks物理时间，构造一个时钟系统Strong Clock Condition，同时可以推导出时钟可能达到的不同步的一个误差范围。

1. **论文缺陷不足及改进方法**

**缺陷1：**

作者对于一个同步问题案例给出了将该偏序关系扩展为某种全序关系的算法，本人认为在该算法的描述中存在易使读者误解的地方。

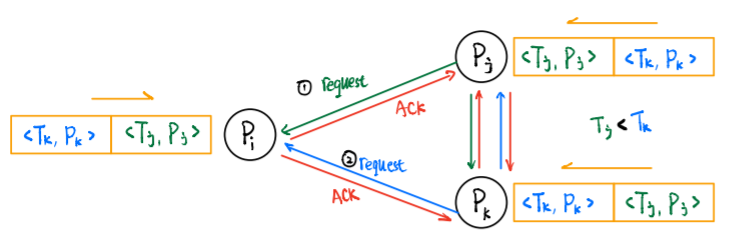
按照原文算法（由5个规则定义）叙述，通常有以下所示的情况一：



情况一：进程Pi发送获取资源的请求

起初进程Pi发送获取资源的请求，首先在自己的请求队列中加入资源请求消息<Tm,Pi>，并将其发送给其他所有进程（规则一）；进程Pj、Pk收到消息<Tm,Pi>后将其放入各自的请求队列，并返回一个带时间戳的ACK给Pi（规则二）；此时对于Pi，毫无疑问<Tm,Pi>位于队列最前面，且已从所有其他进程收到了时间戳>Tm的ACK消息,则可以正式获取资源了（规则五）；使用完资源后依据规则三、规则四释放资源。

问题出现在规则五上，见以下所示的情况二：



情况二：进程Pj、Pk都发送获取资源的请求

起初进程Pj、Pk相继发送获取资源的请求Pj发送的请求未被其他进程收到时，Pk也发出一条请求，Tj<Tk），这样Pj和Pk的请求队列中分别有了<Tj,Pj>和<Tk,Pk>。当<Tj,Pj>和<Tk,Pk>都到达了所有其他进程，不妨假设<Tj,Pj>率先到达，那么可以看到：Pj的请求队列中<Tj,Pj>排<Tk,Pk>前面，而Pk的请求队列<Tk,Pk>排<Tj,Pj>前面。

作者所指的请求队列应该就是典型IFIO的普通队列（后文“Executing a request command adds the request to the **tail** of the queue”更让我相信这一点），那么在上述情况下，根据规则五进程Pj、Pk都会正式获取资源，而该案例中共享资源仅有唯一一个，这就导致了冲突问题。

**改进方法1：**

改进对” request queue”的理解，请求队列应该是一种Priority Queue优先级队列，这里对优先权的定义为：时间戳小的消息，其优先级高。例如，在上述情况二中，尽管<Tj,Pj>、<Tk,Pk>进入Pj、Pk的请求队列顺序不一致，但Tj < Tk决定了<Tj,Pj>一定排在<Tk,Pk>前面，如此一来只有Pj获取资源，运行正常。可以想见，这样的改进也同样适用于该案例中所有其他情形。

或采用向量钟Vector Clock解决“并行发生”的关系。

**缺陷2：**

论文中提出的的算法实现分布式系统中任何形式的进程同步需求，但是该算法的前提是所有进程都参与，同时进程要知道其他进程产生的命令动作，因此只要有一个进程故障或失效，都会导致其他进程无法执行状态机命令带来系统停滞。这便是错误处理问题。

**改进方法2：**

实际上，作者Lamport. L.在他的另一篇论文“The implementation of reliable distributed multiprocess systems”中，论述了如何实现一个可靠的分布式多进程系统，其中对错误处理问题进行了细节性的讨论，给出了一种容错算法。它是一种实时算法，讨论了任意故障（形象为拜占庭问题），假设在没有故障的情况下消息延迟有上限，并且非故障进程的时钟同步到已知范围内。它实际上考虑了恶意行为，而不是简单地将这种行为用作完全不可预测的故障的隐喻。

1. **个人收获**

* 本文提出的算法假设处理器永远不会失败并且所有消息都被传递，介绍了一个复制状态机（Replicated State Machine）的实现。作为解决同步(synchronizing)问题的一种方法，它还可以被特化用来解决物理时钟的同步问题。
* 我认识到了序在分布式系统的重要性，递进理解了偏序与全序的作用，跟随Lamport在偏序的关系中将因果与并发的概念深入进行了分析。
* 文章给出一些启示性的算法与解决方案在以及被大量借鉴。后来产生的一些定义与问题，如线性一致性，顺序一致性，因果一致性等没有直接指出，但文章已经讨论和提及相关的概念，相信本文对其中的影响也是深远的。
* 论文背后包含了极其深刻的数学和哲学含义，可以将其推广到整个宇宙，触发了我重新审视时间和事件这两个概念。