Paxos Made Simple

摘要: 由简单英语表达的Paxos算法。

1. Introduction

被用来实现一个容错分布式系统的Paxos算法被认为太难去理解了，可能因为是原来的表述是由希腊语来表述的。实际上，这是其中最简单明了的分布式算法。下一节将展示这个公式算法如果满足我们期望的属性。最后一节解释了完整的Paxos算法，由构建分布式系统的状态机的共识应用获得——一个知名的途径，因为这是分布式系统理论中最常被引用到的主题。

2. The Consensus Algorithm

2.1 The Problem

假设有一个可以提出值的进程集合。一个共识算法确保被提出的值中的其中单独一个被选中。如果没有值被提出，则没有值被选中。如果一个值被选中，则进程们应该可以得知被选择的值。共识的安全要求是：

* 只有一个被提出的值可以被选择
* 只有一个值被选择
* 一个进度从不得知一个值被选中除非它确实被选中。

我们不会尝试去指定精确的活跃度要求。然而，该目标是为了确保有提出的值最终被选中，并且如果一个值被选中，则一个进程可以最终知道该值。

我们让算法中的这三种角色由三种代理类代表：proposers, acceptors, learners。在一种实现中，一个单独的进程可以表现为超过一种代理角色，但是代理至进程的映射不在考虑范围内。

假设代理们可以通过发送信息相互交流。我们使用传统异步，非拜占庭模型，也就是：

* 代理们以任意速率运行，可能因为停止而失败，可能重启。因为所有的代理在一个值被选中后失败然后重启，除非一些信息可以被一个失败并重启的代理记住，否则解决方案是行不通的。
* 消息可以通过任意时长被分发，可以重复，也可以丢失，但是不能错误。

2.2 Choosing a Value

最简单的选择一个值的方式是只有一个acceptor代理。一个proposer发送一个提案给acceptor，他将选择他收到的第一个提案的值。虽然简单，但是这个解决方案由于acceptor的失败可能导致后续的进程失败所以并不令人满意。

因此，让我们来尝试使用其他的方式来选择一个值。不同于只有一个acceptor，我们现在使用多个acceptor代理。一个proposer发送一个提案值给acceptor集合。一个acceptor可能accept（收到）提案值。当acceptor集合中的足够大部分人accept该值后，该值被选中。那么怎么算足够大呢？为了确保只有一个值被选中，我们可以使足够大的集合由代理中的多数组成。因为任意两个多数至少有一个共同的acceptor，所以如果一个acceptor只能accept最多一个值时，这是行得通的。（可以看到有很多的论文中都对多数如此概括）

在消息不丢失的情况下，我们希望一个值可以被选中即使只有一个proposer提出一个值。以下是要求：

P1. 一个acceptor必须accept他收到的第一个值。

但是该要求带来一个问题。一些值可以同时被不同的proposer提出，造成每个acceptor收到一个值的情况，但是没有一个值被acceptors中的多数所选中。即使只有两个值被提出，如果刚好分别被一半的acceptor选中，一个单独的acceptor的失败都会导致其无法得知哪个值被选中。

P1和一个值只有被acceptors中的多数选中才被选中的要求表明一个acceptor必须可以accept多个提案。我们通过给每一个提案分配一个自然数来持续跟踪一个acceptor可能accept的不同的提案，所以一个提案由一个提案序号和值组成。为了避免冲突，我们要求不同的提案有不同的序号。这依赖于其具体如何实现，所以目前我们只是假设如此。当一个提案被acceptors中的多数accept时一个值被选中。在这样的情景中，我们认为该提案（也就是该值）被选中。

我们可以允许多个提案被选中，但是必须保证所有被选中的提案有相同的值。通过提案序号，可以保证：

P2. 如果一个值为v的提案被选中，则所有被选中的更高序号的提案的值为v。

因为序号是完全排序的，条件P2保证了只有一个值被选中这个关键的安全属性。

为了被选中，一个提案必须被至少一个acceptor所accept。所以，我们可以使P2变为：

P2a. 如果一个值为v的提案被选中，则所有被任意acceptor选中的更高序号的提案值为v。

我们仍然维护P1以确保有提案被选中。因为沟通是异步的，所以一个提案可能被特定的从未收到过其他提案的acceptor c所选中。假设一个新proposer“苏醒”并且提出一个值不同的更高序号的提案。P1要求c去accept该提案，此时违反了P2a。为了同时维护P1和P2a，要求延伸P2a为：

P2b. 如果一个值为v的提案被选中，则所有由其他任意proposer提出的更高序号的提案值为v。

因为一个提案必须由一个proposer在其可以被一个acceptor accept之前提出，P2b实现P2a，也就是实现P2。

为了探索怎么满足P2b，让我们考虑我们怎么证明它拥有。我们假设一些序号为m且值为v的提案被选中并且任意序号n>m的提案的值也是v。我们通过使用n的induction来使证明更简单，所以我们可以证明序号为m值为v的提案在额外的假设之下，假设为所有被提出的序号为m..(n-1)的提案的值为v，其中i..j表示从i到j的序号集合。因为提案序号m被选中，必须有一些由多数的acceptors组成且所有acceptor都accept了它的集合C。结合之前的假设，m被选中的假设推导出：

所有C中的acceptor都accept了序号为m..(n-1)的提案，并且其中所有acceptor accept的序号为m..(n-1)的提案的值都是v。

因为任意由acceptors中的多数组成的集合S都包含了至少一个C中的成员，我们可以总结出一个序号为n的提案通过确保以下的不变条件保持值为v：

P2c. 对于任意v和n，如果一个值为v且序号为n的提案被提出，则有由acceptors中多数组成的集合S，(a) S中没有acceptor accept了序号小于n的提案，或者(b) v是更高序号的提案的值，其中所有的序号小于n的提案都被S中的acceptors所accept。

我们因此通过维护P2c来继续满足P2b。

为了保持P2c的特性，一个proposer如果像提出一个序号为n的提案，必须学习小于n的最高序号的提案，如果有的话，它将会或是已被acceptors中的多数所accept的。学习已经被accept的提案是很简单的；预测将来的acceptances是很困难的。不去尝试预测未来，proposer通过给出不会有任何其他的acceptances的承诺来控制它。换种说法，proposer请求acceptors不去accept其他小于n的提案。这引出如下算法提出提案的算法：

1. 一个proposer选择一个新的提案序号n并且发送一个请求给一些集合中的acceptors，要求如下回应：

(a) 不在accept其他小于n的提案的一个保证

(b) 如果有的话，返回小于n的最大序号的提案

我将这样的一个请求称作序号n的提案的prepare请求。

2. 如果proposer从acceptors中的多数处收到要求的回复，则他可以提出一个序号为n值为v的提案，其中v是回复中最高序号的提案的值，或在没有提案的回复时由proposer选择的任意值。

一个proposer通过发送给一些acceptors的集合一个获取被accept的提案的请求来提出一个提案。（不需要和初始请求是一样的acceptors集合）我们称之为accept请求。

这个描述了一个proposer的算法。那么acceptor的呢？他会收到来自proposer的两种请求：prepare请求和accept请求。一个acceptor可以忽略所有没有妥协安全（compromising safety）的请求。所以我们只在可以响应一个请求时可以说。他会一直对prepare请求进行响应。他可以对accept请求响应，accept一个提案，当且仅当他没有承诺不accept。换句话说：

P1a. 一个acceptor可以accept序号为n的提案当且仅当没有响应序号大于n的提案的prepare请求。

显然P1a包含了P1。

我们现在有选择一个值的完整算法，其满足要求的安全属性——假设唯一的提案序号。最终的算法可由一个更小的优化获得。

假设一个acceptor收到一个序号为n的prepare请求，但是他早就对一个序号大于n的prepare请求做了响应，因此保证了不会接受序号n的提案。对于acceptor而言也就没有理由回复新的prepare请求，因此他将不会accept这个proposer希望提出的序号为n的提案。所以acceptor忽略了这样的一个prepare请求。我们同时也让他忽略了他早就accept的提案的prepare请求。

与此优化相关，一个acceptor只需要记住他accept过的最高序号的提案和他回复过的最高序号的prepare请求。因为P2c必须被保持，一个acceptor必须记住这些信息，即使他失败重启。注意proposer可以一直放弃提案并且忘记所有的一切——只要他不提出另一个相同序号的提案。

将proposer和acceptor的动作放到一起，我们可以看到算法表现为以下两阶段。

Phase 1. (a) 一个proposer选择一个序号n并且发送一个prepare请求给acceptors的多数。

(b) 如果一个acceptor收到一个序号为n的prepare请求，n比他回复过的所有的prepare请求的序号都大，则他回复该请求保证不会accept任何小于n的提案和他accept过的最高序号的提案（如果有）。

Phase 2. (a) 如果proposer从acceptors的多数处收到了prepare请求的回复，则他发送给这些acceptors一个包含了序号为n值为v的提案的accept请求，其中v是这些回复中最高序号的提案的值，或者当回复中没有提案时会是任意值。

(b) 如果一个acceptor收到一个序号为n的提案的accept请求，他accept该提案除非他早就回复了一个序号大于n的prepare请求。

一个proposer可以提出多个提案，只要每个都遵循该算法。他可以在协议中的任意时间点放弃一个提案。（正确性是被保持的，即使请求和/或回复可能到达他们的目的地很久以后才放弃提案）如果一些proposer已经开始尝试提出更高序号的提案，那么在此时放弃一个提案可能是一个好办法。因此，如果一个acceptor因为他早就收到了一个更高序号的prepare请求而忽略一个prepare或者accept请求，则可能通知那些应该放弃提案的proposer。这是一个不影响正确性的性能优化。

2.3 Learning a Chosen Value

为了得知一个被选中的值，一个learner必须找出由acceptors中多数所accept的提案。显而易见的算法是每一个acceptor，无论何时accept一个提案，对所有的learner响应，并发送给他们该提案。这允许learners能尽快找出被选中的值，但是要求每一个acceptor对每一个learner响应——一些回复等于一些acceptors和一些learners的产物。

没有拜占庭问题的假设使得一个learner从其他learner处知道被选中的值更简单。我们可以让acceptors的回复都给一个突出的learner，然后由其通知其他learner被选中的值。该方法要求所有的learner在发现被选中的值时添加一轮新的操作。这也是不太可靠的，因为突出的learner可能失败。而且它要求一些回复的数量只等于acceptors和learners的数量之和。

更通用一点的，acceptors可以将他们的acceptances回复给一些突出learners的集合，这些集合可以在之后通知所有的learner被选中的值。使用一个大的突出learner集合可以获得更大的可靠性，但同时也有更高的通信复杂度。

由于信息丢失，被选中的值可能没有learner得知。learner可以询问acceptors他们accept了哪些提案，但是acceptor的失败可能导致他不知道acceptors的多数是否accept一个特定的提案。在这种情况下，learners将只有在一个新提案被选中时才能找出被选中的值。如果一个learner需要知道一个值是否被选中，他可以通过如上描述的算法提出一个proposer的提案。

2.4 Progress

构造一个两个proposer分别使用递增序号提出一系列的提案，并且没有一个被选中的场景很简单。Proposer p使用序号n1完成Phase1。另一个Proposer q接着使用序号n2>n1完成Phase1。Proposer p的序号n1的提案的Phase2 accept请求被忽略了，因为acceptors承诺不accept小于序号n2的新提案。所以Proposer p接着使用序号n3>n2提案开始并完成Phase1，导致Proposer q的Phase2 accept请求被忽略。然后不停循环。

为了保证进程，一个突出的proposer必须被选出并作为唯一的提出提案的proposer。如果这个突出的proposer可以和acceptors的多数成功进行通信，并且使用一个大于之前所有值的序号，则该提出的提案可以被成功accept。如果他得知一些更高序号的提案请求，可以通过放弃一个提案并再次尝试，突出的proposer最终将会选择一个更高的提案序号。

如果系统的必要条件（proposer，acceptors，网络通信）都正常工作，通过选举一个突出的proposer可以使活跃度达到要求。Fischer, Lynch和Patterson的选择一个proposer的可靠算法必须使用随机或真实时间——比如，使用超时。然而，不管选举是否成功，安全性都被保证了。

2.5 The Implementation