

本科生毕业设计（论文）

外文科技文献译文

|  |  |
| --- | --- |
| 译文题目 | Paxos一致性算法 |
| (外文题目) | Paxos made simple |
| 学 院(系) | 软件学院 |
| 专 业 | 软件工程 |
| 学 号 | 092923 |
| 学生姓名 | 罗晨 |
| 日 期 | 2013年5月16日 |

指导教师签名 日期

# 引言

用于实现容错的分布式系统的Paxos算法被人们认为很难理解，可能是由于最初版本对大多数读者来说都太古怪。实际上，Paxos算法是最简单以及最显而易见的分布式算法之一。它的中心思想是synod算法。下一章介绍了本算法遵从我们的期望属性。最后一章将一致性算法运用到构建分布式系统的状态机模型中，解释了完整的Paxos算法。状态机广为人知，因为它在分布式系统理论中被广为引用。

# 一致性算法

## 问题描述

假设一组进程可以提议一些值。一致性算法的目的是为了保证只有一个值会被选中。如果没有值被提出，那么没有值被选中。如果一个值被选中，那么其他进程应该可以有办法知道被选中的值。一致性算法的安全性需求如下：

* 所有提出的值中只有一个值可能被选中，
* 只有一个单独的值被选中，
* 在值被选中之前，进程不会得知该值被选中。

我们不会试图指定特别精确的时间性需求。但是，该算法的目标在于确保某个值会被最终选中，并且如果有值被选中，那么进程最终可以知道该值。

我们以代理的形式定义一致性算法中的三种角色：提议者（proposer）、接受者（acceptor）和学习者（learner）。在具体的实现中，某个进程可能扮演多种代理，但是代理到进程间的映射不是我们考虑的重点。

* 假设代理之间可以以消息传递的方式通信。我们使用异步、非拜占庭式的模型；
* 代理以任意的速度执行操作，可能失败停止，也可能重启。由于所有的代理在某个值被选择之后可能失败并且重启，因此除非代理可以保存某些信息，否则任何的方案都是不可行的；
* 消息可以以任意的时间达到，可以被复制或丢失，但消息的内容不会改变。

## 选择

最简单的方法就是只有一个接收者代理。某个提议者将提案发送给接收者，并且接收者会选择第一个接收到的值。尽管简单，但是该方法并不可行，因为接受者的失败会导致整个系统停滞。

因此来尝试另外一个方案，即使用多个接受者代理。某个提议者将提案发送给一组接收者。接受者可以接受提议的值。只有当足够多的接受者接受了某个值之后，才认为该值被真正的接受。所以多少才是足够多？为了保证只有一个值会被选中，我们可以让该集合包含多数接受者。因为任意两个多数集合都会至少有一个接受者重叠，因此如果接受者只能接受一个值的话该方案可以工作。

 当没有失败或消息丢失时，我们希望总有一个值会被选中，即使只有一个值被提出。这就意味着以下需求：

* P1.一个接受者必须接受它第一个收到的提议。

但该需求引出了一个问题。某些值可能几乎同时被不同的进程提出，结果导致了每个接受者都接受了一个值，但是却没有一个值被选出。甚至可能尽管只有两个可选值，如果每个都被接近半数接受者接受，某个接受者的失败也会导致不能选出一个值。

P1和只有当某个值被多数接受者接受时才被选中的需求指出接受者必须可以接受多个提议。我们通过为每个提议分配一个自然数的方式进行追踪，因此每个提议包含了序列号和值（即｛序列号，值｝）。为了减少混乱，不同的提议应该有不同的序列号。如果带有某个值的提议被多数接受者接受，则该值被选中。在这种情况下，我们说该提议（也包括提议包含的值）被选中。

我们允许多个提议被选中，但是必须保证这些不同的提议应具有相同的值。通过对提议序列号的归纳，我们可以做以下保证：

* P2.如果提议{n,v}被选中，那么其他被选中的高序列号提议的值都是v。

由于序列号是完全有序的，因此P2可以保证只有一个单独的值被选中。

为了被选中，一个提议至少应被至少一个接受者所接受。因此，我们可以通过满足以下条件来满足P2：

* P2a：如果提议{n,v}被选择，那么每一个被接受者接受的高序列号提议都包含值v。

我们仍然通过P1来保证某些提议会被选中。如果通信是异步的，那么一个提议可能被从未接受过任何提议的接受者c所接受。假设一个提议者启动，并且发出了一个高序列号的提议，但是该提议带有一个不同的值。P1需要c接受该提议，但是这样会违反了P2a。因此，为了维护P1和P2a，需要将P2a强化为：

* P2b：如果提议{n,v}被选择，那么之后提议者提出的任何高序列号的提议的值都是v。

由于某个提议在被接受者接受之前必须要由某个提议者提出，因此P2b蕴含了P2a，并且P2a蕴含了P2。

为考虑如何满足P2b，首先考虑我们将如何证明它。我们假定提议{m,v}被选中，这说明任何带有序列号n(n>m)的提议都拥有值v。为了简化证明，我们将对n进行归纳法。因此我们可以证明当每个提议（编号从m到n-1）的值都是v，那么编号为n的提议的值同样为v。对于被选中的提议{m,v}，一定存在一个集合C以包含多数接受者，并且C中的接受者都接受了{m,v}。将该假设与归纳假设合并，则提议{m,v}被选中意味着：

* C中的每个接受者都接受了序列号从m到n-1的提议，并且这些提议都有值v。

由于任何包含多数接受者的集合S都会至少包含C中的一个成员，因此我们可以推断序号为n的提议的值为v，通过维护以下不变量：

* P2c：对任何v和n，如果提议{n,v}被提出，将会有包含多数接受者的集合S，使得要么(a)S中没有接受者接受了序号小于n的提议，要么（b）v的值是接受者接受的序列号最大的提议的值。

通过保持P2c，就能满足P2b。

为了维护P2c，某个提议者在提出某个序号为n的提议之前，必须知道那些已经或即将被选择的并且比n小的最大序列号的提议。知道已经被接受的提议比较容易，但是预测将来的接受是困难的。因此提议者不会进行预测，而是直接假定不会有这样的接受情况。换言之，提议者会请求接受者不会接受任何序列号小于n的提议。这就导致了如下的提议算法：

1. 提议者选择一个新的序号n，并且向每个接收者发送请求，并请求以下内容：
   1. 保证不会接受任何小于n的提议
   2. 它接受过的比n小的最大序号提议。

该请求被称为准备请求。

1. 如果提议者收到了多数接受者的回复，那么它可以提出提议｛n,v｝，此处的v是所有的回复中序号最大的提议的值，或者当回复中没有提议时，v是由提议者所决定的值。

提议者通过向接受者发送接受该提议的请求，并且该请求被称为接受请求。

以上描述了提议算法。但是对于接受者应该如何处理？接受者可能收到提议者的两种请求，准备请求和接受请求。一个接受者可以忽略任何请求，而无需担心安全性。因此，我们只需要讨论当接受者被允许回复请求的情况。它可以总是回复准备请求。而当且仅当它没有承诺过时，它才能回复接受请求。换句话说：

* P1a：一个接受者可以接受一个序号为n的提议，当且仅当它没有响应过一个比n大的准备请求。

可以看出P1a包含了P1。

我们现在已经有了可满足安全需求的选择算法—通过假定唯一的提议序号。算法的最终版本则需要一点小优化。

假设一个接受者收到了序号为n的准备请求，但是它已经回复了一个序号值比n大的准备请求，应该保证不接受任何序号为n的提议。它没必要再回复新的准备请求，因为它不会接受序号为n的提议。因此我们让接受者忽略这种准备请求。我们同样让接受者忽略它已经接受过的提议的准备请求。

通过以上优化，接受者只需要记住它曾经接受过的提议的最大序列号以及它回复过的准备请求中的最大序号。由于P2c必须是应当保证的，因此接受者即使在失败重启时，也应该记住该信息。注意提议者可以丢弃提议，只要它不会尝试以同样的序列号提出值不同的提议。

把提议者和接受者的操作组合在一起，我们可以看到该算法共分为两个阶段：

**阶段1.**(a)提议者选择序列号n，并且向多数接受者发送准备请求

(b)如果接受者收到了带有序列号n的准备请求，并且n大于它目前已经回复的准备请求的序列号，那么它将回复该请求，并保证不会接受任何小于n的提议，同时附带已经接受的提议的序号最大值。

**阶段2.** (a)如果提议者收到了多数接受者的回复，那么它会发送一个带有提议{n,v}的接受请求，这里的v是所有的回复中序号最大的提议的值，或者当回复中没有提议时，v是由提议者所决定的值。

(b)如果接受者收到了序号为n的接受请求，除非它已经回复过序号大于n的准备请求，否则它将接受该提议。

提议者可以提出多个提议，只要对每个提议都遵从该算法。它也可以在任何时间丢弃某些提议（尽管某些请求或回复可能在提议被丢弃了很久之后才到达，但并不会影响算法的正确性）。如果其他的提议者已经开始提出更高序列号的提议，那么最好能够放弃当前提议。因此，如果一个接受者由于已经接受过更高序列号的准备请求而忽略了某个准备或接受请求，那么它应该可能通知提议者，并且提议者将丢弃该提议。这是一个性能上的优化，但并不会影响正确性。

## 学习决议

为了学习被选中的值，学习者必须发现多数接受者所接受的提议。一个显而易见的算法是每当接受者接受一个提议时，都把该提议发给所有的学习者。这允许了学习者尽快的找到被选出的值，但是需要每个接受者发送给每一个学习者—消息的个数等于接受者和学习者个数的乘积。

非拜占庭失效的假设允许学习者通过另外的学习者发现被接受的值。我们可以让接受者向唯一的学习者发送选择的值，而学习者将反过来通知其他学习者被选择的值。这种方式需要额外的通信回合数以使得所有的学习者发现被选择的值。同时由于唯一的学习者可能失效，该方案也缺乏可靠性。但是它需要的消息个数为接受者和学习者个数的和。

另外，接受者可以向一组学习者发送选中的提议，并且其中的每个学习者都可以向其他学习者通知选中的值。选择更大的学习者集合会提供更好的可靠性，但是会导致网络通信更加复杂。

由于消息丢失的可能性，某个被选中的值可能没有学习者发现过它。学习者可以向接受者询问它们所接受的值，但是接受者的失败使得不可能确切的知道某个提议是否被多数接受者接受。在这种情况下，只有当新的提议被选择时，学习者才有可能发现被选择的值。如果学习者需要知道某个值是否被选择，它可以让一个提议者用以上描述的方法提出一个提议。

## 行进

我们将假设如下场景，两个提议者持续的提出递增序列的提议，但是却从未有提议被接受。提议者p以序列号n1完成阶段1，另外一个提议者q以序列号n2(n2>n1)完成阶段1。然后，p在阶段2中的接受请求会被接受者忽略，因为接受者已经保证将不会接受序列号低于n2的提议。因此p开始用序列号n3(n3>n2)完成阶段1，又导致了q在阶段2中的接受请求被忽略。

为了保证行进，必须选出一个唯一的提议者提出提议。如果该提议者可以成功的与多数接受者通信，并且如果它使用了序列号大于当前最大序列号的提议，那么它会成功的提出提议，并且会被接受。如果它发现某些提议的序列号比当前序列号更大，则可丢弃该提议并且进行重试，因此该提议者最终可找到足够大的序列号。

如果系统足够多的组件（提议者，接受者和通信网络）可以正常工作，系统可以通过选出单个唯一的提议者以保证及时响应。Fischer，Lynch和Patterson的著名实验结果指出，一个可靠的提议者选举算法必须使用随机性或者实时性（例如超时机制）。然而，选举的成功或失败都不会影响安全性。

## 实现

Paxos算法假设一组网络进程。在一致性算法中，每个进程都扮演了提议者、接受者或学习者的角色。该算法选举出一位领导，以扮演唯一的提议者和学习者。Paxos一致性算法的请求和响应可以以普通消息的形式发送（为了避免混乱，响应消息通常带有相应的提议序列号）。另外使用持久化存储可保证接受者即使在失败时也能记住信息。接受者在响应之前，应将响应信息记录在持久化存储中。

接下来还一种机制以确保两个提议序列号不会重复。不同的提议者可从没有交集的集合中选择各自的序列号，因此不同的提议者从不会选择同样的值。每个提议者都记住当前它已经使用的最大序列号，并且在开始阶段1时，使用比该序列号更大的序列号。

# 实现状态机

一个实现分布式系统的简单方法是一系列的客户端向中心服务器发送请求。服务器可被看作是一个顺序执行客户端请求的确定状态机。状态机拥有当前状态；它通过将一个请求作为输入，并且产生输出和新的状态以完成一步操作。例如，分布式银行系统的客户端可能是出纳员，并且该状态机可能包含了所有用户的余额。取款操作可以描述为通过执行一个当余额大于要取的金额时，将余额减去相应余额的请求，并且将新老余额作为结果输出。

对于单服务器的实现方案，如果该服务器失效，那么该方案也会失效。因此我们使用一系列的服务器，其中每个都独自实现了状态机。由于状态机是确定的，那么当服务器在执行相同顺序的请求时，所有的服务器都会产生相同的输出，并产生相同顺序的状态。发出请求的客户端可以采用任意服务器的输出作为结果。

为了保证所有的服务器执行相同顺序的状态机请求，我们实现了一系列独立的Paxos一致性算法实例，被第i个实例选中的值就是第i个状态机请求。每个服务器在各自的算法实例中都扮演了所有角色（提议者、接受者和学习者）。到目前为止，我们假设服务器的集合是固定的，因此所有的一致性算法实例使用相同的代理集合。

在正常操作中，一台服务器被选作领导，以在所有的一致性算法实例中扮演唯一的提议者的角色。客户端向领导发送请求，并且由领导决定了请求出现的顺序。如果领导决定了某个客户端请求应该是第135号，那么它会尝试让该请求被第135个一致性算法实例选为值，并且通常会成功。服务器可能由于某些故障而失败，也可能由于其他服务器认为自己是领导，并且对135号请求有不同的看法。但是一致性算法保证了第135号最多只有一个请求。

该方法高效的关键在于在Paxos一致性算法中，直到阶段2提议才会被真正选出。另外需要记住，当提议者完成阶段1时，要么需要提议的值已经确定了，要么提议者可以提出任何值。

以上讨论了Paxos状态机实现在正常操作中的工作方式。下面将讨论可能发生的错误。我们将考虑当前一个领导刚刚失败，而新的领导已经被选出的情况（系统启动是特殊的情况，因为之前没有请求被提出过）。

新的领导在被选举出之前都作为所有一致性算法实例的学习者，因此应该知道大多数已经被选出的请求。假设它知道请求1-134，138和139，也就对应了在一致性算法实例1-134，138和139中被选出的值（我们将在后面看到这种缺口是如何产生的）。然后它将执行135-137和所有大于139的实例的阶段1。假设这些执行的结果决定了实例135和140应该提议的值，但是对其他的实例中的提议值没有限制。领导将会执行135和140实例的阶段2，并因此选择出135和140号请求。

领导和其他所有的服务器都知道领导所知道的请求，并且现在可以执行请求1-135。然而，虽然服务器知道请求138-140，但却不能执行这些请求，因为136和137号请求还没有被选出。领导可以让后续的两个客户端请求作为136和137号请求。另外，我们可以提出两个空操作提议，以填充136和137号请求（领导可以通过执行136和137一致性算法实例的阶段2）。一旦这些空操作被选中，138-140号请求可以被执行。

现在1-140号请求已经被选出。领导同样已经完成了140以上一致性算法实例的阶段1，并且它可以在这些实例的阶段2里提出任意的值。它将下一个客户端请求作为第141号请求，并将它作为值提出在141号一致性算法实例的阶段2中。同样它也会将下一个客户端请求提议为142号请求，等等。

领导可以在141号请求被选中之前提出142号请求。有可能141号请求提议的所有消息都丢失了，并且142号请求在所有服务器知道第141号请求是什么之前被选中。当领导不能收到141号实例阶段2的期望回复时，它会尝试重发这些消息。如果一切顺利的话，那么它提议的请求将被选中。然而，如果它在此之前失败了，则将会在请求序列中留下缺口。通常来说，假设领导会提前收到a个请求，即在第1-i号请求被选出后，领导可以提出第i+1到i+a号请求。因此会产生最多a-1个请求的间隙。

新选出的领导会为无穷多的一致性算法实例执行阶段1，在上面的场景中，领导会为135-137以及所有大于139的实例执行阶段1。领导可以向其他的服务器发送一条简单的合理短消息，并为所有一致性算法实例使用相同的序列号。在阶段1中，当接受者收到其他提议者阶段2的消息时，它的回复中会包含除了OK的额外信息。（在上面的场景中，这只是针对实例135和实例140）。因此，接受者可以为所有实例都回复一条合理的短消息。并且多次重复执行阶段1不会带来问题。

由于领导的失败和选举是较少发生的事件，执行状态机请求的有效开销（达到一致的请求/值）只包括一致性算法阶段2的开销。可以看出Paxos一致性算法的阶段2拥有目前容错性一致性算法的最小开销。因此，Paxos算法是非常高效的。

以上对系统正常操作的讨论假定系统中只有一个单独的领导，除了当旧领导失效并选举出新领导的短暂情况。在异常情况下，领导选举可能失败。如果没有服务器作为领导，那么将没有新的请求会被提出。如果多台服务器都认为自己是领导，那么他们在同一个一致性算法实例中都可以提出提议，这会导致每个提议都不会被选中。然而，安全性是可以保证的，因为两个不同的服务器不会对已选出的第i个请求产生异议。单领导的模型只是用来保证进度。

如果服务器集合可以变化，那么则需要有某种机制决定哪些服务器负责哪些一致性算法的实例。最简单的方式是通过状态机本身。当前服务器的集合可以作为状态的一部分，并且可以由正常的状态机请求所改变。为了允许领导提前获得a个请求，我们需要在第i个状态机请求之后，允许当前集合中的服务器执行第i+1到i+a的请求。这使得可以通过简单配置来实现任意复杂的算法。