1 简介

Paxos算法作为容错的分布式系统实现已经广泛的被认为是难理解的，这可能由于之前的描述对于许多读者来说不容易读懂。实际上，它是最简单和最显而易见的分布式算法之一。它实际上是一致性算法“会议”算法的一种。在下个章节将展示这个一致性算法无法避免的属性，而这些属性我们想让他满足的。最后的章节中通过构建一个状态机取得一致性的分布式系统的简单程序，来解释完全的paxos算法，当然这个方法是众所周知的，因为这个可能是最经常在分布式理论里提及的主题【4】。

2 一致性算法

2.1 问题

假设一个可以进行建议的进程的集合。一个一致性算法要保证在所有被建议的值中只有一个单一的值可以被选定。如果没有值被建议，当然也就没有值被选定。如果一个值已经被选定了，则这些进程应该能够了解到这个被选定的值.

一致的安全要求如下：

1. 仅仅一个被建议的值才可能被选定
2. 仅仅一个单一的值被选定
3. 一个进程不会去了解一个值，除非他已经被实际的选定了

我们将不尝试去详细说明精确的活性需求。无论如何，目标是确保一些被建议的值最终被选定，并且如果一个值被选定了，然后一个进程最终能够学习到这个值。我们让这三个一致性算法中的三个角色扮演三种代理人类型，建议者，同意者，学习者。在算法实现中，一个单一的进行可能会扮演超过一种类型的代理人，但是，在这里我们并不关心代理人到进程的映射。

假设代理人们能够通过发送消息彼此通信。

我们使用通常的异步的非拜占庭模型，即：

1.代理人以任意速度进行操作，也许中止后失败，也许重启。由于所有的代理人在一个值被选定后也许失败，并且然后重启，所以如果一些信息不被一个失败的重启的代理人记录下来的话，那么任何一个解决方法都是不可能的。

2消息可能花费任意长的时间去传输，也可能重复，丢失，但他们的内容会是正确的。

2.2 选择一个值

选择一个值的最简单的方式是只有一个单独的同意者。一个建议者发送一个建议给同意者，同意者会选定他所接收的第一个建议值。虽然简单，但这个解决方案不能令人满意，因为这个同意者的故障会是使任何的更进一步的处理成为不可能。所以，让我们尝试另一种选定值的方法。作为单一同意者的替代，让我们使用多个同意者代理人。一

个建议者发送一个建议值给一组同意者集合。一个同意者也许同意这个值。当同意者集合中的足够多的成员同意了这个值，那么这个值就被选定了。多大才是足够大呢？为了确保仅仅一个单一的值被选定，我们能够让代理人中的大多数组成这个足够大的同意者集合。因为任意两个大多数集合至少有一个共公共的同意者，如果同意者最多只能同意一个值的话，那么这个将是有效的。 (最初在【3】中，有一个对“大多数”显而易见的概括，这个已经被众多的论文提及。)

在不会失败和消息丢失的情况下，我们想让一个值被选定，即使只有一个值被单一一个建议者提出。这个表明了如下的要求：

P1.一个同意者必须同意他接收到的第一个建议

但这个要求引发了一个问题。几个值能够被不同的建议者差不多同时建议，这个导致了一种情况：每个同意者已经同意了一个值，但没有一个值是大多数的人同意。即使对于仅仅两个建议值，如果每个都被差不多一半的同意者所同意，一个单一的同意者的失败，会使知道那个值是被选定的成为不可能。P1 和 仅当被大多数同意者同意的值才能被选定的要求，意味着一个同意者必须被允许同意超过一个建议。我们通过对每一个建议分派一个自然数序号的方式来记录一个同意者可能同意的不同的建议，所以一个建议包含一个建议序号和一个值。为了避免混淆，我们需要不同的建议有不同的序号。这个如何做到，依赖于实现，以至于我们现在仅仅只能假设他。当一个携带着值的单一的建议被大多数同意者同意时，这个值就被选定了。在这种情况下，我们认为这个建议（还有他的值）被选定了。

我们能够允许多个建议被选定，但是我们必须保证这些所有被选定的建议有同样的值。通过基于建议序号的归纳，这足够保证：

P2.如果一个值v的建议被选定，则每个被选定的更高序号的建议都有值v.因为序号是完全顺序的，条件p2保证了严格的安全属性，即仅一个单一的值被选定了。为了被选定，一个建议必须被至少一个同意者同意。所以我们能够满足p2,通过满足：

P2a.如果带有值v的建议被选定，则每个被任意同意者同意的更高序号的建议拥有值v.

我们一直维护着p1去确保一些建议被选定。由于通信是异步的，虽然稍微特别的同意者c从来没有接收到任何的建议，一个建议还是能够被选定。假设一个新的建议者“苏醒“过来，并且发布了一个带有不同值的更高序号的建议.P1需要c 去同意这个建议，这违背了p2a.为了维持p1和p2a，需要加强p2a为：

P2b.如果一个带有值v的建议被选定了，则每个被建议者发布的更高序号的建议拥有值v.

因为一个建议必须在他被一个同意者同意之前被一个建议者发出，所以p2b暗示了p2a,p2a转而暗示了p2.为了找到如何满足p2b,让我们思考一下我们将如何证明他。我们将假设一些序号为m 值为v的建议被选定，并且任意序号为n(n>m)的被发出的建议也有值v. 我们将通过使用基于n的归纳来做一个较简单的证明，如此，在每一个被发布的编号为m到n-1（i到j意思为从i到j之间的一组序号）之间的建议都有值v这一附加的假设下，我们能够证明序号为n的建议拥有值v. 对于被选定的序号为m的建议，一定有一个大部分的同意者的集合C,这些同意者都同意了这条建议。

集合这个归纳假设，序号为m的建议被选定的假设暗示了：

在C中同意了序号m到n-1中的一条建议的每一个同意者和每条序号在m到n-1之间的被任意同意者同意的建议都拥有值v.由于由大部分同意者组成的集合S,至少包含一个C中的成员，所以我们通过确保维持下面给出的不变式，可以得出一个序号为n的建议拥有v这一结论：

P2c.如果一个值为v和序号为n的建议被提出，对于任意的v和n,之后都有一组由大部分同意者组成的集合S,就像下面这样，或者S中没有同意者同意任何序号少于n 的建议，或者值v是所有被S中这些同意者同意的编号小于n的所有建议中，序号最大的建议所携带的值。

我们因此能够通过维持p2c的不变式来满足p2b.

为了维持p2c不变式，一个想要提出序号为n的建议者，必须学习已经被大多数同意者同意的或将被同意的序号比n小的建议中的序号最大的建议（如果有的话）.学习已经被同意的建议是很容易的；预测未来的同意情况是困难的。作为预测未来的替代，建议者通过做出一个“将不在有任何如此的同意”的保证来控制这个。换句话说，建议者要求同意者们不再同意序号低于n的建议。

这个引出了下面的关于提出建议的算法。

1.一个建议者选择一个新的建议序号n并且给一些同意者集合中的每一个成员发送一个请求，询问回应:

(a)不再同意一个序号小于n的建议的承诺，并且

(b) 已经被同意的序号比n小的建议中的最大序号建议，若这个建议存在。

我将这样一个请求，叫做序号n的准备请求。

2.如果这个建议者接收到了从所有同意者中大部分返回的被请求的回应，然后他能发送一个带有序号n的且拥有值v的建议,在这里值v是这些回应中最大，序号的建议的值，或者如果这些报告者们报告没有建议的话则这个值是被建议者选定的任意值。

一个建议者通过发给一些同意者集合一个“这个建议已经被接受了”的请求。（这个同意者集合和回应初始请求的同意者集合不需要是同一个）。让我们叫这个为接受请求。

这个描述了一个建议者的算法。那么同意者呢？他能从建议者那接受*两种类型的请求：准备请求和同意请求*。一个同意者可以安全的忽略任何的请求。所以，我们需要说仅当他被允许时才去响应一个请求。他能总是响应一个准备请求。它能够响应一个同意请求去同意这个建议，当且仅当他没有许诺不这样做。

换句话说：

P1a. 一个同意者能够同意一个编号为n的建议，当且仅当他还没有回应一个编号比n大的准备请求。

观察到p1a包含了p1。

对于选择一个值，我们现在有了满足所需安全属性的一个完全的算法，这个算法假设了唯一的建议序号。经过做一个小优化，将获得最终的算法。假设一个同意者接收了一个序号为n的准备请求，但是他已经回应了一个序号比n大的准备请求，因而许诺了不再同意任意的新的序号为n的建议。然后这个同意者也没有理由去回应这个新的准备请求，这是因为这个同意者将不同意这个建议者想要发布的序号为n的建议。所以我们使这个同意者忽略如此的一个准备请求。我们也使他忽略一个已经被同意的建议的准备请求。

因为这个优化，一个同意者仅需要记住曾经被同意的最高序号的建议，和已经回应的最高序号的准备请求。由于不管有没有失败，p2c必须被保持不变，一个同意者必须记住

这些信息，即使他失败然后重启。注意：只要建议者没有发布另一个同样序号的建议,他总是能够抛弃一个建议，并且忘记关于这个建议的所有。把建议者和同意者的行为放到一起，我们能够看到这个算法操作有两个阶段：

*阶段1，(a) 一个建议者选择一个建议的序号n,并且向大多数同意者发送一个带有这个*序号n的准备请求

（b）如果一个同意者接收到一个带有序号n的准备请求，并且这个序号比他已经回应的的任意一个准备
请求的序号高，然后他回应一个请求，许诺不再同意任意序号比n小的请求，并且这个回应会带上他已经同意的最大序号建议（如果有的话）。

阶段2.(a)如果建议者从大多数同意者那接收到了他的一个准备请求的回应（序号为*n*）,*然后他发送关于这个序号为n值为v的建议的同意请求给上述这些同意者，这里的v是这些回应中最大序号建议所带的值*或者如果这些回应里没有报告任何建议时,那么v就可以是任意值。

（b）如果一个同意者接收到了一个对序号n的建议的同意请求，除非他已经*回应了一个比序号比n大的建议的准备请求，否则他必须要同意这个建议。*

一个建议者可以发出多个建议，只要每一个都遵守了算法。
在协议中间，它能够在任何时间抛弃一个建议（即使请求或者回应或者两者在它们抵达目的地时这个建议已经被抛弃了很久，正确性还是依然被保证了。如果建议者已经开始尝试去发布一个比当前更高编号的建议，则要抛弃当前的，这可能是一个好的想法。因此，如果同意者因为已经接收到了一个更高编号的准备请求而忽略了一个准备/同意请求，之后它应该通知建议者，而这个建议者然后应该放弃它的建议。这是一个不影响正确性的性能优化。

2.3 学习一个选定的值

为了学习到一个被选定的值，一个学习者必须找出已经被大多数同意者同意的建议。这个显而易见的算法是对于每一个同意者，无论何时同意了一个建议都对所有的学习者发送这个建议。这允许了学习者尽快的找到被选定的值，但是这个算法需要每个同意者发送给每一个学习者响应，响应的数量等于同意者数量和学习者数量的乘积。

没有拜占庭失败的假设，使一个学习者从另一个学习者那找到被同意的值是容易的。我们能够使同意者们发送他们的同意响应给一个著名的学习者，当一个值被选定时，这个学习者反过来通知其他的学习者。使用这个方法，所有的学习者发现被选定的值，需要一个额外的步骤。这个也是不可靠的，因为著名的学习者可能失败。但是它需要的响应数量仅仅等于同意者数量与学习者数量之和。

更通用的是，同意者能够发送带有同意的建议的响应给一组显著的学习者集合，当某个值被选定时集合中的每一个成员会通知所有的学习者。使用一个较大的著名的学习者集合能够提供更好的可靠性，但会花费更多的通信复杂度。

由于消息丢失，即使没有学习者找到被选定的值，这个值也能够被选定。 学习者能够询问同意者那些被他们同意的建议，但是一个同意者的失败能够使得这个变为不可能，无论大部分的同意者是否同意了一个特别的建议。这那种情况下，学习者将仅能找到新的被选定的建议。如果一个学习者需要知道值是否被选择，它能够使一个建议者使用上述算法发布一个建议。

2.4 过程

容易构建一种场景，两个建议者各自持有一个序号递增的未被选定的建议序列。建议者p对于一个序号为n1的建议，完成了阶段1。之后另一个建议者q，对于序号n2（n2 > n1）的建议完成了阶段1。由于同意者们已经许诺了不再同意任何新的序号大于n2的的建议，建议者p的对于序号为n1的建议的阶段2的同意请求被忽略了。所以，建议者p然后为新的序号为n3的建议开始并且完成阶段1，导致了建议者q忽略了序号为n2的建议的阶段2的同意请求等等这些。

为了保证过程，一个著名的建议者必须被选择出来，而且他是唯一的一个尝试去发布建议的。如果这个著名的建议者能够同大部分的同意者成功通信，并且如果他使用了一个建议，而且这个建议的序号比任意一个已使用的建议的序号大，然后他将在发布一个被同意的建议上取得成功。 通过抛弃一个建议和再一次尝试去了解到一些更高序号的建议的请求，这个著名的建议者将最终选择一个足够高的序号。

如果系统的足够的部分（建议者，同意者，网络通信）正在正确的工作，活跃性可以通过选举一个单一的著名建议者被达成。The famous result of Fischer, Lynch, and Patterson Fischer,Lynch 和Patterson的著名的结果【1】，表明选举一个建议者的可靠算法必须是使用随机或者是实时的，例如使用超时。不管怎样，不论选举成功或失败，安全性必须被保证。

2.5 实现

Paxos算法【5】假定了一个多个进程的网络。在它的一致性算法中，每一个进程扮演了建议者，同意者，学习者的角色。这个算法选择了一个领导来扮演著名的建议者和著名的学习者。paxos一致性算法在上文中被精准的描述了，在上文中请求和回应被当做普通的消息来发送。（回应消息被打上了与建议号一致的数字来避免混淆。）稳定的存储（在失败中也可保存信息）被用来维持同意者必须要记住的信息。一个同意者在真正发送意向响应之前，将这个响应记录在了他的稳定存储中。

所有的这些是为了描述保证没有两个建议曾经带有同一个编号被发布这一机制。不同的建议者从分离的数字集合里选择他们的编号，所以两个不同的建议者不会发布同一编号的建议。每个建议者记得（在稳定存储中）被尝试发布的最高编号的建议，并且使用比之前用过的更高建议编号来开始阶段1.

3 实现一个状态机

一个实现分布式系统的简单的方式是实现一个由一组客户端向中央服务器发布命令的系统。这个服务器能够被描述为一个确定性状态机，这个状态机表现为客户端命令在一些序列里。这个状态机有一个当前状态；他通过从序列里取出一个作为一个输入命令来运行一步，并且输出和设定一个新的状态。例如，分布式银行系统的客户端们也许是出纳员，并且这个状态机也许是由所有用户的账户余额组成。取款将表现为执行了一条状态机命令，这条命令减少了一个账户的余额，当且仅当这个账户的余额比取的款项多，这个命令同时输出旧的和新的余额。

如果服务器失败，则用了单一中央服务器的实现也失败了。我们因此使用一组服务器作为代替，每一个都独立的实现了状态机。由于状态机是确定性的，如果他们所有都执行了同样的命令序列，则所有的服务器将得出同样的状态序列并输出。一个客户端发布了命令之后能够使用为了他而被任意服务器生成的输出。

为了保证所有的服务器执行了相同的状态机命令序列，我们为paxos一致性算法的所有的分离实例的实现了一个序列，被第i个实例选择的值是序列里的第i个状态机命令，每个服务器在每一个算法实例里扮演了所有的角色（建议者，同意者，学习者）.现在，我假设服务器集合是被确定的，所以所有的的一致性算法实例使用了一样的代理人集合。

在通常的操作中，一个单一的服务器被选举成为领导，他的行为就像著名的建议者（唯一的一个提出建议的）在所有的一致性算法实例里的一样。客户端们发送命令给领导，这个领导决定了每一条命令应该出现在序列里的位置。如果领导决定了一个确定的客户端命令应该是第135个命令，则他尝试去使这个命令被选择为这个一致性算法中的第135个实例值。这个通常情况下会成功。由于各种失败或者因为其他的服务器也相信他自己是这个领导并且对于第135个命令应该是什么有着不同的想法，这个操作也许会失败。但是这个一致性算法确确保最多一个命令能够被选择为第135命令。

在paxos一致性算法中，这个方法的效率的关键在于这个被建议的值直到阶段2才被选定。回忆这个，在建议者算法中，完了阶段1后，或者这个被建议的值被决定，或者这个建议自由的建议任意的值。

我现在将描述paxos状态机实现是如何在普通操作中工作的。之后，我将讨论导致错误的方面。当之前的领导刚刚失败，并且一个新的领导已经被选择时，我考虑了会发生的事情.(系统启动是一个特殊的情况，这时还没有命令被建议)

这个新的领导在一致性算法的所有的实例中也是一个学习者，他应该知道已经被选择的大部分的命令。假设他知道1-134，138和139的命令，这个是在1-134,138,139一致性实例中被选择的值。（我们之后将看到在命令序列里的这样的的空白是如何产生的）。之后他执行了135-137实例和比139大的实例的阶段一。（下面，我会描述这是怎么产生的）假设这些执行的结果决定了被实例135和140建议的值，但在所有的其他实例里留下了这个不受约束的值。这个领导之后执行了实例135和140的阶段2，因此选择了命令135和140.

这个领导还有学习了这个领导的所有命令的其他服务器，现在能够执行命令1-135.然后，他不能执行他知道的命令138-140，因为命令136和137还没有被选定。领导能够取出被客户端请求的下两条命令作为136和137。 作为上述的代替，我们让他通过建议一个特别的“noop”指令立即填充空白作为命令136和137，这个命令不会让状态改变。（通过执行一致性算法的136,137实例的阶段2来实现）。一旦，这些noop命令被选择，命令138-140能够被执行。

命令1-140现在已经被选择了。领导对一致性算法的比140大的所有实例也已经完成了阶段1，并且在这些实例的阶段2中，他自由地建议任意的值。他为下个客户端请求的命令分配命令序号141，在一致性算法的141号实例的阶段2中作为值进行建议。他建议下个他接收的客户端命令作为142号命令，等等。

领导能够在他了解到他建议的141好命令已经被选定之前，就建议142号命令。有可能他发送的所有的关于建议的141号命令的消息丢失了，并且也有可能在任何其他的服务器了解到领导建议的141号命令是什么之前, 142号命令就被选定了。当领导在实例141上接收对阶段2的预期的回应失败时，他将重新传送这些消息。如果所有的消息成功了，他建议的命令将被选定。无论如何，他能够先失败而留下一个空白在选定命令序列中。

一般情况下，假设领导能够获取位置在最前的a个命令，这就是他能够在命令1到i被选定后就建议的从i+1到i+a的命令。然后一个相当于a-1个命令的空白产生了。

在上述这个场景里对于实例135-137和所有比139大的所有实例，一个新选定的领导对无数多的一致性算法实例执行阶段1。他能够通过发送一个单一的合理的短消息给其他的服务器,从而对所有的实例使用同一个建议编号。在阶段1，只有他已经从一些建议者那接收到了一个阶段2的消息，一个同意者才回应不只是一个简单的(在这个场景中，仅仅是对实例135和140的例子。)因此，一个服务器（作为同意者）能够对所有的实例响应一个单一的合理消息。这些无数多的阶段1的实例的执行，没有因此产生问题。因为领导的失败和一个新领导的选举应该是非常少发生的事件，一个状态机命令的执行（就是在这个命令/值上达成一致）的有效开销仅仅是一致性算法的阶段2的执行开销。

能够看出，paxos一致性算法的阶段2，在任意为了在失败中达成一致的算法中有最小的可能开销。因此，paxos算法是本质上最佳的。

这个关于系统的普通操作的讨论假定了除了在当前领导失败和选举一个新的领导之间有一个短的周期外，总是有单一的一个领导.在一个特殊情况下，这个领导的选举可能失败。如果没有服务器扮演领导，则没新的命令将被建议。如果多个服务器认为他们自己是领导，则他们所有人可能在同一个一致性算法的实例上建议值，这能避免任何的值被选定。无论如何，安全性被保留了，两个不同的服务器将从来不会不同意这个值被选为第i个状态机命令。单一领导的选举，需要保证过程。

如果服务器集合被改变了，从而必须有一些决定了实现这个最简单的方法时通过状态机本身。当前的服务器集合能够使状态的一部分并且能够被普通的状态机命令改变。我们能够允许一个领导取得在前面的a个命令，通过指明这个执行了一致性算法的实例i+a的服务器组执行第i个状态机命令之后的状态。
这允许了任意复杂重构算法的一个简单实现。