The Part-Time Parliament (兼职议会)

Leslie Lamport

This article appeared in ACM Transactions on Computer Systems
16, 2 (May 1998), 133-169. Minor corrections were made on 29 August 2000.

翻译/批注：hutu92

2018 年 9 月 25 日

前言

这篇论文前不久在 TOCS (ACM Transactions on Computer Systems) 编辑室的档案柜后面被发现。尽管年代久远，作者仍然认为它值得发表。因为作者最近在希腊的一些小岛上做野外作业不能到达，所以请求我来为发表做准备。

作者貌似是个只业余对计算机科学感兴趣的考古学家。这非常不幸；尽管他描述的模糊的古老的 Paxon 文明对大多数计算机科学家来说没什么兴趣，但它的议会系统对于怎样在异步环境中实现一个分布式计算系统来说是个杰出的模式。实际上，Paxons 对其协议所做的一些改进似乎在系统文献中找不到。

第一章 The Problem

1.1. Paxos 岛

在这个千年的早期（公元 10 世纪初），爱琴海上的小岛 Paxos 是一个兴盛的商业贸易中心[1]。经济发展带来了政治的进步。Paxos 的公民们用议会形式的政府取代了原先的神权统治。但是对 Paxos 人来说，做生意才是头等大事，城市职责反在其次，没有 Paxos 人愿意把他全部的时间投入到议会事务中。所以 Paxos 的议会必须在每个议员都可能随时缺席的情况下也能工作下去。

兼职议会所面对的问题正好能对应于今天的容错式分布式系统所面对的问题：议员对应于分布式系统中的处理进程(processes)，而议员的缺席对应于处理进程的宕机。因此 Paxos 人的解决方法或许值得计算机科学借鉴。我这里会介绍 Paxos 议会协议的一个简短历史，接下来会简短讨论一下其与分布式系统的相关性。

Paxos 文明已毁灭于一次异族入侵，考古学家只是最近才开始发掘他们的历史。我们对于 Paxos 议会的所知因此比较零散。虽然基本协议是知道的，但对许多细节我们却一无所知，而这正是我们感兴趣的地方，因此我将忝为推测 Paxos 人在这些具体细节上可能的做法。

1.2. 要求（Requirements）

议会的主要任务就是决定这片土地上的法律，法律是由议会通过的一系列法令定义的。一个现代的议会可以雇佣秘书来记录它的每一个活动，但是在 Paxos 没有一个人愿意始终呆在议会大厅里作为一个秘书从头到尾参与每一个会议。取而代之的是每一个议员都会维护一个律簿（ledger），用来记录一系列已通过的法令，每个法令会带有一个编号。例如议员 Λ[2] 的律簿里有这样一个条目：

155：橄榄税是每吨 3 个银币

如果她相信信议会通过的第 155 号法令设置了橄榄的赋税为每吨 3 个银币。律簿是由擦不掉的墨水书写的，所以其中的条目不能被改变。

议会协议的第一个需求就是律簿的一致性。也就是任何两个律簿都不能有互相矛盾的内容。假设议员 Φ 在他的律簿中有这样一个条目：

132：只有橄榄油可以用来做灯

[1] 这应该不会和爱奥尼亚的 Paxoi 岛混淆，Poxoi 有时也被破读为 Paxos

[2] 译者注：由于古希腊字母比较难输入，原文中的希腊文姓名统一用其中的一个字母代替

那么就不会有其他议员的律簿会记录不同内容的第 132 号法令。当然，另一个议员的律簿里可能还没有第 132 号法令的记录，如果他还不知道第 132 号法令已经通过了。

仅仅有律簿的一致性（Consistency）还不够，因为让每个律簿都保留空白也能满足一致性。所以**需要一些要求（Requirement）来保证法令能最终通过并被记录在律簿中**。在现代的议会中，议员达不成一致妨碍了法令的通过。但是在 Paxos 不是这种状况，这里盛行的是**相互信任的气氛，议员愿意通过任何被提交的法令**。但是他们好游历的特性却造成了问题。假如一组议员通过了 37 号法令：

37：禁止在圣殿的墙上种植

然后离开议会厅参加宴会去了，接下来另外一组议员进来议会厅，不知道刚刚发生了什么事情，然后也通过了一个冲突的 37 号法令：

37：允许自由的艺术表达

那么一致性就失去了。

除非足够多的议员在议会厅里呆足够长的时间，否则**进展性(Progress)无法保证**。因为 Paxos 的议员不愿意缩减他们在外面的活动，所以无法保证任何法令都会最终被通过。但是无论如何，议员们愿意保证，只要他们在议会厅中，他们和他们的助手就会快速的处理所有的议会事务。这个保证使 Paxos 公民们能够设计出一个满足如下**进展性条件**（Progress Conditon）的议会协议：

> **如果议员中的多数[3]都在议会厅中，并且在一个足够长的时间内**
> **没有人进出议会厅，那么任一被某个议员提议的法令都将会被通过，**
> **并且每一个被通过的法令都会出现在议会厅中每个议员的律簿上。**

小胡涂：以“议员中的多数”作为前提，能够保证对于任意两次提议，至少存在同一个议员参与了这两次提议。保证了**进展性**。从而不会发生“冲突的 37 号法令”这种情况。

1.3. 假设（Assumptions）

通过提供给议员必要的资源，议会协议的要求(requirements)是可以达到的。每个议员收到了一个结实耐用的律簿来记录法令，一支笔，和擦不掉墨水。议员如果离开过议会厅，可能会在回来后忘记了他们曾经做过什么（（比如）在一个悲剧的事故里，议员 Twvey 被掉下来的雕像击中了头部而永久失忆了~~==！），所以他们会把一些重要的议会任务记在律簿的背面。律簿上的法令条目永远不会改变，但是律簿背面的备注(notes)可能会被划掉。进展条件的达成要求议员能够度量时间的流逝，所以给了他们简单的沙漏计时器。

[3] 在翻译进展条件时，我将 Paxos 的单词 μαδζθωριτϊστ 翻译为议员中的多数。2.2 节中提出了另一个可取的翻译，并做了讨论。

议员任何时候都会带着他们的律簿，并且总是能够从律簿上阅读到法令条目和尚未划掉的备注。律簿由最精良的羊皮纸做成，只用来记录最重要的备注(notes)。其他备注会被记录到小纸条(a slip of paper)上，这些小纸条可能会在议员离开议会厅后丢失。

议会厅里比较嘈杂，影响听觉，不可能在里面做演讲。议员只能通过信使来通信，并且有专款来供议员雇用任意多他们需要的信使。信使不会篡改消息，但是他可能会忘记他递送过了某个消息，并再次递送它。像他们服务的议员一样，信使也只花他们部分的时间在议会职责上。一个信使在投递一个消息前可能会离开议会厅去从事其他的事情，比如一次为期 6 个月的航海。他甚至可能会一去不复返，在这种情况下消息永远也不会被送达[4]。

尽管议员和信使可以随时离开或进入，但是只要他们在议会厅里，他们就会专注于议会事务。只要呆在议会厅，信使会很快速的投递消息，议员会立即快速的处理任何他们收到的消息。

Paxos 的官方记录声称议员和信使绝对的诚实并严格遵循议会协议。大多数学者仅仅把这当做将 Paxos 描绘为在道德上优越于其东方邻国的宣传。不诚实，尽管稀少，但毫无疑问一定是存在的，但由于官方文本里从未提及，我们也不知道议会是怎样应付不诚实的议员或信使的。在 3.3.5 节讨论了已经发现的迹象。

小胡涂：这也是 Paxos 与拜占庭问题的区别。Paxos 假设消息会丢失，但在传输过程中不会被篡改；而拜占庭问题则描述的是将军中存在内奸，消息可能会被篡改。

[4] 信使就好比计算机的网络传输

第二章 The Single-Decree Synod

Paxon 议会由更早期的牧师的宗教仪式会议（Synod，以下简称神会）演化而来。这种神会每 19 年召集一次来选择一个象征性的法令。几个世纪以来，神会约定俗成地要求所有牧师都到场来选择一个法令。但是随着商业的繁荣，牧师开始在神会期间就进进出出会议厅，最后旧的协议失效了，一个神会没有达成任何法令就结束了。为了避免重复这种宗教上的灾难，**Paxos 的宗教领袖请求数学家制定一个协议来达成神会法令**。协议的要求和假定本质上与后来的议会协议是一样的，除了一点不同：**神会只允许在律簿上包含一条法令，而后来的议会可以包含一系列法令**。**神会协议**在本章中描述，**议会协议**在第三章中描述。

数学家通过几个步骤来得到神会协议。首先他们证明了**一个满足特定约束的协议能够保证一致性，并且可以进展下去**[5]。**由这些约束直接得出了一个初级协议。初级协议的一个约束版本提供了能够保证一致性，但不能保证进展性的基本协议（basic protocol）。通过进一步约束基本协议得到了能够满足一致性和进展性要求的完整神会协议(synod protocol)**[6]。

2.1 节描述了这些数学结论。2.2-2.4 节非正式地描述了这些协议。在附录中给出了基本协议的更正式的描述和正确性证明。

2.1 数学结论

神会法令从多轮带编号的表决(ballot)中选择。**一轮表决是对一个单条法令的投票**。在每次表决中，牧师只有两种选择：对这个法令投票（表示赞成这个法令），或不投票（不赞成）[7]。与一轮表决相关联的是一个牧师的集合，称为法定人数集（quorum）。**当且仅当法定人数集中的每个牧师都赞成这个法令时，这轮表决才是成功的**。正式的表述是，一轮表决 B 由下面 4 个要素组成：（除非特别说明，集合指有限集合）

$\mathbf{B_{dec}}$	**一条法令（被投票的那条 decree）**
$\mathbf{B_{qrm}}$	**一个牧师的非空集合（表决的法定人数集 quorum）**
$\mathbf{B_{vot}}$	**一个牧师的集合（所有对法令作出投票（赞成）的牧师）**
$\mathbf{B_{bal}}$	**这轮表决的编号**

说**一轮表决 B 是成功的，当且仅当** $\mathbf{B_{qrm} \subseteq B_{vot}}$，所以一轮成功的投票是每一个法定人数集的成员都投了票的(赞成票，不赞成则不投票)。

---

[5] 详见定理 1 和定理 2

[6] 神会协议的完整的发现历史已不可考。就像现代的计算机科学家，Paxon 的数学家将描述优美的逻辑推演，与算法的实际推导没有任何相似之处。我们知道数学结论（2.1 节的定理 1 和 2）确实是在协议产生之前作出的。当数学家在响应关于协议的请求，试图证明不可能有一个满足要求的协议时，发现了这些数学结论。

[7] 像一些现代国家一样，Paxos 还没有完全掌握雅典民主的本质。

表决的编号从一个无界有序的数字集合中选取。如果 $B'_{bal} > B_{bal}$，则说 B′ 比 B 大。但这并不能表明表决进行的顺序。一个大的表决实际上可以在一个小的表决之前发生。（这里故意将 later 翻译成了大，省得 later 和 before 掺杂不清）。

Paxon 的数学家在**一个由多轮表决构成的集合β**上定义了 **3 个条件**，然后展示了**如果已经进行过的表决满足这些条件，那么一致性会得到保证，进行性也是可能的**。头两个条件比较简单，可以被非正式的描述如下：

> **B1(β)：β中的每一轮表决，都有一个唯一的编号；**
> **B2(β)：β中任意两轮表决的法定人数集中，至少有一个公共的牧师成员；**

第三个条件更复杂一些。在一个 Paxon 人的手稿中包含了如下内容，比较绕，描述了这个条件：

> **B3(β)：对于β中每一轮表决 B，如果 B 的法定人数集中任何一个牧师在β中一个更小轮的表决中投过(赞成)票，那么 B 的法令与所有这些更小轮表决中的最大的那次表决的法令相同。**

图 1 的手稿帮助诠释了这一段隐晦的文本。手稿画出了 5 个牧师 A, B, Γ, Δ, 和 E 的 5 轮投票来阐明条件 B3(β)的含义。5 轮表决组成了集合β，**对于其中的每轮表决，投了票的牧师集合是法定人数集合牧师的子集，投了票的牧师被方框圈起来**。

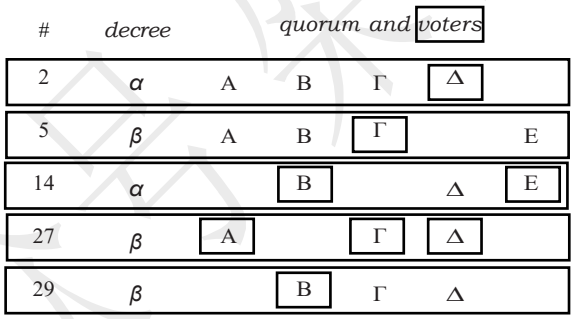

图 1：Paxos 手稿展示了五轮表决组成的集合β，满足条件 B1(β)- B3(β)（加上了说明性的列头）

> **小胡涂：**方框圈起来的投赞成票，未圈起来的投反对票，空白的为缺席。

2. 编号为 2 的表决是最小（早）的表决，从而这轮投票的条件也都满足;
5. 编号为 5 的表决中没有牧师在更小号的投票中投过票，所以这轮的条件也满足;
14. 这轮表决的法定人数集合中，唯一一个在更小编号的表决中投过票的牧师是 Δ，他在编号为 2 的表决中投了票，所以要求这轮的法令和 2 的法令必须相等（成立）
27. （**这是一次成功的表决**）27 轮表决的法定人数集合是 A, Γ, 和 Δ 。牧师 A 没有在更小号的表决中投过票。Γ 投过票的小编号表决只有 5，Δ投过票的小编号表决只有 2；这两个更小编号表决中最大的是 5，所以条件要求本轮的法令和 5 的法令相同。（成立）
29. 本轮的法定人数集合是 B, Γ, 和 Δ。B 投过票的小编号表决只有 14，Γ 是 5 和 27，Δ是 2 和 27。这 4 个小编号表决中最大的一个是 27，所以条件要求 29 的法令和 27 的

法令相同。

正式的表述 B1(β)- B3(β)需要更多的符号标记。我们用**符号 v 表示一个投票**，那么 v 包含 3 个组成部分：**投票的牧师 v_{pst}，本轮表决的编号 v_{bal}，和所表决的法令 v_{dec}**。Paxon 人同时定义了 $v_{bal} = -\infty$ and v_{dec} = blank 的投票 v 为 null 投票。对于 $-\infty < b < \infty$的所有编号为 b 的表决，不会以 BLANK 作为法令。对于任意牧师 p，他们定义了 $null_p$ 作为 v_{pst} = p 唯一的 null 投票 v。

Paxon 数学家为**所有投票定义了一个全局顺序**，但是手稿里包含了这个定义那一部分遗失了，剩下的片段显示，**对于任意 v 和 v′，如果** $v_{bal} < v'_{bal}$，**则** $v < v'$。当 $v_{bal} = v'_{bal}$ 时，v 和 v′ 的相对顺序怎样定义的并不知道。

对于任意的表决组成的集合β，定义**集合 Votes(β)为包含所有满足如下条件的投票 v**：存在 B（**一轮表决**）∈ β 使得 $v_{pst} \in B_{vot}$, $v_{bal} = B_{bal}$，$v_{dec} = B_{dec}$（即用 Votes(β)表示所有在β中的投票，也即赞成票）。

$v_{pst} \in B_{vot}$, $v_{bal} = B_{bal}$，$v_{dec} = B_{dec}$

v ：一个投票
v_{pst}：投票的牧师
v_{bal}：投票所在这轮表决的编号
v_{dec}：投票所在这轮表决的法令

B_{dec}：一条法令（被投票的那条 decree）
B_{vot}：一个牧师的集合（所有对法令作出投票（赞成）的牧师）
B_{bal}：这轮表决的编号

如果 p 是一个牧师，b 是一个表决的编号或正负∞，则 MaxVote(b,p, β)定义为 Votes(β)中由 p 投出的表决编号小于 b（$v_{bal} < b$）的最大的投票 v，如果没有这样的投票则为空投票 $null_p$。因为 $null_p$ 比所有 p 实际投出的票都要小，这就意味着 MaxVote(b,p, β)是下面集合中的最大投票：

$$\{v \in Votes(\beta) : (v_{pst} = p) \wedge (v_{bal} < b)\} \cup \{null_p\}$$

对于任意非空的牧师集合 Q，MaxVote(b,Q, β)定义为对于所有 p∈Q，所有 MaxVote(b,p,β)中的最大值。

条件 B1(β)- B3(β)正式表述如下[8]：

$$\textbf{B1}(\beta) \triangleq \forall B,B' \in \beta : (B \neq B') \Rightarrow (B_{bal} \neq B'_{bal})$$

[8] 我使用 Paxon 数学符号≙，这意味着定义为等于。

B2(β) $\triangleq$ **$\forall$B,B' $\in$ β : $B_{qrm} \cap B'_{qrm} \neq \varnothing$**

B3(β) $\triangleq$ **$\forall$ B $\in$ β : (MaxVote(B_{bal} , B_{qrm}, β)$_{bal} \neq -\infty$) $\Rightarrow$ (B_{dec} = MaxVote(B_{bal} , B_{qrm} , B)$_{dec}$)**

虽然 MaxVote 的定义依赖于投票的顺序，但是 B1(β)表明 MaxVote(b, Q, B)$_{dec}$ 是与表决编号相同的投票间有怎样的顺序无关的。

为了展示这些条件能够导出一致性，Paxon 人首先展示 B1(β)- B3(β)能够得出，如果β中的表决 B 是成功的（$B_{qrm} \subseteq B_{vot}$，即 B 表决的赞成票占大多数/满足法定人数），那么β中更大编号的表决和 B 有相同的法令：

引理：如果保持 B1(β)、B2(β) 和 B3(β)成立，那么对于任意β中的 B, B'有：

(($B_{qrm} \subseteq B_{vot}$) $\wedge$ ($B'_{bal} > B_{bal}$)) $\Rightarrow$ ($B'_{dec} = B_{dec}$)

引理的证明：

对于任意β中的表决 B，令Ψ(B,β)表示β中所有编号大于 B 且法令与 B 不同的所有表决的集合:

Ψ(B,β) $\triangleq$ **{B'$\in$β : ($B'_{bal} > B_{bal}$) $\wedge$ ($B'_{dec} \neq B_{dec}$)}**

证明引理只需证明如果 $B_{qrm} \subseteq B_{vot}$，则Ψ(B,β)为空。Paxon 人给出了一个反证法的证明。他们假设存在 B 使得 $B_{qrm} \subseteq B_{vot}$ 同时 Ψ(B,β) ≠ ∅，然后得到了一个矛盾[9]：

1. 选择 C $\in$ Ψ(B, β)，使得 C_{bal} = min{B'_{bal} : B' $\in$ Ψ(B, β)}
   证明：由Ψ(B, β)不为空并且有限，所以 C 存在得出。

   > **小胡涂：**这里 C_{bal} = min{B'_{bal} : B' $\in$ Ψ(B, β)}可得出，C_{bal} 是Ψ(B, β)中编号最小的表决，当然 C_{bal} 必然还会大于 B_{bal}。

2. $C_{bal} > B_{bal}$
   证明：由 1 和Ψ(B, β)的定义得出。
3. $B_{vot} \cap C_{qrm} \neq \varnothing$
   证明：由 B2(β)和假设 $B_{qrm} \subseteq B_{vot}$ 得出

   > **小胡涂：**有前提条件 $B_{qrm} \subseteq B_{vot}$ 知，B_{vot} 是大多数集，又 C_{qrm} 定义为大多数集，所以 $B_{vot} \cap C_{qrm} \neq \varnothing$

4. MaxVote(C_{bal} , C_{qrm} , β)$_{bal} \geqslant B_{bal}$
   证明：由 2、3 和 MaxVote(C_{bal} , C_{qrm} , β)$_{bal}$ 的定义得出
5. MaxVote(C_{bal} , C_{qrm} , β) $\in$ Votes(β)

[9] 译者注：需要注意的是，这里的证明过程都是基于 C_{qrm} 法定人数集的，即引理成立的基础是 B'也是一轮成功的表决。

证明：由 4 （这意味着 MaxVote(C_{bal} , C_{qrm}, β)不是空投票） 和 MaxVote(C_{bal} , C_{qrm} , β)的定义得出

6. MaxVote(C_{bal} , C_{qrm} , β)$_{dec}$ = C_{dec}
   证明：由 5 和 B3(β) 得出
7. MaxVote(C_{bal} , C_{qrm} , β)$_{dec}$ ≠ B_{dec}
   证明：由 6、1 和Ψ(B,β)的定义得出

> **小胡涂：**由 1 知，C_{bal} 是Ψ(B, β)中编号最小的表决，当然 C_{bal} 必然还会大于 B_{bal}，即 $C_{bal} > B_{bal}$，又有Ψ(B,β)的定义 $C_{dec} \neq B_{dec}$。又因为 6，可得 MaxVote(C_{bal} , C_{qrm} , β)$_{dec}$ ≠ B_{dec}

8. MaxVote(C_{bal} , C_{qrm} , β)$_{bal}$ > B_{bal}
   证明：由 4、7 和 B1(β)表明 MaxVote(C_{bal} , C_{qrm}, β)$_{bal}$ ≠ B_{bal} 得出
9. MaxVote(C_{bal} , C_{qrm} , β) ∈ Votes(Ψ(B, β))
   证明：由 7、8 和Ψ(B,β)的定义得出
10. MaxVote(C_{bal} , C_{qrm} , β)$_{bal}$ < C_{bal}
    证明：由 MaxVote(C_{bal} , C_{qrm}, β)$_{bal}$ 的定义得出
11. 矛盾
    证明：由 9、10 和 1 得出

> **小胡涂：**
> 由 1 知，C_{bal} 是Ψ(B, β)中编号最小的表决；
> 由 10 知，MaxVote(C_{bal} , C_{qrm} , β)$_{bal}$ < C_{bal}；
> 由此得 MaxVote(C_{bal} , C_{qrm} , β)$_{bal}$ 是不存在的。
> 而，又由 9 知，MaxVote(C_{bal} , C_{qrm} , β) ∈ Votes(Ψ(B, β))；
> 所以，这就产生了矛盾。

引理证明结束。

通过这个引理，可以很容易的得出，如果 B1-B3 成立，那么任意两轮成功的表决，都是对相同法令的表决。

定理 1. 如果保持 B1(β)、B2(β) 和 B3(β)成立，那么对于任意β中的 B，B'有：

$$((B_{qrm} \subseteq B_{vot}) \wedge (B'_{qrm} \subseteq B'_{vot})) \Rightarrow (B'_{dec} = B_{dec})$$

定理的证明：

如果 $B'_{bal} = B_{bal}$，则 B1(β)表明 B' = B。如果 $B'_{bal} \neq B_{bal}$，则根据引理直接可以得到定理。

定理证明结束

Poxon 人接下来证明了另一个定理：如果议会厅有足够的牧师，那么在遵守 B1-B3 要求的前提下，是有可能作出一次成功的表决的的（既可能会产生一轮达成一致的表决）。尽管这不能保证进行性(不能保证一定会产生一轮成功表决)，但至少表明在 B1-B3 基础上的表决协议不会产生死锁。

定理 2. 令表决编号 b 和牧师集合 Q 满足对于任意 B ∈ β，$b > B_{bal}$ 同时 $Q \cap B_{qrm} \neq \varnothing$，如果 B1(β)、B2(β) 和 B3(β)条件成立，那么存在一轮表决 B'，$B'_{bal} = b$ 同时 $B'_{qrm} = B'_{vot} = Q$ 使得 B1(β∪{B'}，B2(β∪{B'})，和 B3(β∪{B'})这 3 个条件也成立。（也就是总能找到保持 3 个条件被满足的表决，以使程序进行下去）

定理的证明：

B1(β)、B'bal 的选择（$B'_{bal} = b$），和 b 的假定（对于任意 B ∈ β，$b > B_{bal}$）可以得出 B1(β∪{B'}) 。B2(β)、B'_{qrm} 的选择（$B'_{qrm} = B'_{vot} = Q$）和 Q 的假设（对于任意 B ∈ β， $Q \cap B_{qrm} \neq \varnothing$）可以得出 B2(β∪{B'})。如果 $\text{MaxVote}(b, Q, \beta)_{bal} = -\infty$则让 B'_{dec} 等于任意法令，否则让 B'_{dec} 等于 $\text{MaxVote}(b, Q, \beta)_{dec}$，然后加上 B3(β)，B3(β∪{B'})得以成立。

定理证明结束

2.2 初级协议（The Preliminary Protocol）

Paxon 人**在要求 B1(β)- B3(β)条件保持满足的基础上推演出了初级协议**，β是已经完成或将要完成的所有表决的集合。协议的定义详细描述了集合β是怎样变化的，但这个集合却从来没有被显式的计算过(大小)。Paxos 把β（的大小）认为是一个只有上帝才知道的数量，可能永远不会有任何凡人知道它。

每一轮表决都由一个牧师发起。牧师选择表决的编号，法令和法定人数集合（quorum）。然后法定人数集合中的每个牧师决定是否在本轮表决中投票(投赞成票)。我们可以由保持 B1(β)- B3(β)条件的要求直接得出应该用什么样的规则规定发起者怎样选择表决的编号，法令和法定人数集合，以及牧师怎样投票。

为了使 B1 成立，**每一轮表决都必须获得一个唯一的编号**。通过记住（在律簿上记录备注）他之前发起过的表决，一个牧师可以很容易的避免发起两个编号相同的表决。每个牧师使用不同集合的表决编号，例如，可以简单的用牧师的名称和一个数字组成表决标号，以字典顺序为序：

$$(13, \Gamma) < (13, \Lambda) < (15, \Gamma)$$

因为在 Paxon 字典中 Γ 比 Λ 小。在任何情况下，每个牧师都有一个无限的表决编号集合单独供自己使用。

为了使 B2 成立，表决的法定人数集被选定为集合 M，开始 M 只是简单的包含多数牧师就可以了。过了些时候，发现胖的牧师比瘦的更少行动，并且会花更多的时间在议会厅中。因此 M 被定义为任何包含总体重大于所有牧师总体重一半以上的牧师集合，而不是一个简单的多数。当一些瘦的牧师抱怨这样不公平时，实际的体重被基于牧师们出勤记录的抽象体重取代。M 的主要**要求是任意两个包含 M 的子集至少包含一个相同的牧师。为了保持 B2 成立，牧师发起一个表决 B，选择 B_{qrm} 作为多数集合**。

条件 B3 要求**如果 MaxVote(b, Q, β)$_{dec}$ 不为空集，那么编号为 b，法定人数集为 Q 的表决，表决法令必须为 MaxVote(b, Q, β)$_{dec}$；如果 MaxVote(b, Q, β)$_{dec}$ 为空集，那么表决法令可以是任意法令**。为了保证 B3(β)成立，一个牧师 p 在发起表决前，需要找出 MaxVote(b, Q, β)$_{dec}$，为了找出 MaxVote(b, Q, β)$_{dec}$，p 需要对 Q 中的每一个 q 找出 MaxVote(b, q, β)。

回顾一下，MaxVote(b,q,β)是 Votes(β)中由 q 投出的所有表决编号小于 b(v_{bal} < b)的最大的投票 v 或者空投票 $null_q$（如果 q 在小于 b 的所有表决中都没有投票）。牧师 p 通过一些消息交换从 q 处知道 MaxVote(b,q,β)。因此为了构建由 p 发起的单一轮投票，头两步应该是：

(1) 牧师 p 选择一个新的表决编号 b，发送一条 NextBallot(b)消息给某些牧师。
(2) 牧师 q 响应消息 NextBallot(b)，发送 LastVote(b,v)给 p，v 是 q 的编号小于 b 的表决中编号最大的表决的投票，或者如果 q 没有在任何编号小于 b 的投票中表决过，则为 $null_q$

当 q 发送 LastVote(b,v)时，v 等于 MaxVote(b, q, β)，但是新表决的发起和投票的进行会改变集合β。因为当选择法令时，牧师 p 将会使用 v 作为 MaxVote(b, q, β)的值，为了保持 B3(β)成立，务必要使 MaxVote(b, q, β)在 q 发送 LastVote(b,v) 消息后不再改变。**为了让 MaxVote(b, q, β)保持不变，q 必须在编号介于 v_{bal} 和 b 之间的表决中不再投票**。（**为了保证这个承诺，q 必须在他的律簿中记录一些必要的信息**[10]）。在表决协议中接下来的两个步骤是：

(3) 在**收到某个过半集合 Q 中所有牧师的 LastVote(b,v)消息**后，牧师 p 发起一个编号为 b，法定人数集合为 Q，法令为 d 的新表决。**法令 d 的选择满足条件 B3(β)**。接着他*在他的律簿背面记下这个表决*，然后给 Q 中的每一个牧师发送 BeginBallot(b,d)消息。
(4) 在收到 BeginBallot(b,d)消息后，牧师 q 决定要不要对这个编号为 b 的表决投出他的一票。（如果他已经在其他表决中发送了 LastVote(b', v') 消息，而投这一票会违背这个消息隐含的承诺，那么他不会投出这一票[11]）如果 q 决定为表决 b 投票，那么他发送一个 Voted(b,q)消息给 p 然后*在他律簿背面记录下这个投票*。

步骤(3)的执行可以认为是将一个表决 B 加入到集合β中，其中 B_{bal} = b, B_{qrm} = Q, B_{vot} = ∅（还没有人在这轮表决中投票），B_{dec} = d。在步骤(4)中，如果牧师 q 决定在表决中投票，那么执行这一步骤可以认为是通过将 q 加入到 B 的投票者集合 B_{vot} 中来改变了表决集合β，这里 B ∈ β。

一个牧师有在步骤(4)中不投票的选择，即使投了这票不会违背任何之前的承诺。事实上这个协议中的所有步骤都是可选的。例如，一个牧师 q 可以忽略 NextBallot(b)消息，不执行步骤(2)。采取某个动作失败会阻止流程的继续，但不会造成任何不一致，因为这不能使 B1(β)-B3(β)条件失效。。消息丢失同样不会造成不一致，因为没有收到一条消息唯一能有的影响就是阻止某些动作的发生。因此，协议保证了一致性，即使在牧师离开了议会厅或

[10] 步骤(4)中会提到如果 q 决定为表决 b 投票，那么他发送一个 Voted(b,q)消息给 p 然后在他律簿背面记录下这个投票。
[11] 他可能会因为这样做会违背他已经在其他表决中发送的 LastVote(b', v')消息隐含的承诺而不会投这一票

者消息丢失的情况下。

收到多个消息拷贝会导致某些动作被重复执行。除了步骤(3)之外，第二次执行一个动作没有任何影响。例如在步骤(4)中发送 Voted(b,q)多次和只发送一次效果是一样的。对于步骤(3)，可以通过在每次执行时在律簿背面记录下一个标志的方式来避免重复执行。这样即使在同一个消息会被递送多次的情况下，也能保证一致性条件了。

步骤(1)- (4)描述了发起一个表决和对其投票的完整协议。剩下来的事情就是决定表决的结果和宣布新法令的产生。回顾下，当且仅当法定人数集中的所有牧师都投票通过时一个表决才算成功，一个成功表决通过的法令，就是被神权议会选择的法令。

协议的剩余部分是：

(5) **如果 p 收到了 Q（编号为 b 的表决的法定人数集）中每个 q 的 Voted(b,q)消息，**那么他*在自己的律簿上记下 d（这次表决的记录）*，然后发送 Success(d)消息给每个牧师

(6) 一个牧师在收到 Success(d)消息后，将法令 d 记录到他的律簿上。

步骤(1)- (6)描述了单个表决如何达成。议会协议允许任何一个牧师在任何时间发起一个表决。每一个步骤都保证 B1-B3 条件，因此整个协议也保证这些条件。因为一个牧师**只有在一个法令是某个成功表决的法令时，才在律簿上记录下它，所以根据定理 1，牧师们的律簿是能够保持一致的。协议并没有解决可进展性问题**。（即只保证了一致性，而没有保证流程一定能进行下去）

在步骤(3)中，如果法令是有条件 B3 决定的，那么有可能这个法令已经被写在某个牧师的律簿上了。那个牧师不需要在法定人数集中：他可能已经离开了议会厅。这样的话，如果步骤(3)在选择 d 时允许任何更大的自由（更大的表决编号），那么一致性将不能保证。

2.3 基本协议（The Basic Protocol）

在上文的初级协议中，一个牧师必须记录(i) 他发起的每一个表决的编号，(ii) 他的每个投票，和(iii) 他发出的每一个 LastVote 消息。保持跟踪记录所有这些信息对于那些繁忙的牧师来说是一件困难的事情。Paxon 人因此**对初级协议作了进一步约束，得到一个更具实用性的基本协议**。在基本协议中，每个牧师 p 必须且只需在他的律簿后面记录如下信息：

① lastTried[p] 由 p 试图发起的最后一个表决的编号，如果没有发起过则记录-∞

② nextBal[p] 由 p 发出的所有 LastVote(b,v)消息中，表决编号 b 的最大值。

③ prevVote[p] 由 p 投票的所有表决中，编号最大的表决对应的 p 的投票，如果没有投过票则记录-∞

初级协议的步骤 1-6 描述了一个单一的表决怎样通过他的发起者牧师 p 指挥。**初级协议允许 p 并行的指挥任意数量的表决。在基本协议中，他一个时间只指挥一个表决—**

编号为 lastTried[p]的表决。在 p 发起这个表决后，他忽略与他先前发起的任何其他表决相关的消息。牧师 p 在一个小纸片上保留编号为 lastTried[p]的表决的所有过程信息。如果他丢失了这个小纸片，他就对这轮表决停止指挥[12]。

在初级协议中，牧师 q 发出的每一个 LastVote(b,v)消息意味着承诺不再对任何编号介于 v_{bal} 和 b 之间的表决投票。**在基本协议中，LastVote(b,v)消息意味着更强的承诺：不再对任何编号小于 b 的表决投票。**这个更强的承诺可能会阻止一个原本在初级协议下是允许的投票。但是介于初级协议总是会给 p 不投票的选择，所以基本协议不会要求 p 作出任何违反初级协议的事情。

初级协议的 1-6 个步骤在基本协议中变为如下 6 步：

(1) 牧师 p 选择一个比 lastTried[p]大的表决编号 b，设置 lastTried[p]为 b，然后发送 NextBallot(b)消息给某些牧师。
(2) 在从 p 收到一个 b 大于 nextBal[q]的 NextBallot(b)消息后，牧师 q 将 nextBal[q]设置为 b，然后发送一个 LastVote(b,v)消息给 p，其中 v 等于 prevVote[q]。（b<= nextBal[q]的 NextBallot(b)消息将被忽略）
(3) 在从某个多数集合 Q 中的每个成员都收到一个 LastVote(b,v)消息后，牧师 p 发起一个编号为 b，法定人数集为 Q，法令为 d 的新表决，其中 d 的选择遵守 B3 条件。然后他发送一个 BeginBallot(b,d)消息给 Q 中的每一个牧师
(4) 在收到一个 b= nextBal[q]的 BeginBallot(b,d)消息后，牧师 q 在编号为 b 的表决中投出他的一票，设置 prevVote[q]为这一票，然后发送 Voted(b,q)消息给 p（b!= nextBal[q]的 BeginBallot(b,d)消息将被忽略）
(5) 在 p 收到 Q 中每一个 q 的 Voted(b,q)消息后（这里 Q 是编号为 b 的表决的法定人数集合，b=lastTried[p]），他将 d（这轮表决的法令）记录到他的律簿上，然后发送一条 Success(d)消息给每个 q
(6) 一个牧师在接收到 Success(d)消息后，将法令 d 写到他的律簿上。

基本协议是初级协议的一个受约束版本，意味着基本协议允许的任何行为在初级协议中都是允许的。初级协议满足一致性条件，那么基本协议也满足一致性条件。像初级协议一样，基本协议并未要求任何行为最终会被执行，因此它同样没有解决进展性（progress）问题。

从 B1-B3 到基本协议的演化过程可以显而易见的看出一致性条件是满足的。但是，某些同样给人感觉“显而易见”的古老智慧已经被认为是错误的了，因此具有怀疑精神的公民们要求一个更严格的证明。附录里重现了 Paxon 的数学家们对协议满足一致性条件的证明。

2.4 完整的神会协议（The Complete Synod Protocol）

基本协议维护了一致性，但是没有保证任何可进展性，因为它仅仅规定了一个牧师可以怎么做；它没有要求他一定要做任何事情。完整的神会协议和基本协议一样包含 6 个步骤来操控一个表决。为了促成可进展性，它很直观地要求牧师们尽可能快地执行协议的 2-6

[12] 译者注：就好比计算机内存，程序崩溃，内存数据丢失

步骤。**为了满足可进展性条件，必须要求某些牧师来执行步骤 1 来发起一轮表决**。完整协议的关键一点是决定什么时候一个牧师应该发起一轮表决。

永远不发起表决肯定会阻止进展性。但是发起太多的表决同样会阻止进展性。假设 b 比任何其他表决的编号都大，牧师 q 在步骤 2 中收到一条 NextBallot(b)消息会引出一个承诺，阻止他在步骤 4 中对任何之前发起的表决投票。这样，**一轮新表决的发起会阻止任何之前发起的表决成功。如果新的表决以不断增大的编号在之前的表决有机会成功之前不断地被发起，那么整个（会议）将会没有任何进展。**

达到可进展性条件要求新的表决直到某个表决成功之后才发起，但是不被频繁地发起。为了开发完整协议，Paxon 人首先必须知道信使传递消息和牧师响应要花费多长时间。他们测定了一个没有离开议会厅的信使，总会在 4 分钟之内送出消息，一个呆在议会厅中的牧师，总会在 7 分钟内处理消息。这样如果 p 和 q 在议会厅中，这时某些事件使得 p 发送一条消息给 q，并且 q 响应了一个答复给 p，那么在没有任何信使离开议会厅的情况下，p 将在 22 分钟之内收到这个答复。（牧师 p 会在事件的 7 分钟内发出消息，q 将会在接下来的 4 分钟内收到这个消息，然后在 7 分钟内作出响应，答复会在接下来的 4 分钟内送达 p ）

假设只有一个牧师在发起表决，他在协议的步骤 1 中给每个牧师发送一条消息。如果 p 在多数牧师呆在议会厅中时发起表决，那么他可以期望在 22 分钟之内执行步骤 3，在另一个 22 分钟之内执行步骤 5。如果他没有能够在这些时间内执行这些步骤，那么或者某些牧师或信使在 p 发起这个表决后离开了议会厅，或者一个更大编号的表决已经在之前由另一个牧师发起了(在 p 成为唯一的表决发起人之前)。**为了解决后一个可能性，p 必须学习其他牧师使用的任何编号比 lastTried[p]大的表决。可以通过扩展协议以要求牧师 q 收到 p 的 NextBallot(b) 和 BeginBallot(b,d)消息时，如果发现 b < nextBal[q]，则将 nextBal[q]发送给 p 来完成这个学习。这样牧师 p 就可以发起一个更大编号的新的表决。**

仍然**假设 p 是唯一发起表决的牧师，假定他被要求当且仅当如下条件满足时，才发起一个新表决:(i)在最近的 22 分钟之内他没有执行步骤 3 或步骤 5，或者(ii)他得知另一个牧师已经发起了一个编号更大的表决（上面提到的）。**如果议会厅的门在 p 和过半的牧师在里面时锁住了，那么一条法令会在 99 分钟内通过并记录到议会厅中每个牧师的律簿上。(22 分钟牧师 p 启动下一个表决[13]，22 分钟得知另一个牧师发起过更大编号的表决[14]，55 分钟完成一个成功表决的步骤 1-6)。这样，在只有一个牧师发起表决，并且不会离开议会厅的前提下，可进展性条件将会被满足。

完整协议因此包含一个流程来选择一个唯一的牧师，称作总统，来发起表决。在多数形式的政府中，选择一个总统会是一个困难的问题。但是这种困难仅仅是因为大多数政府要求任意时候都必须有一个唯一的总统存在。例如在美国，如果某些人认为 Bush 已经当选为总统，而这时另一些人认为是 Dukakis，就会造成混乱，因为其中的某个可能会决定签署某个法案而同时另一个决定否决这个法案。但是**在 Paxos 的神会中，拥有多个总统只会阻止进**

[13] 译者注：由『(i)在最近的 22 分钟之内他没有执行步骤 3 或步骤 5 才发起一个新表决』，可知牧师 p 启动下一个表决需要 22 分钟

[14] 译者注：由『一个更大编号的表决已经在之前由另一个牧师发起了(在 p 成为唯一的表决发起人之前)』，可知得知另一个牧师发起过更大编号的表决需要 22 分钟

展性，而不会导致不一致。为了完整协议满足可进展性条件，选择总统的方法只需要满足以下“总统选举要求”：

> **如果没有人进入或离开议会厅，那么接下来的 T 分钟之内只会有一个议会厅中的牧师会认为他自己是总统。**

如果满足了总统选举要求，那么完整协议将具有如下特性：如果多数的议员在议会厅内，并且在 T + 99[15] 分钟内没有人进出议会厅，那么在这段时间的最后，议会厅中的每个牧师的律簿上都会记下一条法令。

Paxon 人将议会厅中名字在字典顺序中排在最后的牧师选为总统[16]，尽管我们不知道这个具体是如何进行的。**如果一个牧师每隔 T - 11[17] 分钟将自己的名字发给所有其他的牧师，并且一个牧师当且仅当在 T 分钟内没有收到“更高的名字”时才认为自己是总统，那么总统选举要求就可以满足**。

通过在基本协议的基础上，要求牧师们及时的处理步骤 2-6，增加一个选择一个总统来发起表决的方法，并且要求总统在适当的时候发起表决，我们就得到了完整的协议。很多协议的细节还尚不清楚。我描述了选择总统和决定总统在什么时候应该发起表决的简单方法，但是这些无疑不是 Paxos 使用的方法。**我给出的要求总统即使在一个法令通过后仍然持续发起表决的规则[18]，保证了刚进入议会厅的牧师能够学习到被通过的法令。**一定有更好的方法保证牧师们能够学习到已经通过的法令。同样的，在选择总统的过程中，每一个牧师适当的发送他自己的 lastTried[p]值到其他牧师，允许总统在他的第一次尝试中选择足够大的表决号。

Paxon 人认识到任何想达成可进展性条件的协议都会涉及到对时间流逝的度量。上文给出的选择总统和发起表决的协议，可以很容易的构建为精确的算法：设置计时器并在超时发生时采取动作——假设是完全精确的计时器。一个近似的分析显示在计时器不完全精确，但是有一个已知的精度范围时，这样的协议也可以达成。Paxon 有技艺的沙漏师傅可以毫不困难的制造出合适的沙漏计时器。

给出 Paxos 数学家的精密证明后，他们普遍相信已经找到了一个最优的满足总统选择要求的算法。我们只能希望这个算法会在将来被发现。

---

[15] 译者注：T 分钟用于选举出唯一一个总统，同时一条法令会在 99 分钟内通过并记录到议会厅中每个牧师的律簿上

[16] 译者注：神会协议的总统选举策略，后面 3.3.1 会提到议会协议的总统选举策略

[17] 译者注：T - 11 分钟是 T 减去当前牧师处理消息的时间(7 分钟)和信使送出消息的时间(4 分钟)

[18] 译者注：持续发起针对同一法令的表决

第三章 The Multi-Decree Parliament

当议会建立之后，一个满足一致性和进展性的协议从神会协议演化而来。演化过程和最初议会协议的特性在 3.1 和 3.2 节描述。3.3 节讨论了这个协议未来的演化。

3.1 协议

与仅仅通过一项法令不同的是，Paxon 议会需要通过一系列编号的法令[19]。像在神会协议中一样，议会协议也要选举总统。任何人想通过一条法令都会通知总统，总统会为法令赋予一个编号，然后尝试通过它。顺理成章地，议会协议为每个法令编号使用不同实例（instance）的完整神会协议。但是会选择一个单独的总统负责所有这些实例，并且他对协议的前两个步骤只会执行一次。

得到议会协议的关键是观察到：在神会协议中，总统直到步骤 3 之前不会选择法令或法定人数集。**一个新选举出的总统 p 可以向某些议员发送一条单个的信息，作为神会协议所有实例的 NextBallot(b)消息。（有无数个实例 - 每个法令编号对应一个实例。）一个议员 q 可以使用单个消息进行回复，该消息充当神会协议的所有实例的步骤 2 的 LastVote 消息。此消息仅包含有限数量的信息，因为 q 只能在有限数量的实例中投票。**

当新的总统收到了一个多数集合中每个成员的回复后，他就准备好了为神会协议的每个实例执行第 3 步。对于某些有限个数的实例（法令编号），步骤 3 中法令的选择由 B3 条件决定。总统立即为所有这些实例执行步骤 3，以尝试让这些法令通过。然后，每当他收到通过一个法令的请求时，他选择他仍然可以选择的法令中编号最小的法令，然后为这个法令编号（神会协议的实例）执行步骤 3，以尝试通过这个法令。

对这个简单协议经过以下修改，得到真正的 Paxos 议会协议：

- **没有理由对已经知道结果的提案仍然走一遍神会协议流程[20]。**因此，**如果一个新当选的总统，在他的律簿上已经记录有所有编号小于等于 n 的法令，他发送一个 NextBallot(b, n)消息，该消息在神会协议的所有实例中用作大于 n 的法令编号的 NextBallot(b)消息。在这个消息的回复中，议员 q 将他的律簿中（已经出现的）所有编号大于 n 的法令通知给 p（除了发送通常的 LastVote 信息，用于不在他的账本中的法令），并且他告诉 p 将他律簿中没有的任何编号小于等于 n 的法令发给他。**
- [21]设想法令 125 和 126 在周五下午的晚些时候引入，126 通过并记录在了 1、2 个议员的律簿上，之后没有发生任何其他事情，所有的议员就都回家过周末了。接着设想在接下来的星期一，**Δ 被选举为新的总统，并学习关于法令 126 的事情**，但是他不知道关于法令 125 的事情，因为之前的总统和所有投了票的议员们都还没

[19] 译者注：所以下面我们都会称一个法令编号对应一个神会协议的实例。
[20] 译者注：前面我们提到：神会协议是通过总统即使在一个法令通过后仍然持续发起表决的规则，来保证了刚进入议会厅的牧师能够学习到被通过的法令。
[21] 译者注：『3.2.1 法令的顺序』一节会详细说明如何解决这一问题。

有回到议会厅。她将**举行一个让法令 126 通过的表决[22]，这在她的律簿上就留下了一个空隙。赋给新的法令编号 125 会使它在律簿上出现在 126 之前，而 126 上周就通过了。在这种场景下倒叙地通过法令可能会造成混乱**——例如，如果出新法令的公民是因为知道 126 已经通过了才出的。**取而代之的，Δ 将会试图通过:**

125：2 月是全国橄榄节

一个对 Paxon 的每个人都绝对没有分别的传统法令。通常，新的总统会通过举行类似“橄榄节”这样的法令表决来填补他律簿上的任何空隙。

议会协议的一致性和进展性直接从神会协议继承而来。就我们所知，Paxon 人会很容易的写出一个对议会协议的更精确的描述，因为它是如此简单的从神会协议演化而来。

3.2 协议特性 (Properties of the Protocol)

3.2.1 法令的顺序

多个不同的法令编号可以并发的进行表决，表决可以由不同的议员发起——每个议员在发起表决时都将他自己当做总统。我们不能确切的说出法令是以怎样的顺序通过的，特别在不知道总统是怎样选择出来的时候。但是，有一个关于法令顺序的重要属性可以被推导出来。

当总统在相应的神会协议实例的步骤 3 中选择它时，一项法令被称为已提议的(proposed)。当一项法令被第一次写入到某个律簿中的时候，这个法令被称为已通过的(passed)。**在一个总统可以议任何新的法令之前，他必须向一个多数(过半的)集合中的每个议员学习他们已经投过票的法令。任何已经通过的法令一定被至少一个多数集合中的议员投过票。这样，总统在发起一个新的法令前，就可以学习到所有之前已经通过的法令。总统将不会用重要的法令来填补之前提到的律簿上的空隙**——也就是说，除了“橄榄节”法令之外的任何法令。**他也不会错序的提议法令**。因此，**协议满足以下法令顺序特性：**

> **如果法令 A 和 B 是重要的，并且法令 A 在法令 B 提议之前就已经通过了，那么 A 的法令编号比 B 小。**

3.2.2 关上门之后

尽管我们不知道选择一个新总统涉及的具体细节，但我们确切地知道选择了新的总统之后， 没有人进出议会厅时，议会是怎样运作的。在收到一个通过一个法令的请求后——或者直接通过公民，或者有其他议会转发来——总统为法令赋予一个编号，然后通过如下的消息交换来通过它(序号与神会协议中的序号对应):

(3)总统发送一个 BeginBallot 消息到某个法定人数集中的每个议员

[22] 译者注：选举出来的新总统学习新的法令的方式是发起相应新法令的表决。

(4)法定人数集中的每个议员发送 Voted 消息给总统

(5)总统发送 Success 消息给每个议员

假设议会有 N 个议员，法定人数集大约 N/2，那么这里总共会有 3 个消息延迟和 3N 个消息。

此外，如果议会繁忙，总统可以将一个法令的 BeginBallot 和上一个法令的 Success 消息合并起来发送，这样每个法令就只有 2N 个消息了。

3.3. 更进一步(Further Development)

管理这个小岛渐渐变成了一个比 Paxon 曾经认为的更复杂的任务。很多问题涌现出来。解决这些问题需要修改协议。最重要的几个修改描述如下:

3.3.1 选择一个总统

议会总统的选举最初使用神会协议中的方法，仅仅建立在名字的字典顺序基础上。这样当议员Ω(名字在希腊字母的字典顺序中最大)在 6 个月的休假中回来之后，他就立即当选为总统了，即使他完全不知道他不在的时候发生了什么。**碰巧 Ω 又是个写字很慢的人，这样当他费力地将 6 个月的法令补齐到他的律簿上时，整个议会的活动都停下来了。**

这个事故导致了关于选择总统的最佳方法的争论。某些 Paxon 人主张一旦一个议员当选为总 统，他应该一直保持是总统，直到他离开议会厅。一个部分有影响力公民希望让最富有的议员当选总统，因为他有能力雇佣更多的助手帮助他履行总统职责。他们主张一旦一个富有的议员将它的律簿更新到了最新，他将没有理由不承担总统职位。其他一些人，主张应该让最正直的公民担任议会总统而不考虑财富的多少。正直大抵意味着不太会不诚实，尽管没有 Paxon 人公开承认有官方渎职的可能。**很不幸的，这些争论的最终产出还不知道。没有关于最终使用的总统选择协议的记录。**

3.3.2 长律簿

随着一年年的过去，议会通过了越来越多的法令，Paxon 公民需要仔细阅读很长的法令清单，才能知道现在的橄榄赋税是多少，或什么颜色的山羊可以出售。一个议员在一次长期航海之旅后回到议会厅时，需要做相当多的抄写工作来将他的律簿更新到最新。事实上，议员们不得不**将律簿上的法令清单转变为法律书。法律书包含法律的当前状态，和最后一条通过的反应在当前状态中的法令编号。**

为了知道当前的橄榄赋税，可以在法律书中的“赋税”这一节查阅;为了知道那种颜色的山羊可以出售，可以在“贸易法”下面查阅。**如果一个议员的律簿包含了直到法令 1298 的法律，然后他学习到法令 1299 将橄榄的赋税设为了每吨 6 银币，他只修改橄榄赋税这个条目，并且记下他的律簿已经包含了直到 1299 的法令。如果之后他学习了法令 1302，他将会在律簿的背面记下 1302，然后等待他学习完 1300 和 1301 法令，再将它们一起合并写入律簿中。**

为了使一个短期离开的议员不用拷贝整本法律书而追上最新的状态，议员们在法律书的背面保持一份过去几个星期的法令列表。他们可能已经在小纸条上记下了这样的列表，但是将法令输入到律簿的背面会更方便一些，因为他们 2、3 个星期才通过一次法令并更新律簿。

3.3.3 官员化

随着 Paxon 的兴旺，议员变得非常忙。议会不再能处理政府的所有细节问题，所以一个政府机构建立了。议会不再是通过一条法令来声明是否每块奶酪符合销售标准，而是通过一条法令指定一个奶酪检查员来做出这些决定。

大家很快发觉选择官员并不像刚开始看起来那么简单。议会通过了一个法令让 Δ 成为第一个奶酪检查员。几个月之后，商人们投诉 Δ 太严格了，拒绝了完美的好奶酪。所以议会通过如下法令替换了他:

1375：Γ 是新的奶酪检查员

但是 Δ 没有密切关注议会的活动，因此没有立即学习到这个新法令。这样在奶酪市场上就有了一个混乱的时期，两个新旧检查员同时检查奶酪并且做出互相冲突的决定。

为了避免这种混乱，Paxon 人必须保证任何一个时刻最多只有一个官员。为了达到这个目的，总统在提议法令时，为每个法令加入了一个日期时间的部分。一个让 Δ 成为奶酪检查员的法令可能像这样:

2716：72 年 1 月 15 日 8:30 —— Δ 开始 3 个月任期的奶酪检查员

这就定义了 Δ 的任期从 72 年 1 月 15 日 8:30 开始，前一个检查员的任期到这个时间结束。他的任期将在 3 月 15 日 8:30 结束，除非他显式地通过要求总统通过如下法令来辞职:

2834：72 年 3 月 3 日 9:15—— Δ 辞去奶酪检查员职务

一个官员以短的任期任命，可以及时的被替换，例如，如果他离开了小岛。议会可以通过一个法令来延长一个官员的任期，如果他比较称职的话。

一个官员需要知道当前的时间，以确定他现在是不是占有某个职位。在 Paxos 还没有机械钟表，但是人们可以通过太阳或星星的位置确定误差在 15 分钟之内的时间。如果 ΔΥκστρα 的任期从 8:30 开始，那么在他通过观察天空得出时间在 8:45 之前不会开始检查奶酪。

很容易将这种任命官员的方法在法令中实行，只要将更大编号的法令推迟 15 分钟提议。但是假如有如下两条法令:

2854：78 年 4 月 9 日 9:45 —— Φ 赢得 2 个月任期的品酒师职位

2855：78 年 4 月 9 日 9:20 —— Π 赢得 1 个月任期的品酒师职位

如果这两条法令在 9:30 至 9:35 之间被两个各自都认为自己是总统的议员分别提议，会发生什么事情呢?这种提议时间顺序颠倒的可以很容易的避免，因为**议会协议满足如下特性**:

> **如果两个法令被不同的总统通过，那么其中一个总统在得知另一个法令提议之后才会提议他的法令。**

我们来看看这个特性是否满足要求。假设表决 b 通过了法令 D，表决 b' 通过了法令 D'，b < b'， **假设议员 q 在两个表决中都投了票。**法令 D' 的表决以一个 NextBallot(b',n)消息开始。**如果这个消息的发送者还不知道法令 D，那么 n 比法令 D 的编号要小，从而 q 对 NextBallot 消息的答复必须说明他已经对 D 投了票。**

3.3.4 法律的学习

除了请求通过法令之外，普通的公民需要获知岛上的当前法律。Paxon 人开始想一个公民可以简单的查询任意一个议员的律簿，但是下面的事故显示出需要一个更严格的途径来完成查询。几个世纪以来只出售白色的山羊是合法的。一个叫 ω 的农民让议会通过了一条法令:

77：允许出售黑色的山羊

然后 ω 指示他的牧羊人将一些黑色的山羊卖给了一个叫 Σ 的商人。作为一个守法公民，Σ 向议员 T 询问这样的出售是否是允许的。但是 T 最近一段时间一直不在议会厅，并且他的律簿上没有法令 76 通过的记录。他告知 Σ 这种出售是当前法律不允许的，因此 Σ 拒绝了购买这批山羊。

这个事故导致法律查询上的如下单调一致性条件的建立：

> **如果一个查询先于第二个查询，那么第二个查询得到的法律的状态不能比第一个得到的更旧。**

如果一个公民得知一个特定的法令通过了，那么得知这一信息的过程被认为是遵守这个条件的隐含查询。我们将会看到，对单调性条件的解释在若干年后发生了变化。

刚开始，单调性条件由为每一个查询通过一条法令来达成。如果 Σδ 想知道当前的橄榄赋税， 他将让议会通过这样一条法令：

87：公民 Σδ 在阅读法律

然后他阅读任何一个完整到包含法令 86 律簿，来得知当前的橄榄赋税。如果公民 Γ 接下来查询橄榄赋税，这个查询的法令在法令 87 通过之后被提议，因此法令顺序特性(3.2.1 节)表明他接收到了一个比 87 更大的法令编号，因此 Γ 不会接收到比 Σδ 更早

的橄榄赋税值。

当“先于”被解释为，查询 A 先于查询 B，当且仅当查询 A 在查询 B 开始之前结束时，上面的这种方法满足了法律查询的单调性条件。

为每个查询通过一个法令很快被证明太笨重了。Paxon 人认识到如果他们通过改变对“先于”的解析而弱化单调性条件，那么可能会有更简单的方法。他们决定对于一个事件先于另一个 事件，第一个事件不仅仅要在更早的时间发生，还要是影响第二个事件发生的条件。弱化的单调性条件可以阻止第一次由农民 ω 和商人 Σ 遇到的问题发生，因为在 ω 的显式查询结束和 Σ 的查询开始之间有了一个事件的因果链。

弱化的单调性条件可以通过在所有的贸易交易或法律查询中使用法令编号来满足。例如，农民 ω 的畜群中有一些不是白色的山羊，他到议会去通过法令：

277：允许出售棕色的山羊

当把他的棕色的山羊卖给 Σ 时，他告诉 Σ 这种出售是被法令 277 允许的。 Σ 询问议员 T 这种出售至少到法令 277 是否是合法的。**如果 T 的律簿没有完整到包含法令 277，他将或者等待律簿完整，或者告诉 T 去询问其他议员。**如果 T 的律簿一直记录到了法令 298，那么他将告诉 Σ 这种出售直到 298 都是合法的。商人 Σ 将记住 298 这个编号，以便在接下来的贸易交易或法律查询中使用。

Paxon 人满足了单调性条件，但是普通的公民不喜欢去记法令编号。Paxon 人再一次通过重新解释单调性的含义来解决这个问题——这一次，**通过改变“法令的状态”的含义。他们把法律分成了几个领域，并为每个领域选择一个议员作为专家。法律每一个领域的当前状态由这个领域的专家的律簿决定。**例如，假设法令 1517 修改了关税法律，法令 1518 修改了赋税法律。**如果赋税法专家在关税法专家之前学习了这两个法令，那么赋税法将先修改，法律的状态就不会一定按法令编号通过的顺序被获取了。**

为了避免对当前状态的定义冲突，Paxon 人规定任一时间任何一个领域最多只能有一个专家。使用和选择官员(3.3.3 节)同样的方法来选择专家可以满足这个要求。如果每个查询只涉及法律的一个领域，将查询引导到那个领域的专家，专家通过它的律簿就可以回复，一致性要求就达到了。注意到一个法律的通过构成了一个隐式的查询，Paxon 人**要求一个法令最多只改变法律的一个领域，并且法令的通过只能由那个领域的专家来通知**。

涉及多个领域的查询也不难处理。当商人 Σ 询问进口羊毛的关税是否比本地的交易税高的时 候，赋税法专家和关税法专家需要协同合作才能给出一个回答。例如，赋税专家可以先询问关税专家山羊毛的关税，直到收到回复前不修改自己的律簿。

这个方法被证明是符合要求的，直到必须对法律的多个领域同时作大面积的修改时。然后 Paxon 人认识到维护单调性的必要条件不是一条法令只影响一个领域，而是它影响的所有领域拥有一个相同的专家。议会可以通过一个法令修改法律的几个领域，只要先任命一个单独的议员作为所有这些领域的专家。更进一步，同一个领域可以有多个专家，只要这个领域的法律不允许修改。在收取赋税之前，议会将会任命多个赋税法专家来应对赋税

法资讯的季节性洪流。

3.3.5 不诚实的议员和无心之过

尽管与官方宣称的相反，在 Paxon 的历史上可能还是有过少数不诚实的议员。他们被抓到后可能就被放逐了。通过发送矛盾的消息，一个恶意的议员可以使其他议员的律簿产生不一致。诚实的议员或信使也可能因为记忆的缺失而导致不一致。

当不一致被识别出时，可以很容易的通过新的法令来修复。例如，橄榄赋税的不一致，可以由通过一个新的法令定义橄榄税为一个确定的值来消除。**修复不一致的律簿的困难在于甚至没有人发觉这个不一致。**

从议会建立几年之后，开始出现在律簿中的冗余法令记录，可以推导出不诚实的议员或议员失误的存在。例如，法令:

2605：橄榄税为每吨 9 银币

通过了，即使法令 2155 已经将橄榄的赋税设置成了每吨 9 银币，并且在这之间没有其他法令修改过它。**议会看起来每 6 个月循环地过一遍他们的法律，以使议员的律簿即使开始存在不一致，所有议员也将在 6 个月内对岛上的当前法律达成一致。Paxon 人相信通过这些冗余法令的使用，可以使他们的议会自稳定(self-stabilizing)。**(self-stabilizing 是 Dijkstra 于 1974 年提出的一个现代术语)

在议会随意的进出的情况下，尚不清楚一个议会自稳定的确切含义。Paxon 人是不会满足一个要求所有议员在某个时间全部坐在议会厅中来保证一致性的定义的。无论如何，**一致性要求当一个议员在律簿中有某个确定法令编号的条目而另一个议员没有时，后者会最终填入这个条目。**

很不幸的，我们没有确切地知道 Paxos 议会运转的自稳定性是哪一种，或者它是怎样达到的。Paxos 的数学家毫无疑问解决了这个问题，但是他们的工作还没有被发现。我希望 Paxos 的进一步考古发掘会在寻找自稳定性的手稿方面给予更高的优先级。

3.3.6 选择新的议员

开始的时候，议员是世袭的，从上一代传给下一代。当最早的政治家 TT 退休后，他把律簿交给了他的儿子。其他和 TT 共事的议员也没有什么不同。

随着旧的家庭移民出去和新的家庭移民进来，这个系统不得不改变了。Paxon 人决定通过法令来增加和取消议会成员。这暴露了一个循环的问题：议会的成员由法令的通过决定，而法令的通过又需要确定一个多数的议员集合由哪些成员组成，而这又反过来依赖于谁是议会的成员。让通过法令 n 的议员由法令 n-3 的法律确定的成员来组成可以打破这个循环。

总统在知道直到 3252 的所有法令之前，不会去通过法令 3255。事实上，通过法令：

3252：Epv 现在是一个议员了

之后，总统可以立即通过 3253 和 3254 这样的“橄榄节”法令。

这种改变议会构成的方式是一种危险并且容易出错的方式，虽然一致性和进展性可以保持。**进展性条件只在多数议员呆在议会厅的时候才能保证进展性，但是它不能保证议会厅里永远有多数的议员在。**事实上，选择议员的机制可导致了 Paxos 议会系统的崩溃。因为一个抄写错误，一条本来想为海难中遇难的水手们赋予荣誉的法令，却将他们声明为议会中仅有的议员。这条法令的通过阻止了所有新法令的通过——包括那些试图纠正这个错误的法令。Paxos 政府就这样停止运转了。一个叫 Aav 的将军趁着这个混乱时期发动了兵变，建立了一个军事独裁政权，结束了发展了几个世纪的议会政权。Paxos 在几任腐败堕落的独裁者统治下变得衰弱了。终于不能抵挡一次来自东方的入侵，他们的文明也在这次异族入侵下毁灭了。

第四章 与计算机科学的关系

4.1 状态机模式 (The State Machine Approach)

尽管 Paxos 的议会在几个世纪之前就毁灭了，但他们的协议仍然有用。例如，考虑一个可能用来做命名服务的简单的分布式数据库系统。数据库的状态由多个名字和对应的值组成，多个服务器维护多个数据库副本。客户端程序可以向任一服务器出请求，读取或改变某个名字的值。这里有两种读取请求：一种是慢速读取，返回名字当前对应的值；另一种是快速读取，返回更快，但是可能会反映不出数据库最近的更新。

在这种数据库系统和 Paxos 议会之间，明显有一种对应

议会		分布式数据库
议员	↔	服务器
公民	↔	客户端程序
当前法令	↔	数据库状态

客户端请求修改一个值，对应通过一条法令。一个慢速读取也涉及到通过一项法令，见 3.3.4 节的描述。一个快速读取，读取服务器的当前数据库版本。Paxos 议会协议供了一个数据库系统的分布式、容错实现。

这种实现分布式数据库的方法，是由 Lamport 在 1978 年首次提出的状态机模式的一个实例。在这种模式里，人们首先定义一个状态机，状态机由一组状态集合，一组命令集合，一组响应集合，和一个函数组成。函数为每一个命令/状态对指定一个响应/状态对(一个响应和一个状态组成的对)。直观地讲，状态机通过产生一个响应和改变它的状态来执行一个命令；命令和状态机的当前状态决定了它的响应和新的状态。对于分布式数据库来说，状态机的状态就是数据库的状态。状态机的命令以及确定响应和新状态的函数述如下:

命令：	read(name,client)	update(name,val,client)
响应：	(client,value of name)	(client, “ok”)
新状态：	和当前状态相同	除了 name 的值改变成 val 外，和当前状态相同

在状态机模式中，一个系统由联网的多个服务器进程实现。服务器将客户端请求转换为状态 机的命令，执行命令，然后将状态机的响应转换为给客户端的应答。一个通用的算法保证所有的服务器得到相同序列的命令，从而保证它们都产生同样序列的响应和状态变更——假设它们都从同样的初始状态开始。**在数据库的例子中，进行一个慢速读或改变一个值的客户端请求被转换成状态机的 read 或 update 命令。**这个命令被执行，然后状态机的响应被转换成一个对客户端的应答，由接收请求的服务器发送给客户端。**因为所有的服务器执行相同序列的状态机命令，它们都保持了数据库的一致版本。**尽管如此，**在任一时刻，某些服务器的版本可能会比其他服务器旧，因为一个状态机命令不需要在所有服务器上同一时刻执行。一个服务器使用它状态的当前版本来应答一个快速读请求，不执行状态机命令。**

状态机表达了系统的功能，而本身只是一个从命令/状态到响应/状态的函数。同步和容错的问题通过通用的算法处理，服务器通过这个通用的算法来取得命令序列。当定义一个新的系统时，只有状态机是新的。服务器通过一个已经证明了是正确的标准分布式算法得到状态机命令。函数相比于分布式算法来说非常容易设计，且更不容易出错。

实现无限状态机的第一个算法出现在[Lamport 1978]。之后，**容忍固定个数 f 之内任意错误的算法被发明[Lamport 1984]。这些算法保证，如果小于 f 个进程失败，那么状态机命令会在固定长度的时间内执行。**这个算法因此适合需要实时响应的应用。但是如果超过 f 个进程失败发生，那么不同的服务器可能会有不一致的状态机副本。此外，两台服务器之间不能相互通信等价于他们其中一台失败。为了把系统发生不一致的可能性将到很低，必须使用 f 值更大的算法，这也意味着要在硬件冗余、通信带宽和响应时间上付出更大代价。

Paxos 议会协议提供了另一个途径来实现一个无限状态机。议员的法律书对应于状态，通过一个法令对应于执行一个状态机命令。最终的算法没有之前的算法健壮，也没有之前的算法昂贵。它**不能容忍任意的，恶意的错误，也不能保证有限时间内的响应。但是保持了任意数量的进程和通信路径（良性）失败下的一致性。**Paxos 算法适合那些要求适度的可靠性但是不希望在极度容错和实时性上投入太多的系统。

如果状态机的算法可以保证响应时间在确定的范围之内(time-bounded)，那么时间可以作为状态的一部分，状态机的动作可以通过时间的流逝来触发。例如，考虑一个分配资源的系统。赋予一个客户端资源的时间，可以包含在状态里，这样状态机可以在这个客户端持有资源太久时自动执行一个命令取消其授权。

在 Paxos 算法里，时间没办法以这样自然的方式成为状态的一部分。如果错误发生，它可以用任意长的时间执行命令(通过一个法令)，一个命令可以在另一个更早出(更早出现在法令序列中)的命令之前执行。但是，一个状态机仍然可以像 Paxos 议会那样使用实际时间。例如，3.3.3 节中述的如何确定谁是当前奶酪检查官的方法，可以用来确定谁是当前的资源拥有者。

4.2.(Commit Protocols)

Paxos 神会协议与标准的 3 阶段提交协议[Bernstein 1987 ,Skeen 1982]很类似。Paxos 表决和 3 阶段提交协议都包含一个协调者(总统)和其他法定成员(议员们)之间的 5 个消息的交换。提交协议选择 commit 和 abort 两个值中的一个，而神会协议选择任意的一个法令。将提交协议转化为神会协议，只需在最初一轮的消息中发送法令。一个 commit 决定表示通过这个法令，一个 abort 决定表示通过那个“橄榄节”法令(没有任何影响的法令)。

神会协议与上述转化后的提交协议不同，因为法令直到第二阶段才发送。这允许相应的初级协议为所有的法令只执行一次第一阶段，这样通过每一个单独的法令只需要 3 个消息的交换。

作为神会协议基础的几个定理与 Dwork，Lynch 和 StockMeyer[1988]得出的结论类似。但是他们的算法在不同轮中串行的执行表决，看起来与神会协议没有什么关联。

第五章 附录:神会协议一致性的证明

TODO

第六章 End

关注公众号

欢迎大家公众号留言反馈。

邮箱：631521383@qq.com