[译]兼职议会（Paxos）

2021-10-16

https://ying-zhang.cn/dist/1989-paxos-cn/

The Part-Time Parliament

Leslie Lamport, *Digital Equipment Corporation.*

Concurrency: the Works of Leslie Lamport. 277–317. Oct. 2019. doi.org/10.1145/3335772.3335939

Originally published in *ACM Transactions on Computer Systems (TOCS)*. 16(2), 133–169. May 1998. doi.org/10.1145/279227.279229

Originally published in Compaq-DEC Technical Report SRC-RR-49, Sept. 1989

https://lamport.azurewebsites.net/pubs/lamport-paxos.pdf

译者：Ying ZHANG. 2021-09, 10; 2025-06.

Authors' address: Systems Research Center, Digital Equipment Corporation, 130 Lytton Avenue, Palo Alto, CA 94301.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
©1998 ACM 0734-2071/98/0500-0133 $5.00.

我的文章（节选）

Leslie Lamport

https://lamport.azurewebsites.net/pubs/pubs.html#lamport-paxos

123. *The Part-Time Parliament*

ACM Transactions on Computer Systems, 16, 2 (May 1998), 133-169. 以及 SRC Research Report 49. 本文于1990年投稿，创下了我发表文章的耗时记录，后来被 [60] 打破了。

SRC在80年代末开发了一个名为Echo的容错文件系统。开发者声称，不论出现多少非拜占庭的故障，它总能保持一致，且如果大多数进程都正常，那么它就能正常推进(make progress)。与大多数此类系统一样，没有任何故障时，事情非常简单，但要复杂的算法来处理实现者可能想到的各种故障。我认为他们的尝试是不可能的，并着手证明这一点。事与愿违，我发现了本文中描述的 Paxos 算法。该算法的核心是一个三阶段共识协议。为了避免出现任意单个故障时发生阻塞，需要三阶段协议，Dale Skeen 似乎是第一个认识这一点的。然而，据我所知， Paxos是第一个真正的，有明确表述的正确性条件和证明的三阶段提交协议。

我过去认为，并且一直认为，Paxos 是一个重要的算法。受到自己用拜占庭将军〔的故事〕来介绍从而普及共识问题的成功经历启发，我决定将算法表示为古希腊岛屿议会〔的故事〕。Leo Guibas 建议将岛的名字称为Paxos。我用该领域的计算机科学家的名字来命名希腊立法者，在 Guibas 的帮助下音译为假的希腊方言。（Peter Ladkin 推荐了标题。）虚构失落的文明，让我能省略无趣的细节，并托词议会协议的一些细节已经丢失来泛谈。为了进一步展示这个形象，我扮演成印第安纳·琼斯式的考古学家做了几场讲座，戴着牛仔帽，还有扁酒壶〔译注：印第安纳·琼斯是《夺宝奇兵》系列电影的主要人物〕。

我试图在该主题中加入一些幽默的尝试失败了。听过我讲座的人记得印第安纳·琼斯，但不记得算法。论文的读者显然被希腊故事分心，无法理解算法。收到我发送论文的人们中，声称已经读过的有Nancy Lynch， Vassos Hadzilacos， 和Phil Bernstein。几个月后，我通过电子邮件向他们发送了下面的问题：

> 你能否实现一个分布式数据库，它可以容忍任意数量（可能是所有）进程的故障而不破坏一致，且多数进程再次正常工作后，此数据库也将恢复正常行为？

他们都没有注意到这个问题与 Paxos 算法之间有任何联系。

1990年，我把论文提交给*TOCS*期刊，所有三位审稿人都说这篇论文有点意思，虽然不是很重要，但是关于Paxos岛的内容都得删掉。我对从事该领域的人没有幽默感而恼火，于是将论文束之高阁。几年后，SRC 的一些人需要为他们构建的分布式系统寻找算法，而 Paxos 正是他们所需的。我把论文给他们看，他们非常满意。下面是Chandu Thekkath 讲述的 Paxos 在SRC的历史。

> 当 Ed Lee 和我开发Petal时， 我们需要一个提交协议，以保证分布式系统中的全局操作，在服务器可能故障的情况下，仍能正确完成。我们了解到 3PC （三阶段提交）协

> 议，并从 Bernstein，Hadzilacos 和 Goodman 的书 *Concurrency Control and Recovery in Database Systems* 中学到了它的内容。我们发现该协议有点难以理解，因此没有尝试实现它。就在这段时间，Mike Schroeder 向我们介绍了 Leslie Lamport 发明的共识协议，并建议我们向他请教该协议。Lamport 给了Ed一份*The Part-Time Parliament*技术报告的副本，我们都读得很有趣。我特别喜欢它的幽默，直到今天，我也无法理解人们为什么不喜欢那篇技术报告。Paxos 拥有我们系统所需的所有必要特性，我们认为可以实现它。Leslie 也提供了重要的咨询帮助，据我所知，这产生了 Paxos 算法的第一个实现（包括动态重配置）。一年后，当我们需要一个用于 Frangipani 文件系统的分布式锁服务时，我们又用到了 Paxos。

所以，我想也许是时候尝试再次发表它了。

与此同时，这个不幸故事的一个例外是Butler Lampson〔译注：1992年图灵奖得主〕，他立即理解了该算法的重要性。他在多次讲座和一篇论文都提到了Paxos算法，还吸引了Nancy Lynch对此的兴趣。De Prisco，Lynch 和 Lampson 发表了他们对此算法的规约和证明。他们的论文更明显地表明我该发表自己的论文了。于是，我向当时的*TOCS*编辑 Ken Birman 提议经他发表。他建议修改，或许添加算法的 TLA 规范。但是重读论文后，我相信算法的描述和证明足够准确和严谨，虽然与我现在的写作不同。诚然，该论文需要修改，以考虑到期间发表的工作。为了延续玩笑且减少自己工作，我建议不写修订版，而是将其称为最近重新发现的手稿来发表，并附有 Keith Marzullo 的注释。Marzullo 愿意，Birman 同意了，论文终于刊出了。

还有个关于排版的有趣附注。为了突出 Marzullo 的注释，我决定为其设置灰色背景。ACM 最近采用了似乎很棒的新排版软件，而*TOCS*不接受排版好的副本。不幸的是，他们很棒的新软件不支持底纹。因此，我必须为带底纹的文字提供排版好的副本。此外，他们聪明的软件对这段文字只接受浮动图形格式，因此 Marzullo 的注释没有出现在预期的位置。此外，他们显然昂贵的软件无法排版复杂的数学公式。（毕竟这是一本计算机期刊，为什么要排版公式？）因此，我还要为附录A2一节不变式定义提供排版好的副本，在发表的版本中，他们将其作为图 3插入。因此，该图中的字体与论文其余部分的字体不匹配。

2012年，这篇论文获得了 ACM SIGOPS 名人堂奖。

最近在Paxos岛上的考古发现表明，尽管兼职的立法员惯于四处游荡，但议会仍可运转。虽然立法员经常闪现于议事厅，而且他们的信使粗心健忘，但是这些立法员仍维护着一致的议会记录副本。Paxos议会协议提供了一种新的途径来实现状态机方法，用以设计分布系统。

最近在*TOCS*编辑办公室的文件柜后面发现了这份投稿。尽管年代久远，主编仍认为值得发表。由于作者目前在希腊的群岛进行实地考察，无法联系，委托我准备文稿以发表。

作者似乎是一位考古学家，对计算机科学的兴趣浅尝辄止。这是不幸的；尽管大多数计算机科学家对他描述的晦涩古老的Paxos文明没什么兴趣，但它的立法体系是异步环境中实现分布计算机系统的绝佳模型。事实上，Paxos岛人对其协议所做的一些改进似乎尚未被系统文献得知。

作者确实在第4节中简要讨论了Paxos议会与分布式计算的关联。计算机科学家可能想先阅读该节。甚至在此之前，他们可能想阅读Lampson [1996]向计算机科学家解释的上述算法。De Prisco等[1997]更形式化地介绍了该算法。在第4节之后，我添加了关于古代协议和近期研究之间关联的进一步评论。

Keith Marzullo
加州大学圣地亚哥分校

1. 问题

1.1 Paxos岛

约一千年前，爱琴海的Paxos岛是繁荣的商业中心 [原注1. 不要与Ionian群岛的Paxoi岛混淆，后者有时被误认为**Paxos**]〔译注：原文的Early in this millennium，注意到本文发表于公历的第二个千年末期，但后文提到的一些日期似乎是指第一个千年，即公元元年前后。当然，这对本文没有实质影响〕。财富催生了复杂的政治，Paxos岛人用议会形式的统治取代了他们古老的神权政治。但生意重于公民义务，Paxos岛没有人愿意为议会全职工作。即使立法员不断从议事厅进进出出，Paxos岛的议会也要能够运转。

兼职议会执政的问题与当今容错分布系统面临的问题有明显的对应关系：其中，立法员对应进程，离开议事厅对应进程失效。因此，计算机科学家可能会对Paxos岛人的解决方案感兴趣。我在本文简要介绍了Paxos议会协议的历史，然后更简短地讨论了它与分布系统的关联。

Paxos文明被外来入侵摧毁，考古学家最近才开始发掘它的历史。因此，我们对Paxos议会的了解是零碎的。虽然已知基本协议，但我们仍缺失很多细节。对这些有趣的细节，我将冒昧地推测Paxos岛人可能的做法。

1.2 要求

议会的主要任务是确定关于本地的法律〔译注：law of the land，意为宪法或基本法〕，这是指由议会批准的一系列法令（decree）。现代议会聘请秘书来记录活动，但Paxos岛没有人愿意在整个会议期间留在议事厅担任秘书。取而代之，每位Paxos立法员都保有一本**帐簿**（ledger），在其中记录了议会批准的一系列有编号的法令。例如，立法员Λĭνχϑ的帐簿有如下条目

155：橄榄税是每吨3德拉克马

如果她相信议会批准的第155号法令将橄榄税规定为每吨3德拉克马。帐簿是用不褪色的墨水书写的，其中的条目不能被篡改。

议会协议的第一项要求是**帐簿的一致性**（consistency of ledgers），即任意两本帐簿都不能包含相互矛盾的信息。如果立法员Φισϑερ的帐簿上有条目

132：灯只能使用橄榄油

那么，没有其他立法员帐簿的第132号法令内容与之不同。但是，如果一位立法员还没有得知此条法令已获批准，那么他的帐簿上可能没有第132号法令的记录。

帐簿的一致性是不够的，因为可以通过将所有帐簿留空来实现。需要若干要求来保证法令最终得以批准并记录到帐簿上。在现代议会，法令无法批准是因为立法员之间的分歧。在Paxos岛情况并非如此，那里相互信任的风气盛行。Paxos的立法员愿意批准任何提议的法令。然而，他们的离岗游荡倾向产生了一个问题。如果一组立法员批准了下面的法令，然后离开，去参加宴会

> 37：禁止在寺庙墙壁上绘画

而另一组立法员进入议事厅，对刚刚发生的事情一无所知，批准了矛盾的法令，就破坏了一致性

> 37：保障艺术表达的自由

除非有足够多的立法员在议事厅停留足够长的时间，否则无法保证取得进展。因为Paxos的立法员不愿意限制他们的外部活动，所以不可能确保任意法令都会被批准。然而，立法员愿意保证，在议事厅，他们和他们的助手会对所有议会事务迅速采取行动。这种保证允许Paxos设计满足以下**进展条件**（progress condition）的议会协议。

> 如果大多数立法员在议事厅内 [原注2. 在翻译"进展条件"时，我将Paxos的词μαδζϑωριτῖσετ译为**多数集**（majority）。在第2.2节提出并讨论了该词的其它译法]，并且足够长的时间内没有人进入或离开议事厅，那么议事厅内立法员提议的任何法令都将获得批准，并且已批准的每条法令都会出现在议事厅内每位立法员的帐簿上。

1.3. 假设

只有向立法员提供必要的资源，才能实现议会协议的要求。每位立法员都收到了一本结实的帐簿，用于记录法令、一支笔和不褪色的墨水。如果离开议事厅，立法员可能会忘记他们刚才在做什么 [原注3. 有一次悲惨的事故，立法员Τωυεγ被议事厅外一尊倒下的雕像击中头部，患上了不可逆转的健忘症]，所以他们会在帐簿背面写下笔记，提醒自己重要的议会任务。法令清单中的条目不会被篡改，但笔记可以被划掉。实现进展条件要求立法员能够测量时间的流逝，因此他们还收到了简单的计时沙漏。

立法员随时携带他们的帐簿，并且可以随时阅读法令清单和所有未被划掉的笔记。帐簿由最精美的羊皮纸制成，仅用于最重要的笔记。立法员会将其它笔记写在一张纸条上，如果离开了议事厅，他可能会（也可能不会）弄丢纸条。

议事厅的音响效果很差，无法演讲。立法员只能通过信使交流，并有资金随意雇用信使。可以指望信使不会篡改消息，但他可能会忘记他已经投递了一条消息，于是重复投递。与他们所服务的立法员一样，信使也是兼职服务议会的。信使可能离开议事厅处理个人事务——也许是航行六个月——然后才投递消息。他甚至可能永远离开，在这种情况下，消息将永远不会被投递。

尽管立法员和信使可以随时进出，但在议事厅内，他们致力于议会的事务。当他们留在议事厅时，信使及时投递消息，立法员对他们收到的任何消息迅速作出反应。

Paxos的官方记录声称立法员和信使恪守诚信，并严格遵守议会协议。大多数学者认为这是宣传，旨在将Paxos描绘成道德上优于其东方邻居。不诚实虽然少见，但毫无疑问发生过。然而，因为它从未在官方文件中提及，我们对议会如何应对不诚实的立法员或信使知之甚少。第3.3.5节将讨论已发现的证据。

2. 单一法令宗教会议（Synod）

Paxos岛的议会是从早期的牧师宗教会议（Synod of priests）演变而来的，该大会每19年召开一次，选定一条象征意义的法令。几百年来，Synod使用传统的程序来选定法令，即要求所有牧师到场出席。但随着商业兴盛，牧师开始在Synod进行期间〔开小差〕进出议事厅。最后，旧的协议失败了，Synod无果而终。为了避免重蹈覆辙，Paxos的宗教领袖要求数学家制定一项确定Synod法令的协议。该协议的

要求和假设与之后的议会制度的要求和假设基本相同，只是帐簿上最多只有一条法令，而非一系列法令。本节描述了由此产生的Synod协议；第3节介绍〔多法令的〕议会协议。

数学家通过一系列步骤推导出Synod协议。首先，他们证明的结果表明，满足某些约束的协议能保证一致性，并允许进展。从这些约束直接导出了**初步协议**（preliminary protocol）。初步协议的修改版本是**基本协议**（basic protocol），能保证一致性，但不保证进展。通过修改基本协议，得到了完整Synod协议，满足一致性和进展要求[原注4. 探索Synod协议的完整历史尚不清楚。像现代计算机科学家一样，Paxos的数学家会描述优雅的逻辑推导，与实际如何推导出算法毫无相似之处。然而，人们了解到数学的结论确实是先于协议得到的（第2.1节的定理1，2）。当时数学家为了应付制定协议的要求，尝试证明不可能存在一项令人满意的协议，然后发现了上述数学定理]。

第2.1节介绍了数学的结论，第2.2~2.4节中非形式化地介绍了协议。基本协议的形式化说明和正确性证明请见附录。

2.1 数学的结论

Synod的法令是通过一系列有编号的**表决**（ballots）选定的。表决是对单条法令的全民表态。对每轮表决，每位牧师只能选择投票赞成法令（voting for）或者弃权 [原注5. 像一些现代国家一样，Paxos岛人还没有完全理解雅典民主的本质]。与表决相关的是牧师的集合，称为**法定人数集合**（quorum）。当且仅当Quorum的每位牧师都投票赞成该法令时，表决才是成功的。形式化的，一轮表决B由以下4项内容组成（除非另有说明，**集合**（set）是指**有限集**（finite set）[原注6. 虽然Paxos的数学家在他们那个时代非常先进，但他们显然没有集合论的知识。我冒昧地将Paxos的原始符号翻译成现代集合论的语言])。

- $B.dec$ 一条法令（待投票）。
- $B.qrm$ 非空的牧师集合（表决的Quorum）。
- $B.vot$ 牧师集合（投票赞成法令的）[原注7. 只有Quorum中牧师的投票有实际影响，但Paxos的数学家发现，如果在他们的证明中，允许任意牧师在每轮表决中投票，那么更容易说服人们该协议是正确的]。
- $B.bal$ 表决的编号。

〔译注：原文中，上述符号是下标格式的，如B_{dec}，译文中全部改用点号格式。〕

当且仅当$B.qrm \subseteq B.vot$，称这一轮表决B是**成功的**，即一轮成功的表决意味着每位Quorum的成员都投了票。

表决编号是一个无界有序的数值集合中的元素。若$B'.bal > B.bal$，则称表决B'**晚于**B。然而，这并不反映表决的实际顺序；较“晚”的表决实际上可能先于较“早”的表决发生。

Paxos的数学家在表决的集合$\mathcal{B}$上定义了三个条件，然后表明如果进行的多轮表决满足这些条件，那么可以保证一致性，并且有可能取得进展。前两个条件很简单；它们可以非形式化地表述如下。

- $B1(\mathcal{B})$：$\mathcal{B}$中的每轮表决有唯一的编号。
- $B2(\mathcal{B})$：$\mathcal{B}$中任意两轮表决的Quorum至少有一位共同的牧师。

第三个条件比较复杂。一份Paxos手稿包含以下相当令人迷惑的说明。

- $B3(\mathcal{B})$：对$\mathcal{B}$中的每轮表决B，如果B的Quorum中有牧师参与了$\mathcal{B}$中早先轮次的表决，那么B的法令应与先前的最近一轮表决的法令相同。

图1中描绘的手稿有助于解释这个拗口的说明。该图解释了$B3(\mathcal{B})$，其中$\mathcal{B}$包含五轮表决，由五位牧师参与Synod，分别为A，B，Γ，Δ，和E。对$\mathcal{B}$包含的五轮表决，每轮表决的牧师集合都在Quorum中，将其姓名加方框表示。例如，第14号表决的法令是α，某个Quorum包含三位牧师，有两位投票了。条

件$B3(\mathcal{B})$的“对$\mathcal{B}$中的每轮表决B：...”，其中“...”是对一轮表决B的约束条件。图1中五轮表决B的约束条件如下。

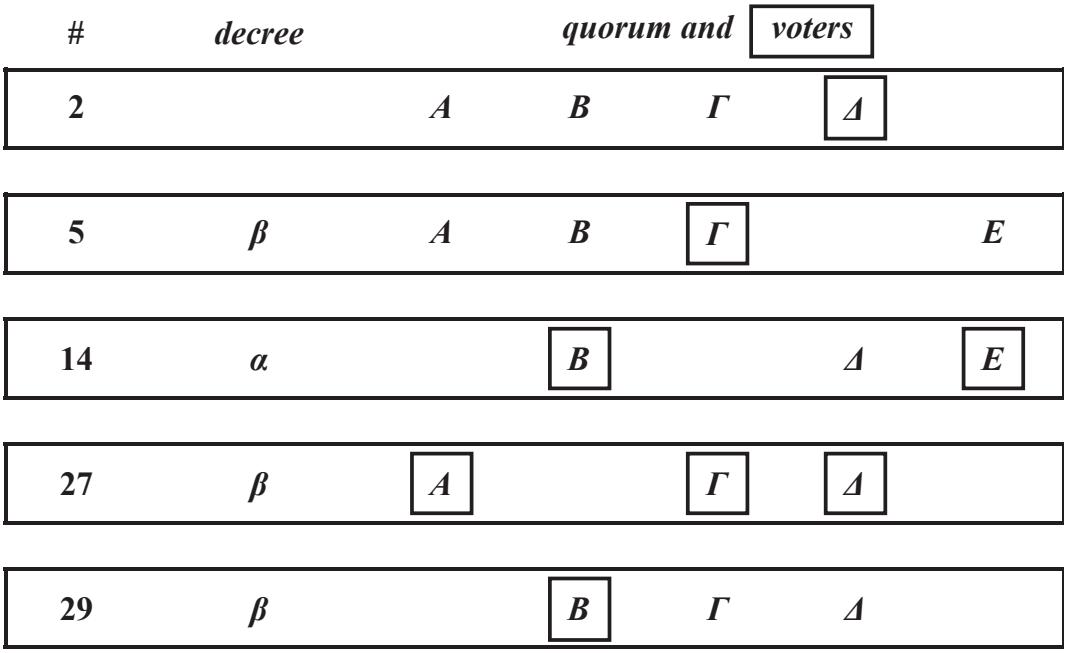

图 1. Paxos手稿显示了由五轮表决组成的集合$\mathcal{B}$，满足条件$B1(\mathcal{B})$~$B3(\mathcal{B})$（添加了解释性的列标题）。
〔译注：注意，图中decree这一列的值是等右侧投票结束后才确定的。〕

2. 第2号表决是最早的表决，因此该轮表决的约束条件显然满足。
5. 第5号表决的某个Quorum的四位成员都没有在较早的表决中投过票，因此该轮表决的约束条件也显然满足。
14. 第14号表决的某个Quorum中，唯一在较早的表决中投过票的成员是Δ，它在第2号表决中投票了，因此约束条件要求第14号表决的法令必须等于第2号表决的法令。
27. 第27号表决（这是一次成功的表决）的某个Quorum是A，Γ，和Δ。牧师A没有在较早的表决中投过票，Γ参与投票的唯一一轮较早的表决是第5号表决，Δ参与投票的唯一一轮较早的表决是第2号表决。先前的这两轮表决中，最近的一轮是第5号，因此约束条件要求第27号表决的法令必须等于第5号的法令。
29. 第29号表决的某个Quorum是B，Γ，和Δ。B参与投票的唯一一轮较早的表决是第14号表决，牧师Γ在第5号和27号表决中投票了，Δ在第2号和27号表决中投票了。先前的这四轮表决中，最近的一轮是第27号表决，因此约束条件要求第29号表决的法令必须等于第27号表决的法令。

为了形式化地说明$B1(\mathcal{B})$~$B3(\mathcal{B})$，需要更多的记号。投票v定义为三元组：牧师$v.pst$，表决编号$v.bal$，和法令$v.dec$。它表示牧师$v.pst$在第$v.bal$号表决为法令$v.dec$投出的赞成票。Paxos岛人还定义了无效票（$null$），即投票v，且$v.bal = -\infty$，$v.dec = BLANK$，其中，对任意表决的编号b都有$-\infty < b < \infty$，$BLANK$不是一条法令。对于任意牧师p，还定义$null_p$是他的专属无效票，即无效票v，且$v.pst = p$。

Paxos的数学家在所有投票的集合上定义了全序，但包含该定义的部分手稿已丢失。剩余的片段表明，对于任意投票v和v'，若$v.bal < v'.bal$，则$v < v'$。但不知$v.bal = v'.bal$时如何规定v与v'的顺序。

对于任意的表决集合$\mathcal{B}$，其中的投票集合$Votes(\mathcal{B})$定义为包含所有投票v的集合，满足$v.pst \in B.vot$，$v.bal = B.bal$，且$v.dec = B.dec$，其中$B \in \mathcal{B}$。如果p是一位牧师，b是表决编号或为$\pm\infty$，那么定义$MaxVote(b, p, \mathcal{B})$为$Votes(\mathcal{B})$中，由$p$投出的，编号$v.bal < b$的，且编号最大的选票$v$〔译注：即编号**仅次于**$b$

的选票〕，若不存在则为$null_p$。由于$null_p$小于任意实际投票，这意味着$MaxVote(b,p,\mathcal{B})$是如下集合中**最大**的投票

$$\{v \in Votes(\mathcal{B}) : (v.pst = p) \wedge (v.bal < b)\} \cup \{null_p\}$$

对任意非空的牧师集合Q，$MaxVote(b,Q,\mathcal{B})$定义为所有$MaxVote(b,p,\mathcal{B})$中的最大值，其中p属于Q。

条件$B1(\mathcal{B})$~$B3(\mathcal{B})$形式化地说明如下[原注8. 我使用Paxos的数学符号 $\triangleq$，表示**按定义等于**]。

$B1(\mathcal{B}) \triangleq \forall B, B' \in \mathcal{B} : (B \neq B') \Rightarrow (B.bal \neq B'.bal)$

$B2(\mathcal{B}) \triangleq \forall B, B' \in \mathcal{B} : B.qrm \cap B'.qrm \neq \emptyset$

$$\begin{aligned} B3(\mathcal{B}) \triangleq \forall B \in \mathcal{B} : &(MaxVote(B.bal,\ B.qrm,\ \mathcal{B}).bal \neq -\infty) \\ &\Rightarrow (B.dec = MaxVote(B.bal,\ B.qrm,\ \mathcal{B}).dec) \end{aligned}$$

虽然$MaxVote$的定义取决于投票的顺序，但是$B1(\mathcal{B})$意味着，对表决编号相同的投票，$MaxVote(b,Q,\mathcal{B}).dec$与其排序方式无关。

为了表明这些条件意味着一致性，Paxos岛人首先表明$B1(\mathcal{B})$~$B3(\mathcal{B})$意味着，如果$\mathcal{B}$中的表决B成功了，那么$\mathcal{B}$中更晚的表决涉及的法令都与B相同。

引理 如果$B1(\mathcal{B})$，$B2(\mathcal{B})$，和$B3(\mathcal{B})$成立，那么对$\mathcal{B}$中的任意B，B'，都有

$$((B.qrm \subseteq B.vot) \wedge (B'.bal > B.bal)) \Rightarrow (B'.dec = B.dec)$$

引理的证明

对$\mathcal{B}$中的任意表决B，令$\Psi(B,\mathcal{B})$是$\mathcal{B}$中晚于B且法令与B不同的表决的集合：

$$\Psi(B,\mathcal{B}) \triangleq \{B' \in \mathcal{B} : (B'.bal > B.bal) \wedge (B'.dec \neq B.dec)\}$$

为了证明引理，只要证明如果$B.qrm \subseteq B.vot$，那么$\Psi(B,\mathcal{B})$是空集。Paxos岛人给出了反证法的证明。他们假设存在一轮表决B，$B.qrm \subseteq B.vot$，且$\Psi(B,\mathcal{B}) \neq \emptyset$，得到了如下的矛盾 [原注9. Paxos的数学家总是为重要的定理提供仔细的、结构化的证明。他们不像现代数学家那么高深，后者会省略很多细节，编写自然语言式的证明，却从不出错]。

1. 取$C \in \Psi(B,\mathcal{B})$，且$C.bal = \min\{B'.bal : B' \in \Psi(B,\mathcal{B})\}$。
   证明： C存在是因为$\Psi(B,\mathcal{B})$非空且有限。
2. $C.bal > B.bal$
   证明： 由1和$\Psi(B,\mathcal{B})$的定义。
3. $B.vot \cap C.qrm \neq \emptyset$
   证明： 由$B2(\mathcal{B})$和引理的假设$B.qrm \subseteq B.vot$。
4. $MaxVote(C.bal,\ C.qrm,\ \mathcal{B}).bal \geq B.bal$〔译注：以下将$MaxVote(C.bal,\ C.qrm,\ \mathcal{B})$记为$v_{\max}$〕
   证明： 由2，3，和$v_{\max}$的定义。
5. $v_{\max} \in Votes(\mathcal{B})$
   证明： 由4（可推出$v_{\max}$不是无效票），以及$v_{\max}$的定义。
6. $v_{\max}.dec = C.dec$
   证明： 由5和$B3(\mathcal{B})$。
7. $v_{\max}.dec \neq B.dec$
   证明： 由6，1，和$\Psi(B,\mathcal{B})$的定义。
8. $v_{\max}.bal > B.bal$
   证明： 由4，因为7和$B1(\mathcal{B})$意味着$v_{\max}.bal \neq B.bal$。

9. $v_{\max} \in Votes(\Psi(B, \mathcal{B}))$
   证明： 由7，8，及$\Psi(B, \mathcal{B})$的定义。
10. $v_{\max}.bal < C.bal$
    证明： 由$v_{\max}$的定义。
11. 矛盾
    证明： 由9，10和1。

引理证毕

〔译注：由1的定义，C是$\Psi(B, \mathcal{B})$中编号最小的那轮表决；由9，$v_{\max}$是$\Psi(B, \mathcal{B})$中某轮表决的投票，从而$v_{\max}.bal \geq C.bal$，但由10，$v_{\max}$定义为仅次于$C.bal$的投票，即$v_{\max}.bal < C.bal$，从而产生了矛盾。〕

有了这个引理，很容易证明，如果$B1$~$B3$成立，那么任意两次成功的表决涉及的法令是相同的。

定理 1 如果$B1(\mathcal{B})$，$B2(\mathcal{B})$，和$B3(\mathcal{B})$成立，那么对$\mathcal{B}$中的任意B，B'，都有

$$((B.qrm \subseteq B.vot) \wedge (B'.qrm \subseteq B'.vot)) \Rightarrow (B'.dec = B.dec)$$

定理的证明

如果$B'.bal = B.bal$，那么 $B1(\mathcal{B})$意味着$B' = B$。如果$B'.bal \neq B.bal$，那么立即可以从引理推出定理。

定理证毕

〔译注：$B'.bal \neq B.bal$意味着$B'.bal > B.bal$或$B.bal > B'.bal$，注意到对称性，将任一项与定理1的前提结合，即可应用引理。〕

然后Paxos岛人证明了另一个定理，即如果议事厅内有足够多的牧师，那么就有可能进行一次成功的表决，且符合$B1(\mathcal{B})$~$B3(\mathcal{B})$。虽然这并不能保证取得进展，但至少表明基于$B1(\mathcal{B})$~$B3(\mathcal{B})$的表决协议不会死锁。

定理 2 令b为表决编号，Q为一组牧师，使得对所有$B \in \mathcal{B}$，都有$b > B.bal$且$Q \cap B.qrm \neq \emptyset$。如果$B1(\mathcal{B})$，$B2(\mathcal{B})$，和$B3(\mathcal{B})$成立，那么存在表决$B'$，其中$B'.bal = b$且$B'.qrm = B'.vot = Q$，使得$B1(\mathcal{B} \cup \{B'\})$，$B2(\mathcal{B} \cup \{B'\})$，和$B3(\mathcal{B} \cup \{B'\})$也成立。

定理的证明

约束条件$B1(\mathcal{B} \cup \{B'\})$由$B1(\mathcal{B})$，及$B'.bal$和$b$的定义可得。约束条件$B2(\mathcal{B} \cup \{B'\})$由$B2(\mathcal{B})$，及$B'.qrm$和$Q$的定义可得。如果$MaxVote(b, Q, \mathcal{B}).bal = -\infty$，那么$B'.dec$可以是任意法令，否则令其为$MaxVote(b, Q, \mathcal{B}).dec$。从而由$B3(\mathcal{B})$可得约束条件$B3(\mathcal{B} \cup \{B'\})$成立。

定理证毕

2.2 初步协议

Paxos岛人从约束条件$B1(\mathcal{B})$~$B3(\mathcal{B})$保持成立的要求推导出**初步协议**，其中$\mathcal{B}$是已经或正在进行的所有表决的集合。协议的定义规定了集合$\mathcal{B}$如何变化，但从未明确得到过该集合。Paxos岛人将$\mathcal{B}$称为只有诸神才能观察到的量，因为它可能永远不会被任何凡人所知。

每轮表决都由一位牧师发起，他确定了表决编号、法令和Quorum。然后，Quorum的每位牧师决定是否在表决中投票。从保持$B1(\mathcal{B})$~$B3(\mathcal{B})$的要求可以直接推导出相关的规则：发起人如何确定表决编号、法令和Quorum；以及牧师如何决定是否在表决中投票。

为保持$B1$，每轮表决的编号必须是唯一的。通过记住（在帐簿背面写下笔记）之前发起的表决，牧师可以很容易地避免用相同的编号发起两轮不同的表决。为了防止不同的牧师以相同的编号发起表决，需要将表决编号候选集合按牧师划分。虽然不知道这是如何实现的，但一个简单的方法是让表决编号由一个整数和一位牧师的姓名组成一对，使用字典顺序，其中

$$(13, \Gamma\rho\alpha\check{\imath}) < (13, \Lambda\iota\nu\sigma\epsilon\check{\imath}) < (15, \Gamma\rho\alpha\check{\imath})$$

因为在Paxos字母表中Γ排在Λ之前。总之，我们知道了每位牧师都有一组无限的表决编号供他使用。

为保持$B2$，表决的Quorum规定为包含牧师的μαδζ∂ωριτΐσετ。最初，μαδζ∂ωριτΐσετ指简单多数。后来，据观察，胖牧师比瘦牧师行动迟缓，在议事厅呆的时间更长，因此μαδζ∂ωριτΐσετ指任意一组牧师，其总体重超过所有牧师总体重的一半，而不再指简单多数。一群瘦牧师抱怨这不公平，实际体重被替换为基于牧师出勤记录的象征性权重。对μαδζ∂ωριτΐσετ的关键要求是，任意包含μαδζ∂ωριτΐσετ牧师的两个集合至少有一位共同的牧师。为保持$B2$，发起表决B的牧师确定$B.qrm$为多数集。

条件$B3$要求，如果$MaxVote(b, Q, \mathcal{B}).dec$不等于$BLANK$，那么编号为$b$，Quorum为$Q$的表决的法令必须是$MaxVote(b, Q, \mathcal{B}).dec$。如果$MaxVote(b, Q, \mathcal{B}).dec$等于$BLANK$，那么表决可以用任意法令。为保持$B3(\mathcal{B})$，在发起编号为$b$，Quorum为$Q$的新表决之前，牧师$p$必须确定$MaxVote(b, Q, \mathcal{B}).dec$。要实现这一点，$p$必须知道$Q$中每位牧师$q$的$MaxVote(b, q, \mathcal{B}).dec$。

注意到$MaxVote(b, q, \mathcal{B})$是$q$投出的所有选票中编号仅次于$b$的，若不存在则为$null_q$。牧师$p$通过与$q$交换消息获知$MaxVote(b, q, \mathcal{B})$。因此，由$p$发起的，执行单轮表决协议的前两步是：[原注10. 牧师p和q可能是同一位。为简单起见，这种情况下，协议描述为p向自己发送消息。实际上，牧师可以自言自语，不用信使]。

(1) 牧师p确定一个新的表决编号b，并向一组牧师发送一条$NextBallot(b)$消息。

(2) 一位牧师q收到$NextBallot(b)$消息后，向p发送$LastVote(b, v)$消息作为响应，其中v是q已投出的选票中表决编号仅次于b的，若不存在则是他的无效票$null_q$。

牧师q必须在他的帐簿背面记下他之前投出的票。

在q发送的$LastVote(b, v)$消息中，v等于$MaxVote(b, q, \mathcal{B})$。但随着发起新表决和投出新选票，集合$\mathcal{B}$随之变化。由于牧师$p$将要用$v$作为$MaxVote(b, q, \mathcal{B})$，来确定法令，为保持$B3(\mathcal{B})$成立，要求$q$发出$LastVote(b, v)$消息后，$MaxVote(b, q, \mathcal{B})$的值不会变化。为保持$MaxVote(b, q, \mathcal{B})$不变，禁止$q$在编号介于$v.bal$和$b$之间的表决中投票。发送了$LastVote(b, v)$消息，$q$也就承诺了不会参加此类投票（为了遵守这个承诺，$q$必须在他的帐簿上记录必要的信息）。

表决协议之后的两步是（由牧师p从步骤 (1) 开始）：

(3) 收到某个多数集Q中每位牧师的$LastVote(b, v)$消息后，牧师p用编号b，Quorum Q和法令d发起新的表决，其中按$B3$来确定d。之后他将表决记录在帐簿背面，并向Q中每位牧师发送$BeginBallot(b, d)$消息。

(4) 收到$BeginBallot(b, d)$消息后，牧师q决定是否可以在编号b的表决中投票（如果违反在其它轮表决中发出的$LastVote(b', v')$消息对应的承诺，那么他就不会投票）。如果q决定为编号b的表决投票，那么他向p发送一条$Voted(b, q)$消息，并将投票记录在自己的帐簿背面。

执行了步骤 (3)，就认为将表决B添加到了$\mathcal{B}$，其中$B.bal = b$，$B.qrm = Q$，$B.vot = \emptyset$（这次表决还没有投票），以及$B.dec = d$。在步骤 (4) 中，如果牧师q决定在表决中投票，那么认为执行此步骤改变了集合$\mathcal{B}$中的B，向$B.vot$添加了q。

牧师可以选择在步骤 (4) 中不投票，即使投票不会违反任何先前的承诺。事实上，该协议中的所有步骤都可以跳过。例如，牧师q可以忽略一条$NextBallot(b)$消息，不执行步骤 (2)。不采取动作可能会阻碍进展，但不会造成不一致，因为不会违反$B1(\mathcal{B})$~$B3(\mathcal{B})$。由于未收到消息的唯一效果是阻止采取对应动作，从而消息丢失也不会导致不一致。因此，即使牧师离开议事厅或消息丢失，该协议也能保证一致性。

收到重复的消息会产生重复的动作。除步骤 (3) 以外，重复执行动作没有影响。例如，步骤 (4) 中发送多次$Voted(b, q)$消息与仅发送一次效果相同。为防止步骤 (3) 重复执行，执行时要检查帐簿背面的记

录。因此，即使信使多次投递相同的消息，也能保持一致性条件。

步骤 (1)~(4) 描述了发起表决和投票的完整协议。剩下的是确定表决结果，并在选定法令后宣布。注意，表决是成功的，等价于Quorum的每位牧师都投票了。表决成功的法令即Synod协议确定的法令。协议的剩余步骤是：

(5) 如果p收到了来自Q（编号为b的表决的Quorum）的每位牧师q的$Voted(b,q)$消息，那么他将d（该轮表决的法令）记到帐簿上，并向每位牧师发送一条$Success(d)$消息。

(6) 收到$Success(d)$消息后，每位牧师将法令d记到各自的帐簿上。

步骤 (1)~(6) 描述了如何进行单轮表决。初步协议允许任意牧师随时发起新的表决。每一步都保持$B1(\mathcal{B})$~$B3(\mathcal{B})$，所以整个协议也保持这些条件。由于牧师只在帐簿上记录成功表决的法令，因此定理 1 意味着所有牧师的帐簿是一致的。该协议没有解决进展问题。

在步骤 (3) ，如果法令d是按条件$B3$确定的，那么该法令很可能之前已经记录到某位牧师的帐簿上了。这位牧师不必在Quorum Q中；可能他已经离开议事厅了。进而，如果步骤 (3) 允许放松确定d的规则，那么将无法保证一致性。

2.3 基本协议

初步协议中，牧师必须记录 (i) 他发起的每轮表决的编号，(ii) 他投出的每张选票，以及 (iii) 他发送的每条$LastVote$消息。对于忙碌的牧师来说，记录所有这些消息很困难。因此，Paxos岛人修改了初步协议，以获得更实用的**基本协议**，其中每位牧师p只需在他的帐簿背面保留以下消息：

- $lastTried[p]$：p发起的最晚一轮表决的编号，若没有则为$-\infty$。
- $prevVote[p]$：p投出的选票中，表决编号最大的那张，若从未投票则为$-\infty$。
- $nextBal[p]$：p发送过$LastVote(b,v)$消息的表决中，最大的编号b，若从未发过该消息则为$-\infty$。

初步协议的步骤 (1)~(6) 描述了如何由发起表决的牧师p实施单轮表决。初步协议允许p并发地实施任意轮表决。基本协议中，他一次只能实施一轮表决——编号为$lastTried[p]$的表决。发起这轮表决后，p会忽略所有与他之前发起的其它轮表决相关的消息。与编号为$lastTried[p]$的表决过程相关的所有信息，将被牧师p记在一张纸条上。如果他弄丢了这张纸条，那么他将停止实施该轮表决。

初步协议中，牧师q发来的$LastVote(b,v)$消息表达了承诺：即不在任何编号介于$v.bal$和b之间的表决中投票。基本协议中，它代表了更强的承诺：即不会在任何编号小于b的表决中投票。本来他在初步协议的步骤 (4) 中允许参与的投票，可能就被基本协议中这项更强的承诺阻止了。然而，由于初步协议总是允许q跳过投票，基本协议没有要求他做出任何初步协议禁止的事。

初步协议的步骤 (1)~(6) 变成了基本协议中实施表决的以下六个步骤（除了$lastTried[p]$，$prevVote[p]$，和$nextBal[p]$，p将实施表决用到的其它消息都记在一张纸条上）。

(1) 牧师p确定一个新的，大于$lastTried[p]$的表决编号b，之后将$lastTried[p]$改为b，并向一组牧师发送一条$NextBallot(b)$消息。

(2) 牧师q收到来自p的$NextBallot(b)$消息后，如果$b > nextBal[q]$，那么他将$nextBal[q]$改为b，向p回复$LastVote(b,v)$消息，其中v等于$prevVote[q]$（若$b \le nextBal[q]$，则忽略对应的$NextBallot(b)$消息）。

(3) 牧师p收到了某个多数集Q的每位牧师的$LastVote(b,v)$消息后，其中$b = lastTried[p]$，p发起新的表决，其编号为b，Quorum为Q，且法令为d，其中d是按$B3$来确定的。之后他向Q的每位牧师发送一条$BeginBallot(b,d)$消息。

(4) 牧师q收到$BeginBallot(b,d)$消息后，其中$b = lastTried[q]$，q为编号b的表决投票，并将$prevVote[q]$改为这张选票，再向p发送一条$Voted(b,q)$消息（若$b \neq nextBal[q]$，则忽略对应的$BeginBallot(b,d)$消息）。

(5) 如果p收到了Q（编号为b的表决的Quorum）的每位牧师q的$Voted(b,q)$消息，其中$b = lastTried[p]$，那么他将d（该轮表决的法令）记在帐簿上，并向每位牧师发送一条$Success(d)$消息。

(6) 收到$Success(d)$消息后，每位牧师将法令d记到各自的帐簿上。

基本协议是初步协议的修改版本，这意味着基本协议允许的每个动作也被初步协议允许。由于初步协议满足一致性条件，因此基本协议也满足该条件。与初步协议一样，基本协议允许跳过任意动作，因此它也不解决进展问题。

从$B1$~$B3$到基本协议的推导使其显然满足一致性条件。然而，一些同样“显而易见”的古老智慧被证明是错误的，持怀疑态度的市民要求更严格的证明。附录转抄了他们的Paxos岛数学家对协议满足一致性条件的证明。

2.4 完整Synod协议

基本协议保持了一致性，但它不确保进展，因为它只规定了牧师可以做什么；它不强制牧师做任何事。完整协议包含与基本协议相同的六个步骤，用于实施表决。为了帮助取得进展，它包含了一个明显的额外要求，即牧师应尽快执行协议步骤 (2)~(6)。但是，为了满足进展条件，要由**某位**牧师执行步骤 (1)，即发起表决。完整协议的关键在于确定牧师应当何时发起表决。

永不发起表决必然没有进展。然而，发起太多表决也会阻碍进展。如果b大于任意其它表决编号，那么牧师q在步骤 (2) 收到$NextBallot(b)$消息后，做出的承诺将阻止他在步骤 (4) 参与任何先前发起的表决投票。因此，发起新的表决会阻止任何先前发起的表决成功。如果在先前的表决有机会成功之前，以更大的编号不断发起新的表决，那么可能无法取得任何进展。

满足进展条件需要发起新的表决，直到有一轮成功，但不能过于频繁地发起。为了得到完整协议，Paxos岛人首先必须知道信使投递消息和牧师响应需要多长时间。他们确定一位没有离开议事厅的信使总会在4分钟之内投递消息，而留在议事厅的牧师总会在某事件发生后7分钟之内采取响应动作 [原注 11. 我假设δζ∂ιφῖ的值为30秒，这是Paxos岛的计时单位。从沙漏碎片的研究可以判定该值的范围。牧师的反应时间很长，因为他们必须在7分钟（14 δζ∂ιφῖ）之内回复每一条消息，即使有许多消息同时到达]。因此，如果p和q在议事厅内，当某事件使p向q发送了消息，且q回复了p，那么p将在22分钟之内收到答复，假设双方的信使也没有离开议事厅（牧师p将在事件发生后7分钟之内发送消息，随后q在4分钟之内收到消息，在7分钟之内回复，再过4分钟之内送达p）。

假设仅有一位牧师p正在发起表决，他按协议的步骤 (1)，向每位牧师发送消息。如果p发起表决时大多数牧师都在议事厅内，那么他可以期望在发起表决后的22分钟之内执行步骤 (3)，并在另22分钟之内执行步骤 (5)。如果届时他无法执行这些步骤，那么或者某位牧师或信使在p发起表决后离开了议事厅，或者之前由另一位牧师发起了编号更大的表决（在p成为唯一发起表决的牧师之前）。为了处理后一种可能，p必须了解其他牧师使用的，比$lastTried[p]$更大的表决编号。这可以通过扩展协议来实现，要求如果牧师q收到了p发来的$NextBallot(b)$或$BeginBallot(b,d)$消息，且$b < nextBal[q]$，那么q会向p发送一条包含$nextBal[q]$的消息。随后，牧师p将用更大的编号发起新的表决。

仍然假设p是唯一发起表决的牧师，假设当且仅当满足如下的条件，他才能发起新的表决：(i) 他在此前22分钟之内没有执行步骤 (3) 或 (5)；或者 (ii) 他得知另一位牧师已经发起了编号更大的表决。如果大多数牧师都在议事厅内，而且门被p锁上了，那么在99分钟之内，一条法令将被批准并记录到议事厅内所有牧师的帐簿上。（p可能需要22分钟来发起新一轮表决，再过最多22分钟才能知道是否有另

一位牧师发起了编号更大的表决，之后成功表决需要最多 55 分钟以完成步骤 (1)~(6)）。因此，如果仅有一位牧师p发起表决，且他没有离开议事厅，就能满足进展条件。

因此，完整协议包括选择一位牧师的程序，称之为**总统**（president），由他发起表决〔译注：在不同场景，president可指院长、议长、总统等，译文统一称为总统〕。大多数政府形式中，选举总统是一个难题。然而，困难的出现是因为大多数政府要求任何时候恰好都只有一位总统。以美国为例，如果1988年大选后，有人认为布什当选了总统，而另一些人认为杜卡基斯当选了，那就造成混乱了，因为其中一位可能决定签署某项法案，而另一位可能决定否决它。然而，在Paxos岛的Synod中，拥有多位总统只会阻碍进展；这不会造成不一致。对于满足进展条件的完整协议，选举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事厅，那么T分钟之后，议事厅内恰好仅有一位牧师会认为自己是总统。

若符合总统选举要求，则完整协议将有如下性质：如果在$T + 99$分钟内，大多数牧师都在议事厅内，并且没有人进入或离开议事厅，那么之后议事厅内每位牧师的帐簿上都会记录一条法令。

Paxos岛人将议事厅内所有牧师的姓名按字母顺序，选择排在最后的一位牧师当总统，然而我们不知道确切的做法。下述做法可以满足总统选举要求：如果议事厅内的一位牧师至少每$T - 11$分钟向其他牧师发送一条包含他姓名的消息，当且仅当T分钟内没有收到来自“名单后面”牧师的消息，那么他就可以认为自己是总统了。

完整Synod协议来源于基本协议，并额外要求牧师尽快执行步骤 (2)~(6)，且增加了选举总统的方法，总统负责发起表决，并要求总统在恰当的时机发起。该协议的诸多细节尚不清楚。我已经描述了选举总统和决定总统何时发起新表决的简单方法，但这无疑不是Paxos岛人使用的方法。我给出的规则要求总统即使在选定了一条法令之后也要继续发起表决，从而确保刚刚进入议事厅的牧师了解所选定的法令。当法令被选定后，显然有更好的方法确保被牧师获知。另外，在选举总统的过程中，每位牧师或许可以向其他牧师发送其$lastTried[p]$，从而允许总统第一次尝试就可能确定一个足够大的表决编号。

Paxos岛人意识到，任何实现进展条件的协议都必然涉及测量时间的流逝 [原注12. 然而，几百年后才给出这一结论的严格证明 [Fischer et al. 1985]]。上述选举总统和发起表决的协议很容易表述为准确的算法：设定计时器，并在超时后执行动作——假设计时器完全准确。进一步分析表明，使用〔不完全准确，但〕准确度范围已知的计时器，这些协议也能完成。Paxos岛熟练的玻璃工可以毫不费力地造出合适的沙漏计时器。

鉴于Paxos数学家的高深，人们普遍认为他们一定已经找到了满足总统选举要求的最佳算法。我们只能希望未来挖掘Paxos岛会发现这个算法。

3. 多法令（Multi-Decree）议会（Parliament）

议会成立后，从Synod协议推导出了一项满足其一致性和进展要求的协议。第3.1节和第3.2节介绍初始议会协议的推导和性质。第3.3节讨论协议的进一步演变。

3.1 协议

Paxos议会必须批准一系列有编号的法令，而不是仅一条法令。如Synod协议一样，选举了一位总统。任何想要批准法令的人都会通知总统，总统会为该法令分配一个编号并试图批准它。从逻辑上讲，议会协议为每个法令编号分别使用了完整的Synod协议实例。然而，所有这些实例都选举了同一位总统，对协议的前两个步骤，他只需执行一次。

推导出议会协议的关键是观察到，Synod协议中，在步骤 (3) 之前，总统不会选定法令或Quorum。新当选的总统p可以向某些立法员发送一条消息，作为Synod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）。立法员q可以回复一条消息，作为Synod协议所有实例的步骤 (2) 中的$LastVote$消息。该消息的效用是有限的，因为q可能只对有限数量的实例投票。

收到多数集每个成员的回复后，新总统准备为Synod协议的每个实例执行步骤 (3)。对于数量有限的某些实例（法令编号），在步骤 (3) 将按$B3$确定法令。总统立即对这些这些实例执行步骤 (3)，以尝试批准这些法令。然后，每当收到批准法令的请求，他从尚未确定的法令中选取编号最小的那条，并对该法令编号（Synod协议的实例）执行步骤 (3) 以尝试批准。

对该简单协议的下述修改就得到了实际的Paxos议会协议。

- 对结果已知的法令编号，无需执行一遍Synod协议。因此，如果一位新当选的总统在他的帐簿上已经记载了所有编号小于n的法令，那么他发送一条$NextBallot(b, n)$消息，作为所有法令编号大于n的Synod协议实例中的$NextBallot(b)$消息。在对该消息的回复中，立法员q将其帐簿中存在的所有编号大于n的法令告知p（对不在帐簿上的法令，还要按常规发送$LastVote$消息），他还要求p向他发送所有编号小于n且不在q的帐簿上的法令。
- 假设第125号和第126号法令在周五下午很晚才提出，第126号法令获得批准，并被一两本帐簿记录了，但之后，所有立法员都回家过周末了。再假设接下来的周一，Δφωρκ当选为新总统，且获知了第126号法令，但她对第125号法令一无所知，因为前任总统和所有投票支持该法令的立法员仍不在议事厅内。她将主持批准第126号法令的表决，这会在帐簿上留下空白。将编号125分配给新的法令，将使新法令比上周批准的第126号法令在帐簿上更靠前。以这种方式乱序批准法令可能会引起混乱——例如，如果提议新法令的公民正是因为知道第126号法令已经批准了才提议的〔译注：此处“公民”指代立法员〕。相反，Δφωρκ会试图批准一条传统的法令

  125：2月13日是全国橄榄节

  该法令对Paxos岛的所有人都没有任何影响。一般来说，新总统会批准“橄榄节”法令来填补其帐簿中的任何空白。

议会协议的一致性和进展性质直接继承自Synod协议的相应性质。据我们所知，Paxos岛人从未费心写下议会协议的精确描述，因为它很容易从Synod协议推导出来。

3.2 协议的性质

3.2.1 法令的顺序

对多条不同法令编号的表决，可以并发进行，而且表决可能由不同的立法员发起——这些立法员发起表决时都认为自己是总统。我们无法准确得知法令将以何种顺序批准，尤其是在不知道总统是如何选出的情况下。然而，关于法令的顺序，有一个重要的性质是可以推导出来的。

在对应的Synod协议实例中，一条法令被总统在步骤 (3) 确定后，就称该法令被**提议**（proposed）。该法令只要被写入某一本帐簿后，就称它被**批准**（passed）了。在提议任何新法令之前，总统必须从多数集的所有成员获知他们已经参与投票的法令。任何已批准的法令都必然经由多数集中的至少一位立法员投票赞成。因此，在发起任何新法令之前，总统必然知晓了所有先前批准的法令。总统不会用重要的法令来填补帐簿上的空白——也就是说，只能用“橄榄节”法令填补空白。他也不会乱序提议法令。因此，该协议满足以下**法令顺序性质**。

> 如果法令A和B是重要的，且在法令B被提议之前，法令A就已经被批准了，那么法令A的编号小于B。

3.2.2 闭门议事

虽然我们不知道选举新总统涉及的细节，但我们确实知道当总统被选中，且没有人进入或离开议事厅时，议会是如何运作的。收到批准法令的请求后——无论是直接来自一位公民还是由一位立法员转交——总统为该法令分配一个编号，并交换以下消息，批准法令（以下列表的序号指Synod协议的对应步骤）。

(3) 总统向Quorum的每位立法员发送一条*BeginBallot*消息。

(4) Quorum的每位立法员向总统发送*Voted*消息。

(5) 总统向每位立法员发送*Success*消息。

这共有三个消息延迟和约$3N$条消息，假设议会有N位立法员，Quorum约为$N/2$。此外，如果议会很忙，总统会将一条法令的*BeginBallot*消息和前一条法令的*Success*合并起来，使每条法令总共只有$2N$条消息。

3.3 进一步发展

事实证明，管理该岛比Paxos岛人意识到的要复杂。很多问题冒出来了，其解决方案需要更改协议。下面介绍了最重要的部分变更。

3.3.1 选举总统

议会总统最初是按Synod使用的方法选出的，该方法完全基于姓名的字母顺序。于是，立法员Ωκι休完六个月的假期回来，立即被任命为总统了——尽管他不知道缺席期间发生了什么。Ωκι写字很慢。他费力地抄写前六个月以来的法令，以更新他的帐簿，期间议会活动暂停了。

这件事引发了一场关于总统选举的最佳方案的辩论。一些Paxos岛人鼓吹一旦立法员成为总统，他应该一直担任，直到他离开议事厅。一群有影响力的公民希望议事厅中最有钱的立法员成为总统，因为他有能力聘请更多的抄写员和其他仆人来帮助他履行总统职责。他们争辩说，一旦一位有钱的立法员把他的帐簿抄写到最新，没有理由不让他担任总统。然而，其他人则认为，无论财产如何，最正直的公民才应该成为总统。正直可能意味着不太可能不诚实，尽管没有Paxos岛人会公开承认可能渎职。不幸的是，不知道这场辩论的结果；最终使用的总统选举协议没有记载。

3.3.2 长帐簿

年复一年，议会批准的法令越来越多，Paxos岛人不得不仔细研究越来越长的法令清单，以找出当前的橄榄税或可以出售什么颜色的山羊。一位长途航行后回到议事厅的立法员，不得不大量抄写以更新他的帐簿。最终，立法员被迫将他们的帐簿从法令清单转换为法典格式，其中仅包含法律的最新状态，以及该状态对应的，最近批准的法令的编号〔译注：帐簿实体没有变，改变的是记录格式〕。

要了解当前的橄榄税，可以查看法典中的“税收”；要了解可以出售什么颜色的山羊，可以查看“商业法”。如果某位立法员的帐簿记录的法律仅到第1298号法令，且他了解到第1299号法令将橄榄税设置为每吨6德拉克马，那么他只要更改橄榄税的那项条目，且注明他的帐簿到第1299号法令都是完整的。如果他稍后了解到第1302号法令，他会把它记在帐簿背面，等他获知第1300号和第1301号法令，再随后将第1302号法令纳入法典。

为了让出短差的立法员赶上进度，又不必抄写整本法典，立法员们在帐簿背面保留了一份上周法令的清单。他们本可以将这份清单写在一张纸条上，但法令批准后，立法员将其记在帐簿背面也很方便，而且他们每周只更新两三次法典。

3.3.3 官僚

随着Paxos岛的繁荣，立法员变得非常忙碌。议会无法再处理所有的管理细节，因此建立了官僚机构。议会不再为每批奶酪批准一条法令来宣布是否合格出售，而是批准了一条法令，任命一位奶酪检查员来做出这些决定。

很快有迹象表明，派遣官员并不像最初看起来那么简单。议会批准了一条法令，任命Δĭκστρα为第一任奶酪检查员。几个月后，商家纷纷抱怨Δĭκστρα过于严格，上等的奶酪也不放行。议会随后批准了下述法令，撤换了他

> 1375：Γωυδα是新的奶酪检查员

但是Δĭκστρα并没有密切关注议会的活动，因此他没有立即得知这项法令。奶酪市场出现了一段时期混乱，当时Δĭκστρα和Γωυδα都在检查奶酪，并做出相互矛盾的决定。

为了防止这种混乱，Paxos岛人必须保证任何时候一个职位最多只能由一位官员在任。为此，总统将提议日期和时间作为每项法令的一部分。一条任命Δĭκστρα为奶酪检查员的法令可能如下

> 2716：72年1月15日 8:30 — Δĭκστρα是奶酪检查员，任期3个月

这宣布他的任期将从1月15日的8:30或前任检查员的任期结束后开始（以**较晚者**为准）。他的任期将于3月15日 8:30 结束〔译注：原文如此〕，除非他要求总统批准如下的法令，明确辞职

> 2834：72年3月3日 9:15 — Δĭκστρα辞去奶酪检查员的职务

官员的任期较短，所以可以很快被替换——例如，他可能离开了这个岛。如果他的工作令人满意，议会将批准一条法令延长他的任期。

官员需要知道时间，来确定他目前是否在任。Paxos岛上没有机械钟，但通过太阳或星星的位置，Paxos岛人可以得出在15分钟准确度之内的时间 [原注13. Paxos岛气候温暖，很少阴天]。如果Δĭκστρα的任期从8:30开始，直到他的天文观测表明到了8:45，他才会开始检查奶酪。

如果编号较大的法令总是有较晚的提议时刻，那么很容易使这种任命官员的方法奏效。但是如果议会批准了如下的法令呢？

> 2854：78年4月9日 9:45 — Φρανσεζ是品酒师，任期2个月
> 2855：78年4月9日 9:20 — Πνυελĭ是品酒师，任期1个月

这是在 9:30 到 9:35 之间由两位都自认为是总统的立法员提议的。由于议会协议满足以下性质，因此很容易避免这种乱序的提议时间。

> 如果两条法令由不同的总统批准，那么其中一位总统在得知另一条法令已被提议后，才提议他的法令。

为了确认满足此性质，假设法令D的成功表决编号为b，法令D'的成功表决编号为b'，且$b < b'$。令q是两次表决中都投了票的立法员。法令D'的表决起始于$NextBallot(b', n)$消息。如果该消息的发送人还不知道D，那么n小于法令D的编号，且q对该$NextBallot$消息的回复必然要说明他为D投过票。

3.3.4 获知法律

普通市民除了要求批准法令外，还需要查询现行的本地法律。最初Paxos岛人认为市民只需简单地查看任意立法员的帐簿，但下面的事件表明需要更复杂的方法。几百年来，只有出售白山羊是合法的。一位名叫Δωλεφ的农场主让议会批准了这项法令

> 77：允许出售黑山羊

之后Δωλεφ让他的羊倌把一群黑山羊卖给一位名叫Σκεν的商人。作为守法公民，Σκεν向立法员Στωκμεĭρ询问这笔买卖是否合法。但是Στωκμεĭρ已经离开了议事厅，且帐簿上没有第76号法令之后的

条目。他告诉Σκεν，根据现行法律，买卖是非法的，因此Σκεν拒绝购买山羊。

这一事件导致制定了以下关于法律查询的**单调性条件**。

> 如果一次查询先于另一次查询，那么第二次查询不能显示比第一次查询更旧的法律状态。

如果公民获悉某项法令已获批准，那么获取该信息的过程被视为隐式查询，也应符合上述条件。正如我们将看到的，多年来对单调性条件的解释有所变化。

最初，单调性条件是通过为每次查询批准一条法令来实现的。如果Σðνĭδερ想知道当前的橄榄税，他会让议会批准一条如下的法令

> 87：市民 Σðνĭδερ 正在阅读法律

之后他会阅读任意一本至少完整包含到第86号法令的帐簿，以了解截至该项法令的橄榄税。如果市民Γρεες随后查询橄榄税，那么他的查询法令是在第87号法令批准之后提议的，因此法令顺序性质（第3.2.1节）意味着这条法令的编号大于87。从而，Γρεες不可能获得比Σðνĭδερ更旧的橄榄税值。这种读取法律的方法满足了单调性条件，其中**先于**（precede）解释为：查询A先于查询B，等价于A的完成时刻早于B的开始时刻。

很快就表明每次查询都批准一条法令太麻烦了。Paxos岛人意识到，如果他们修改对**“先于”**的解释，弱化单调性条件，就可以有更简单的查询方法。他们决定，一个事件**先于**另一事件，则第一个事件不仅必须在更早的时刻发生，而且必须能够对第二个事件产生因果影响。较弱的单调性条件防止了首先由农场主Δωλεφ和商人Σκεν遇到的问题，因为Δωλεφ隐式查询的结束，与Σκεν查询的开始，这两个事件存在因果链。

在所有商业交易和查询中使用法令编号，就满足较弱的单调性条件了。例如，农场主Δωλεφ的羊群中有不少山羊不是白色的，他让议会批准了下面的法令

> 277：允许出售棕色山羊

当把他的棕色山羊卖给Σκεν时，他告诉这位商人，截至第277号法令，这笔交易是合法的。Σκεν随后询问立法员Στωκμεĭρ，至少截至第277号法令的法律，这笔买卖是否合法。如果Στωκμεĭρ的帐簿到第277号法令不完整，那他要么等到帐簿更新，要么让Σκεν去问别人。如果Στωκμεĭρ的帐簿一直记录到了第298号法令，那么他会告诉Σκεν，截至第298号法令，这笔买卖是合法的。商人Σκεν会记住编号298，以便在他下一次商业交易或法律查询中使用。

Paxos岛人保证了单调性条件，但普通市民不喜欢必须记住法令编号。再一次，Paxos岛人通过重新解释单调条件解决了这个问题——这一次，修改了**法律状态**的含义。他们将法律划分为不同的领域，每个领域确定一位立法员作为专家。每个领域法律的当前状态由该专家的帐簿确定。例如，假设第1517号法令修改了**关税法**，第1518号法令修改了**税法**。如果税法专家在关税法专家之前已经获知了两条法令，那么税法可能将先被修改，从而产生无法通过编号顺序获取的法律状态〔译注：指第1517号关税法令。如果关税法专家工作正常，那么这种异常是暂时的〕。

为了避免对当前法规的定义产生冲突，Paxos岛人要求任意领域同时最多只能有一位专家。通过使用与确定官员相同的方法来确定专家（见第3.3.3节），可以满足这一要求。如果每次查询只涉及单一领域的法律，那么将查询交给该领域的专家，由他基于自己的帐簿来回应，就可以实现单调性。由于“获知一条法律已批准”即是一次隐式查询的结果，Paxos岛人要求一条法令最多修改一个法律领域，并且只能由该领域的专家来通知法令被批准了。

涉及多个领域的查询不难处理。当商人Λισκωφ询问进口金羊毛的关税是否高于本地购买的销售税时，税法和关税法专家不得不合作以提供答案。例如，税法专家可以先问关税专家金羊毛的关税，从而回答Λισκωφ，只要税法专家在收到答复之前不改动其帐簿。

这种方法被证明是令人满意的，直到不得不一次性彻底修改几个法律领域。那时Paxos岛人意识到保持单调性的必要条件不是一条法令只能影响一个领域，而是该法令影响的所有领域应该有同一位专家。为了用一条法令修改多个法律领域，议会可以首先任命一位立法员作为所有这些领域的专家。此外，同一领域可以有多位专家，只要该领域的法律不允许修改。临近所得税期，议会将任命多位税法专家来处理涌入的季节性税法查询。

3.3.5 不诚实的立法员和诚实的错误

尽管官方的说法相反，但在Paxos的历史上肯定有一些不诚实的立法员。被抓到后，他们很可能被流放了。通过发送相互矛盾的消息，恶意的立法员可能导致不同立法员的帐簿不一致。诚实的立法员或信使记忆疏忽也可能导致不一致。

发现不一致后，可以通过批准法令轻松纠正。例如，为了消除对当前橄榄税的分歧，可以通过批准一条新法令宣布该税的值。困难的问题是纠正不一致的帐簿，即使还没有人意识到不一致。

从议会成立几年后就开始出现在帐簿上的冗余法令，可以推断出存在不诚实或出错的立法员。例如，下面的法令被批准了

> 2605：橄榄税为每吨9德拉克马

尽管第2155号法令已经将橄榄税定为每吨9德拉克马，并且中间没有任何法令修改过它。议会显然每六个月循环一遍法律，因此即使最初立法员的帐簿不一致，所有立法员也会在六个月内就该地的现行法律达成一致。人们相信，通过使用这些冗余的法令，Paxos岛人使他们的议会**自稳定**（self-stabilizing，自稳定是一个现代术语，由Dijkstra[1974]提出）。

在立法员随意进出的议会中，自稳定究竟意味着什么尚不清楚。如果是指达成一致前要求所有立法员都在议事厅，这不会让Paxos岛人满意。但是，为了实现一致性，如果一位立法员的帐簿上有某项法令编号的记录，而另一位立法员没有，那么后者应该最终能填上该项记录。

不幸的是，我们不知道Paxos议会拥有什么样的自稳定性质，也不知道是如何实现的。Paxos岛的数学家们无疑解决了这个问题，但他们的工作还没有被发现。我希望未来对Paxos岛的考古考察优先寻找关于自稳定的手稿。

3.3.6 确定新的立法员

起初，议会的成员资格是世袭的，父传子。年老的政治家Παρνας退休后，他将自己的帐簿交给了儿子，儿子继续工作。对与Παρνας交流的其他立法员而言，这没有任何变化。

随着旧家族迁出，新家族迁入，这个制度不得不改变。Paxos岛人决定使用法令增减议会成员。这是一个循环问题：议会成员资格取决于批准的法令，但批准法令需要知道多数集成员，而这又取决于谁是议会成员。在截至$n-3$号法令的法律中，明确了用以批准第n号法令的议会成员，从而打破了循环。总统在获知截至第3252号法令之前，不能试图批准第3255号法令。实际上，在批准下述法令后

> 3252：Στρωνγ现在是立法员了

总统将立即批准“橄榄节”法令作为第3253号和第3254号法令。

以这种方式改变议会的组成很危险，必须谨慎行事。一致性和进展条件应当始终成立。然而，只有在议事厅内有多数集时，进展条件才能保证；并不能保证多数集一直存在。事实上，确定立法员的机制导致了Paxos议会制度的灭亡。由于抄写员的错误，一条原本是纪念海难中溺水水手的法令，却误将他们宣布为仅有的议会成员。这条法令的批准阻止了批准任何新法令——包括提议纠正该错误的法令。Paxos岛的统治陷入停顿。一位名叫Λαμπσων的将军趁机发动政变，建立了军事独裁，终结了数百年的开明统治。Paxos岛在一系列腐败独裁者的治下日渐虚弱，无力反抗东方的入侵者，导致其文明覆灭了。

4. 与计算机科学的关联

4.1 状态机方法

尽管Paxos岛的议会在几百年前就被摧毁了，但它的协议仍然有用。例如，考虑一个可能用作名字服务器的简单分布式数据库系统。数据库的状态涉及写入名字的值。数据库的副本由多个服务器维护。客户端程序可以向任何服务器发出请求，读取或修改名字的值。有两种读请求：**慢读**（slow read），返回名字最新的值，以及**快读**（fast read），速度更快但可能不会反映最近对数据库的更改。

这个数据库系统和Paxos议会之间有明显的对应关系：客户端修改值的请求是通过批准法令来执行的。**慢读**需要批准法令，如在第3.3.4节中所述。**快读**是通过读取服务器当前版本的数据库执行的。Paxos议会协议提供了数据库系统的分布、容错实现。

议会		分布式数据库
立法员	↔	服务器
市民	↔	客户端程序
当前法律	↔	数据库状态

这种实现分布式数据库的方法是状态机方法的一个样例，首先在 Lamport [1978] 中提出。这种方法中，首先定义一个**状态机**（state machine），它由状态集合、命令集合、响应集合和一个函数组成，函数将“响应/状态”对（由响应和状态组成的一对）关联到每组“命令/状态”对。直观地说，状态机执行命令是指产生响应并改变内部状态；命令和状态机的当前状态决定了它的响应和新状态。对于分布式数据库，状态机的状态就对应数据库状态。图2描述了状态机命令，及确定响应和新状态的函数。

命令：	**read**(*name, client*)	**update**(*name, val, client*)
响应：	(*client, name*的值)	(*client*, "ok")
新状态：	与原状态相同	除了*name*的值改为*val*，其它与原状态相同

图 2. 简单数据库的状态机

状态机方法中，系统是由网络连接的服务进程实现的。服务器将客户端请求转换为状态机命令，执行命令，并将状态机响应转换为对客户端的回复。这是一个通用算法，确保所有服务器获得相同的命令序列，从而确保它们都产生相同的响应序列和状态变化——假设它们的初始状态都相同。对数据库的示例，执行**慢读**或修改值的客户端请求被转换为状态机**读取**或**更新**命令。执行命令后，状态机的响应转换成对客户端的响应，由接收到请求的服务器发送给客户端。由于所有服务器执行相同的状态机命令序列，它们都维护数据库的一致版本。然而，任何时候，总会有些服务器的版本可能落后于其它服务器的版本，因为状态机命令不需要所有服务器同时执行。服务器使用其当前版本的状态来响应**快读**请求，而无需执行状态机命令。

系统的功能由状态机来表达，它只是一个从“命令/状态”对到“响应/状态”对的函数。同步和容错问题由服务器获取命令序列的通用算法处理。设计新系统时，只有状态机是新的。服务器由标准的分布式算法获取状态机命令，该分布式算法已被证明是正确的。与分布式算法相比，函数更容易设计和正确处理。

实现任意状态机的第一个算法出现在Lamport [1978]。后来，[Lamport 1984]设计了容忍不超过f个进程任意类型故障的算法。这些算法保证，如果失效进程少于f个，那么可以在有限时间内执行完状态机命令。因此，这些算法适用于需要实时响应的应用 [原注14. 这些算法来自另一个地中海国家的军事协议]。但如果超过f个进程故障了，那么不同的服务器可能会有不一致的状态机副本。此外，两台服务器无法通信，相当于其中一台发生故障。要降低丧失一致性的概率，系统必须使算法的f值较大，这意味着冗余硬件、通信带宽和响应时间带来的大量成本。

Paxos议会协议提供了另一种实现通用状态机的方法。立法员的法典对应机器状态，批准法令对应执行状态机命令。与早期的算法相比，由此产生的算法不够健壮，但成本较低。它不容忍任意类型、恶意的故障，也不保证在有限时间内响应。然而，即使任意数量的进程和信道出现（良性）故障，仍能

保持一致性。Paxos算法适用于具有适度可靠性要求的系统，这些系统不能承受高度容错且实时实现方案的成本。

如果状态机是用确保响应时间有界的算法来执行的，那么时间可以成为状态的一部分，状态机的动作可以随着时间的推移而触发。例如，考虑一个授予资源所有权的系统。状态可以包括客户端被授予资源的时刻，如果客户端持有该资源的时间过长，状态机可以自动执行命令以收回所有权。

使用Paxos算法，时间不能以这种自然的方式成为状态的一部分。如果故障发生了，执行命令（批准法令）可能需要任意长的时间，并且一个命令可能比另一个更早发出的命令先执行（在法令序列中出现得更靠前）。但是，状态机仍然可以像Paxos议会那样使用物理时间。例如，第3.3.3节中描述的用于确定谁是当前奶酪检查员的方法，可用于确定当前谁是资源的所有者。

4.2 提交协议

Paxos的Synod协议类似于标准的三阶段提交协议（3PC）[Bernstein 1987; Skeen 1982]。Paxos表决和三阶段提交协议都涉及协调员（总统）和其他Quorum成员（立法员）之间的五条消息交换。提交协议选择两个值之一——**提交**（commit）或**中止**（abort）——而Synod协议选择任意法令。要将提交协议转换为Synod协议，需要在第一轮消息中发送法令。**提交**决定意味着这项法令获得批准，而**中止**决定意味着批准“橄榄节”法令。

Synod协议与转换后的提交协议不同，因为直到第二阶段才会发送法令。这允许相应的议会协议对所有法令只执行一次第一阶段，因此只需要交换三条消息即可批准一项法令。

Synod协议所基于的定理与Dwork、Lynch和Stockmeyer获得的结果相似[Dwork et al. 1988]。然而，他们的算法在不同的轮次中顺序地执行表决，这似乎与Synod协议无关。

> 自本文定稿以来，该领域已经发表了大量研究。[Schneider 1990]是状态机方法的综述。Keidar和Dolev[1996]的恢复协议以及Fekete等[1997]的全序广播算法与本文描述的Paxos协议非常相似。作者显然也没有注意到Oki和Liskov[1988]的视图管理协议，该协议似乎与Paxos协议相同。
>
> 本文中提出的许多改进也出现在当时或后续文章中。第3.3.3节中描述的代理方法与Gray和Cheriton[1989]的租约机制非常相似。Ladin等[1992]描述了第3.3.4节中Paxos岛人使用法令编号满足单调性条件的技术。Schneider[1990]也给出了第3.3.6节中添加新立法员的技术。
>
> K. M.

致谢

Daniel Duchamp向我指出需要一个新的状态机实现。与Martín Abadi、Andy Hisgen、Tim Mann和Garret Swart的讨论使我想到了Paxos。Λεωνῖδας Γκῖμπας提供了关于Paxos方言的宝贵帮助。

参考文献

[1]. Bernstein, P. A., Hadzilacos, V., and Goodman, N. 1987. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Longman Publ. Co., Inc., Reading, MA.

[2]. De Prisco, R., Lampson, B., and Lynch, N. 1997. Revisiting the Paxos Algorithm. In *Proceedings of the 11th International Workshop on Distributed Algorithms*, M. Mavronicolas and P. Tsigas, Eds., LNCS, vol. 1320. Springer-Verlag, Berlin, Germany, 111-125.

[3]. Dijkstra, E. W. 1974. Self-stabilizing Systems in spite of Distributed Control. *Commun. ACM* 17, 11, 643-644.

[4]. Dwork, C., Lynch, N., and Stockmeyer, L. 1988. Consensus in the Presence of Partial Synchrony. *J. ACM* 35, 2 (Apr.), 288-323.

[5]. Fekete, A., Lynch, N., and Shvartsman, A. 1997. Specifying and Using a Partitionable Group Communication Service. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing*. ACM Press, New York, NY, 53-62.

[6]. Fischer, M. J., Lynch, N. A., and Paterson, M. S. 1985. **Impossibility of Distributed Consensus with One Faulty Process.** *J. ACM* 32, 1 (Jan.), 374-382.

[7]. Gray, C. and Cheriton, D. 1989. Leases: An Efficient Fault-tolerant Mechanism for Distributed File Cache Consistency. *SIGOPS Oper. Syst. Rev.* 23, 5 (Dec. 3-6), 202-210.

[8]. Keidar, I. and Dolev, D. 1996. Efficient Message Ordering in Dynamic Networks. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*. ACM Press, New York, NY.

[9]. Ladin, R., Liskov, B., Shrira, L., and Ghemawat, S. 1992. Providing High Availability using Lazy Replication. *ACM Trans. Comput. Syst.* 10, 4 (Nov.), 360-391.

[10]. Lamport, L. 1978. **Time, Clocks, and the Ordering of Events in a Distributed System.** *Commun. ACM* 21, 7, 558-565.

[11]. Lamport, L. 1984. Using Time instead of Timeout for Fault-tolerant Distributed Systems. *ACM Trans. Program. Lang. Syst.* 6, 2 (Apr.), 254-280.

[12]. Lampson, B. W. 1996. How to Build a Highly Available System using Consensus. In *Distributed Algorithms*, O. Babaoglu and K. Marzullo, Eds. Springer LNCS, vol. 1151. Springer-Verlag, Berlin, Germany, 1-17.

[13]. Oki, B. M. and Liskov, B. H. 1988. Viewstamped Replication: A General Primary Copy. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing* (Toronto, Ontario, August 15-17, 1988). ACM Press, New York, NY, 8-17. 〔译注：另参见 ***Viewstamped Replication Revisited*, 2012** 和 **译文**.〕

[14]. Schneider, F. B. 1990. Implementing Fault-tolerant Services using the State Machine Approach: A Tutorial. *ACM Comput. Surv.* 22, 4 (Dec.), 299-319.

[15]. Skeen, M. D. 1982. Crash Recovery in a Distributed Database System. Ph.D. Thesis. University of California at Berkeley, Berkeley, CA.

〔译注：1989年技术报告中的参考文献是上面列表的第1, 3, 4, 6, 10, 11 和 15项。〕

附录: Synod 协议一致性的证明

A1. 基本协议

Synod 的基本协议，在第2.3节中已做非形式化的描述，此处将使用现代算法符号进行陈述。我们从每位牧师p必须维护的变量开始。首先是代表记录在帐簿中信息的变量。 (为方便起见，第2.3节中使用的投票 $prevVote[p]$ 被其组成部分 $prevBal[p]$ 和$prevDec[p]$ 所替代。)

变量	描述
$outcome[p]$	记录在p的帐簿上的法令，若尚未记录任何内容则为BLANK。
$lastTried[p]$	p试图发起的最后一轮表决的编号，若没有则为$-\infty$。
$prevBal[p]$	p参与投票的最后一轮表决的编号，若从未投票则为$-\infty$。
$prevDec[p]$	p最近一次投票的法令，若从未投票则为BLANK。
$nextBal[p]$	p同意参与的最后一轮表决的编号，若从未同意参与任何表决则为$-\infty$。

接下来是代表牧师p可以记在一张纸条上的信息的变量：

变量	描述
$status[p]$	以下值之一： *idle* 未在执行或尝试发起一轮表决 *trying* 正在尝试发起编号为 $lastTried[p]$ 的表决 *polling* 正在执行编号为 $lastTried[p]$ 的表决 如果p丢失了他的纸条，那么 $status[p]$ 被假定为 *idle*，并且下面四个变量的值无关紧要。
$prevVotes[p]$	对于当前表决 (即编号为 $lastTried[p]$ 的表决)，在 $LastVote$ 消息中收到的投票集合。
$quorum[p]$	若 $status[p] = polling$，则为当前表决的 Quorum 成员 (牧师集合)；否则无意义。
$voters[p]$	若 $status[p] = polling$，则为在当前表决中已收到其 $Voted$ 消息的 Quorum 成员集合；否则无意义。
$decree[p]$	若 $status[p] = polling$，则为当前表决的法令；否则无意义。

还有一个历史变量 $\mathcal{B}$，它是已开始的表决及其进展的集合——即哪些牧师已经投票。 (历史变量是在算法的开发和证明中使用的变量，但实际上并未在实现中出现。)

接下来是牧师p可能采取的动作。这些动作被假定为**原子**的，意味着一个动作一旦开始，必须在牧师p开始任何其他动作之前完成。一个动作由一个**生效条件** 和一个**动作列表**描述。生效条件描述了何时可以执行该动作；接收消息的动作在信使带着相应消息到达时即生效。动作列表描述了该动作如何改变算法的变量，以及它发送什么消息 (如果有的话)。 (每个独立动作最多发送一条消息。)

回想一下，表决编号在牧师之间是划分好的。对于任何表决编号b，Paxos 岛人定义 $owner(b)$ 为被允许使用该表决编号的牧师。基本协议中的动作都是**允许的动作** ；协议不要求牧师必须做任何事。这里没有做任何效率上的尝试；这些动作允许p做一些傻事，比如向一个已经收到其 $LastVote$ 消息的牧师再次发送 $BeginBallot$ 消息。

尝试新表决

- **生效条件**: 始终生效。
- **动作**:
  - 将 $lastTried[p]$ 设置为任何大于其旧值且满足 $owner(b) = p$ 的表决编号b。
  - 将 $status[p]$ 设置为 $trying$。
  - 将 $prevVotes[p]$ 设置为 $\varnothing$。

发送 $NextBallot$ 消息

- **生效条件**: 当 $status[p] = trying$ 时生效。
- **动作**:
  - 向任何牧师发送一条 $NextBallot(lastTried[p])$ 消息。

接收 $NextBallot(b)$ 消息

若 $b \geq nextBal[p]$，则

- 将 $nextBal[p]$ 设置为b。

发送 $LastVote$ 消息

- **生效条件**: 当 $nextBal[p] > prevBal[p]$ 时生效。
- **动作**:
  - 向牧师 $owner(nextBal[p])$ 发送一条 $LastVote(nextBal[p], v)$ 消息，其中 $v.pst = p$，$v.bal = prevBal[p]$，且 $v.dec = prevDec[p]$。

接收 $LastVote(b, v)$ 消息

若 $b = lastTried[p]$ 且 $status[p] = trying$，则

- 将 $prevVotes[p]$ 设置为其原值与 $\{v\}$ 的并集。

以多数集 Q 开始投票

- **生效条件**: 当 $status[p] = trying$ 且 $Q \subseteq \{v.pst : v \in prevVotes[p]\}$，其中 Q 是一个多数集时生效。
- **动作**:
  - 将 $status[p]$ 设置为 $polling$。
  - 将 $quorum[p]$ 设置为 Q。
  - 将 $voters[p]$ 设置为 $\varnothing$。
  - 将 $decree[p]$ 设置为按如下方式选择的法令 d：令 v 为 $prevVotes[p]$ 中的最大元素。若 $v.bal \neq -\infty$，则 $d = v.dec$，否则 d 可以是任何法令。
  - 将 $\mathcal{B}$ 设置为其原值与 $\{B\}$ 的并集，其中 $B.dec = d$， $B.qrm = Q$， $B.vot = \varnothing$，且 $B.bal = lastTried[p]$。

发送 $BeginBallot$ 消息

- **生效条件**: 当 $status[p] = polling$ 时生效。
- **动作**:
  - 向 $quorum[p]$ 中的任何牧师发送一条 $BeginBallot(lastTried[p], decree[p])$ 消息。

接收 $BeginBallot(b, d)$ 消息

若 $b = nextBal[p] > prevBal[p]$，则

- 将 $prevBal[p]$ 设置为 b。

- 将 $prevDec[p]$ 设置为 d。
- 若在 $\mathcal{B}$ 中存在一个 $B.bal = b$ 的表决 B [一定会有]，则选择任何这样的 B [只会有一个]，并通过将其旧值中的 $B.vot$ 设置为旧值与 $\{p\}$ 的并集，来获得 $\mathcal{B}$ 的新值。

发送 $Voted$ 消息

- **生效条件**: 当 $prevBal[p] \neq -\infty$ 时生效。
- **动作**:
  - 向 $owner(prevBal[p])$ 发送一条 $Voted(prevBal[p], p)$ 消息。

接收 $Voted(b, q)$ 消息

若 $b = lastTried[p]$ 且 $status[p] = polling$，则

- 将 $voters[p]$ 设置为其原值与 $\{q\}$ 的并集。

成功

- **生效条件**: 当 $status[p] = polling$， $quorum[p] \subseteq voters[p]$，且 $outcome[p] = \text{BLANK}$ 时生效。
- **动作**:
  - 将 $outcome[p]$ 设置为 $decree[p]$。

发送 $Success$ 消息

- **生效条件**: 当 $outcome[p] \neq \text{BLANK}$ 时生效。
- **动作**:
  - 向任何牧师发送一条 $Success(outcome[p])$ 消息。

接收 $Success(d)$ 消息

若 $outcome[p] = \text{BLANK}$，则

- 将 $outcome[p]$ 设置为 d。

这个算法是 Paxos 牧师执行的真实协议的抽象描述。该算法的动作能准确地模拟真实牧师的动作吗？一个牧师可以“原子地”执行三种动作：接收消息、写笔记或帐簿条目、以及发送消息。除了 *Receive* 动作既接收消息又设置变量外，这些动作中的每一种都由算法中的单个动作表示。我们可以假装消息的接收发生在牧师对消息采取行动的时候；如果他在行动前离开了议事厅，那么我们可以假装消息从未被接收。由于这种假装不影响一致性条件，我们可以从该算法的一致性推断出基本 Synod 协议的一致性。

A2. 一致性的证明

$$
\begin{aligned}
&I1(p) \triangleq \qquad \text{[Associated variable: } outcome[p]\text{]}\\
&\quad (outcome[p] \neq \text{BLANK}) \Rightarrow \exists B \in \mathcal{B} : (B.qrm \subseteq B.vot) \wedge (B.dec = outcome[p])
\end{aligned}
$$

$$
\begin{aligned}
&I2(p) \triangleq \qquad \text{[Associated variable: } lastTried[p]\text{]}\\
&\quad \wedge\ owner(lastTried[p]) = p\\
&\quad \wedge\ \forall B \in \mathcal{B} : (owner(B.bal) = p) \Rightarrow\\
&\qquad\qquad \wedge\ B.bal \leq lastTried[p]\\
&\qquad\qquad \wedge\ (status[p] = trying) \Rightarrow (B.bal < lastTried[p])
\end{aligned}
$$

$$
\begin{aligned}
&I3(p) \triangleq \qquad \text{[Associated variables: } prevBal[p], prevDec[p], nextBal[p]\text{]}\\
&\quad \wedge\ prevBal[p] = MaxVote(\infty, p, \mathcal{B}).bal\\
&\quad \wedge\ prevDec[p] = MaxVote(\infty, p, \mathcal{B}).dec\\
&\quad \wedge\ nextBal[p] \geq prevBal[p]
\end{aligned}
$$

$$
\begin{aligned}
&I4(p) \triangleq \qquad \text{[Associated variable: } prevVotes[p]\text{]}\\
&\quad (status[p] \neq idle) \Rightarrow\\
&\qquad \forall v \in prevVotes[p] : \wedge\ v = MaxVote(lastTried[p], v.pst, B)\\
&\qquad\qquad\qquad\qquad \wedge\ nextBal[v.pst] \geq lastTried[p]
\end{aligned}
$$

$$
\begin{aligned}
&I5(p) \triangleq \qquad \text{[Associated variables: } quorum[p], voters[p], decree[p]\text{]}\\
&\quad (status[p] = polling) \Rightarrow\\
&\qquad \wedge\ quorum[p] \subseteq \{v.pst : v \in prevVotes[p]\}\\
&\qquad \wedge\ \exists B \in \mathcal{B} : \wedge\ quorum[p] = B.qrm\\
&\qquad\qquad\qquad\quad \wedge\ decree[p] = B.dec\\
&\qquad\qquad\qquad\quad \wedge\ voters[p] \subseteq B.vot\\
&\qquad\qquad\qquad\quad \wedge\ lastTried[p] = B.bal
\end{aligned}
$$

$$
\begin{aligned}
&I6 \triangleq \qquad \text{[Associated variable: } \mathcal{B}\text{]}\\
&\quad \wedge\ B1(\mathcal{B}) \wedge B2(\mathcal{B}) \wedge B3(\mathcal{B})\\
&\quad \wedge\ \forall B \in \mathcal{B} : B.qrm \text{ is a majority set}
\end{aligned}
$$

$$
\begin{aligned}
&I7 \triangleq \qquad \text{[Associated variable: } \mathcal{M}\text{]}\\
&\quad \wedge\ \forall NextBallot(b) \in \mathcal{M} : (b \leq lastTried[owner(b)])\\
&\quad \wedge\ \forall LastVote(b, v) \in \mathcal{M} : \wedge\ v = MaxVote(b, v.pst, \mathcal{B})\\
&\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ nextBal[v.pst] \geq b\\
&\quad \wedge\ \forall BeginBallot(b, d) \in\ : \exists B \in \mathcal{B} : (B.bal = b) \wedge (B.dec = d)\\
&\quad \wedge\ \forall Voted(b, p) \in \mathcal{M} : \exists B \in \mathcal{B} : (B.bal = b) \wedge (p \in B.vot)\\
&\quad \wedge\ \forall Success(d) \in \mathcal{M} : \exists p : outcome[p] = d \neq \text{BLANK}
\end{aligned}
$$

为了证明一致性条件，必须证明只要 $outcome[p]$ 和 $outcome[q]$ 都不等于 BLANK，它们就是相等的。一个严格的正确性证明需要对算法进行完整的描述。上面给出的描述几乎是完整的。缺少的是一个变量 M，其值是所有传输中消息的多重集（multiset）〔多重集（multiset）是允许存在重复值的集合〕。每个 $Send$ 动作向这个多重集添加一条消息，每个 $Receive$ 动作移除一条。还需要代表消息丢失和重复的动作，以及一个代表牧师丢失纸条的 $Forget$ 动作。加上这些补充，我们得到了一个定义了一组可能行为的算法，其中状态的每次改变都对应于一个允许的动作。

Paxos 岛人通过找到一个谓词（predicate）I来证明正确性，该谓词满足：

1. I在初始状态为真。
2. I蕴含了所需的一致性条件。
3. 每个允许的动作都使I保持为真。

谓词I被写成一个合取式 $I1 \wedge \ldots \wedge I7$，其中 $I1 - I5$ 又分别是所有牧师p的谓词 $I1(p) - I5(p)$ 的合取。尽管大多数变量在多个合取项中被提及，但除了 $status[p]$ 外，每个变量都自然地与一个合取项相关联，每个合取项可以被看作是对其关联变量的一个约束。I的各个合取项的定义如下，其中用 $\wedge$ 符号标记的项目列表表示这些项目的合取。与合取项关联的变量列在方括号注释中。

Paxos 岛人必须证明I满足上述三个条件。第一个条件，即I初始为真，需要检查每个合取项对于所有变量的初始值是否为真。虽然没有明确说明，但这些初始值可以从变量的描述中推断出来，检查第一个条件是直接了当的。第二个条件，即I蕴含一致性，由 $I1$、$I6$ 的第一个合取项和定理1 得出。困难的部分是证明第三个条件，即I的**不变性**，这意味着要证明每个允许的动作都会使I保持为真。这个条件通过证明对于I的每个合取项，当I为真时执行任何动作都会使该合取项保持为真来证明。证明的简述如下。

- $I1(p)$: $\mathcal{B}$ 的改变只会是增加一个新的表决或向某个 $B \in \mathcal{B}$ 的 $B.vot$ 中增加一个新的牧师，这两者都不会使 $I1(p)$ 为假。$outcome[p]$ 的值仅由 $Succeed$ 和 收到$Success$消息 动作改变。生效条件和 $I5(p)$ 蕴含了 $Succeed$ 动作会使 $I1(p)$ 保持为真。生效条件、$I1(p)$ 和 $I7$ 的最后一个合取项蕴含了 收到$Success$消息 动作会使 $I1(p)$ 保持为真。
- $I2(p)$: 该合取项仅依赖于 $lastTried[p]$、$status[p]$ 和 $\mathcal{B}$。只有 $TryNewBallot$ 动作会改变 $lastTried[p]$，也只有该动作能将 $status[p]$ 设置为 $trying$。由于该动作将 $lastTried[p]$ 增加到一个满足 $owner(b) = p$ 的值b，它会使 $I2(p)$ 保持为真。一个全新的元素仅由 开始投票 动作添加到 $\mathcal{B}$ 中；$I2(p)$ 的第一个合取项和该动作的规范蕴含了添加这个新元素不会使 $I2(p)$ 的第二个合取项为假。$\mathcal{B}$ 改变的唯一其他方式是向某个 $B \in \mathcal{B}$ 的 $B.vot$ 中增加一个新牧师，这不影响 $I2(p)$。
- $I3(p)$: 由于投票永不从 $\mathcal{B}$ 中移除，唯一能改变 $MaxVote(\infty, p, B)$ 的动作是向 $\mathcal{B}$ 中添加一个由p投出的票。只有 收到$BeginBallot$消息 动作能做到这一点，也只有该动作会改变 $prevBal[p]$ 和 $prevDec[p]$。$I7$ 的 $BeginBallot$ 合取项蕴含了该动作确实向 $\mathcal{B}$ 中添加了一票，而 $B1(\mathcal{B})$ ($I6$ 的第一个合取项) 蕴含了只有一个表决可以被添加这一票。生效条件、执行动作前 $I3(p)$ 为真的假设以及 $MaxVote$ 的定义，共同蕴含了该动作会使 $I3(p)$ 的前两个合取项保持为真。第三个合取项保持为真，因为 $prevBal[p]$ 仅通过被设置为 $nextBal[p]$ 来改变，而 $nextBal[p]$ 永不减少。
- $I4(p)$: 该合取项仅依赖于 $status[p]$、$prevVotes[p]$、$lastTried[p]$、某些牧师 q 的 $nextBal[q]$ 和 $\mathcal{B}$ 的值。$status[p]$ 的值从 $idle$ 变为非 $idle$ 仅由 尝试新表决 动作完成，该动作将 $prevVotes[p]$ 设置为 $\varnothing$，使得 $I4(p)$ 无意义地为真。改变 $prevVotes[p]$ 的其他唯一动作是 $Forget$ 动作 (它通过将 $status[p]$ 设为 $idle$ 使 $I4(p)$ 保持为真) 和 收到$LastVote$消息 动作。从生效条件和 $I7$ 的 $LastVote$ 合取项可以得出 收到$LastVote$消息 动作会保持 $I4(p)$。$lastTried[p]$ 的值仅由 尝试新表决 动作改变，该动作通过将 $status[p]$ 设为 $trying$ 使 $I4(p)$ 保持为真。$nextBal[q]$ 的值只能增加，这不会使 $I4(p)$ 为假。最后，$MaxVote(lastTried[p], v.pst, \mathcal{B})$ 仅在 $v.pst$ 被添加到某个

$B \in \mathcal{B}$ 且 $B.bal < lastTried[p]$ 的 $B.vot$ 中时才会改变。但 $v.pst$ 被添加到 $B.vot$ (通过 收到 $BeginBallot$ 消息) 的条件是 $nextBal[v.pst] = B.bal$，在这种情况下 $I4(p)$ 蕴含了 $B.bal \geq lastTried[p]$。

- $I5(p)$: $status[p]$ 的值仅由 开始投票 动作设置为 $polling$。该动作的生效条件保证了第一个合取项为真，并且它向 $\mathcal{B}$ 中添加的表决使得第二个合取项为真。没有其他动作会在保持 $status[p]$ 为 $polling$ 的同时改变 $quorum[p]$、$decree[p]$ 或 $lastTried[p]$。$prevVotes[p]$ 的值在 $status[p] = polling$ 时不能改变，而 $\mathcal{B}$ 的改变只会是增加新元素或向 $B.vot$ 中增加新牧师。使 $I5(p)$ 为假的唯一剩下可能是由 收到$Voted$消息 动作向 $voters[p]$ 中添加一个新元素。$I7$ 的 $Voted$ 合取项、$B1(\mathcal{B})$ ($I6$ 的第一个合取项) 和动作的生效条件共同蕴含了被添加到 $voters[p]$ 的元素在 $B.vot$ 中，其中 B 是 $I5(p)$ 中断言其存在的那个表决。
- $I6$: 由于对于任何 $B \in \mathcal{B}$，$B.bal$ 和 $B.qrm$ 永不改变，$B1(\mathcal{B})$、$B2(\mathcal{B})$ 和 $I6$ 的第二个合取项能被证伪的唯一方式是向 $\mathcal{B}$ 中添加一个新的表决，这仅在 $status[p]$ 等于 $trying$ 时由 多数集Q开始投票 动作完成。从 $I2(p)$ 的第二个合取项可以得出该动作会使 $B1(\mathcal{B})$ 保持为真；并且，生效条件中关于 Q 是一个多数集的断言蕴含了该动作会使 $B2(\mathcal{B})$ 和 $I6$ 的第二个合取项保持为真。有两种可能的方式可以证伪 $B3(\mathcal{B})$：通过向 $\mathcal{B}$ 中添加一个新投票来改变 $MaxVote(B.bal, B.qrm, \mathcal{B})$，以及向 $\mathcal{B}$ 中添加一个新的表决。新投票仅由 收到$BeginBallot$消息 动作添加，而 $I3(p)$ 蕴含了该动作添加的投票晚于 p 在 $\mathcal{B}$ 中投出的任何其他票，因此它不能改变任何 $B \in \mathcal{B}$ 的 $MaxVote(B.bal, B.qrm, \mathcal{B})$。合取项 $I4(p)$ 蕴含了由 开始投票 动作添加的新表决不会证伪 $B3(\mathcal{B})$。
- $I7$: 若证伪 $I7$，要么通过向 $\mathcal{M}$ 中添加一条新消息，要么通过改变 $I7$ 所依赖的其他变量的值。由于 $lastTried[p]$ 和 $nextBal[p]$ 永不减少，改变它们不会使 $I7$ 为假。由于 $outcome[p]$ 在其值不为 BLANK 时永不改变，改变它不会证伪 $I7$。由于 $\mathcal{B}$ 仅通过添加表决和添加投票来改变，唯一能使 $I7$ 为假的改变是通过 $v.pst$ 添加一票，这票通过改变 $MaxVote(b, v.pst, \mathcal{B})$ 使 $LastVote(b, v)$ 合取项为假。这仅在 $v.pst$ 在一个 $B.bal < b$ 的表决 B 中投票时才会发生。但 $v.pst$ 只能在编号为 $nextBal[v.pst]$ 的表决中投票，并且该合取项初始为真的假设蕴含了 $nextBal[v.pst] \geq b$。因此，我们只需检查每条被发送的消息是否满足 $I7$ 中相应合取项的条件。

- $NextBallot$: 从 发送$NextBallot$消息 动作的定义和 $I2(p)$ 的第一个合取项得出。
- $LastVote$: 发送 $LastVote$ 消息 动作的生效条件和 $I3(p)$ 蕴含了 $MaxVote(nextBal[p], p, \mathcal{B}) = MaxVote(\infty, p, \mathcal{B})$，由此得出该动作发送的 $LastVote$ 消息满足 $I7$ 中的条件。
- $BeginBallot$: 从 $I5(p)$ 和 发送$BeginBallot$ 动作的定义得出。
- $Voted$: 从 $I3(p)$、$MaxVote$ 的定义和 发送$Voted$消息 动作的定义得出。
- $Success$: 从 发送$Success$消息 的定义得出。

1990年1月收到；1998年3月录用