软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*, [doi: 10.13328/j.cnki.jos.005964]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

# PaxosStore 中共识算法 TPaxos 的推导、规约与精化*

易星辰[1], 魏恒峰[1], 黄宇[1], 乔磊[2], 吕建[1]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

[2](北京控制工程研究所,北京 100190)

通讯作者: 魏恒峰, E-mail: hfwei@nju.edu.cn

**摘 要**: PaxosStore 是腾讯开发的高可用分布式存储系统,现已用于全面支持微信核心业务.PaxosStore 实现了分布式共识协议 Paxos 的一种变体,我们称之为 TPaxos.TPaxos 的新颖之处在于它的"统一性":它为每个参与者维护统一的状态类型,并采用统一格式的消息进行通信.然而,这种设计方案也带来了 TPaxos 与 Paxos 之间的诸多差异,给理解 TPaxos 造成了障碍.其次,虽然腾讯开源了 TPaxos 协议的核心代码(包括伪代码与 C++代码),TPaxos 仍缺少抽象而精确的形式化规约.最后,就我们所知,TPaxos 的正确性尚未经过必要的数学论证或者形式化工具的检验.针对这些情况,本文有三个主要贡献.首先,我们从经典的 Paxos 协议出发,论证如何逐步推导出 TPaxos 协议.基于这种推导,我们可以将 TPaxos 看作 Paxos 的一种自然变体,更易于理解.其次,我们给出了 TPaxos 协议的 TLA+形式化规约.在开发规约的时候,我们发现 TPaxos 协议描述中存在至关重要但并未充分阐明的微妙之处:在消息处理阶段,参与者(作为接受者角色)是先作出"不再接受具有更小编号的提议"的承诺(Promise)还是先接受(Accept)提议?这导致对 TPaxos 的两种不同理解,并促使我们提出 TPaxos 的一种变体,称为 TPaxosAP.在 TPaxosAP 中,参与者先接受提议后作承诺.最后,我们使用精化(refinement)技术论证了 TPaxos 与 TPaxosAP 的正确性.特别地,由于已知的投票机制 Voting 不能完全描述 TPaxosAP 的行为,我们首先提出了适用于 TPaxosAP 的投票机制 EagerVoting,然后建立了从 TPaxosAP 到 EagerVoting 以及从 EagerVoting 到 Consensus 的精化关系,并使用 TLC 模型检验工具验证了它们的正确性.

**关键词**: Paxos;PaxosStore;共识算法;TLA+;精化关系;模型检验

**中图法分类号**: TP301.2

中文引用格式: 易星辰,魏恒峰,黄宇,乔磊,吕建.PaxosStore 中共识算法 TPaxos 的推导、规约与精化.软件学报,2020,31(8) http://www.jos.org.cn/1000-9825/5964.htm

英文引用格式: Yi XC, Wei HF, Huang Y, Qiao L, Lu J. TPaxos in PaxosStore: derivation, specification, and refinement. Ruan Jian Xue Bao/Journal of Software, 2020,31(4) (in Chinese). http://www.jos.org.cn/1000-9825/5964.htm

## TPaxos in PaxosStore: Derivation, Specification, and Refinement

YI Xing-Chen[1], WEI Heng-Feng[1], HUANG Yu[1], QIAO Lei[2], LU Jian[1]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[2](Beijing Institute of Control Engineering, Beijing 100190, China)

**Abstract**: PaxosStore is a highly available distributed storage system developed by Tencent Inc. to support the comprehensive business of WeChat. PaxosStore employs a variant of Paxos which is a classic protocol for solving distributed consensus. We call it TPaxos in this paper. The originality of TPaxos lies in its "uniformity": it maintains for each participant a unified state type and adopts a universal message format for communication. However, this design choice brings various differences between TPaxos and Paxos, rendering TPaxos hard to understand. Moreover, although the core code (including both pseudocode and source code in C++) for TPaxos is publicly

* 基金项目: 国家自然科学基金(61690204,61702253,61772258)
Foundation item: National Natural Science Foundation of China (61690204,61702253,61772258)
收稿时间: 2019-09-09; 修改时间: 2019-11-02; 采用时间: 2019-12-30; jos 在线出版时间: 2020-04-18

available, there still lacks a formal specification of TPaxos. Finally, as far as we know, TPaxos has not yet been manually proven or formally checked. To address these issues, we make three main contributions in this paper. First, we demonstrate how to derive TPaxos from classic Paxos step by step. Based on this derivation, TPaxos can be regarded as a natural variant of Paxos and is much easier to understand. Second, we describe TPaxos in TLA+, a formal specification language. In the course of developing the TLA+ specification for TPaxos, we uncover a crucial but subtle detail in TPaxos which is not fully clarified: Upon messages, do the participants (as Acceptors) make promise that no proposals with smaller proposal numbers will be accepted before accepting proposals or vice versa? This leads to two different interpretations of TPaxos and motivates us to propose a variant of TPaxos, called TPaxosAP. In TPaxosAP, the participants accept proposals first and then make promise. Third, we verify the correctness of both TPaxos and TPaxosAP by refinement. Particularly, since the known voting mechanism called Voting cannot capture all the behaviors of TPaxosAP, we first propose EagerVoting for TPaxosAP and then establish the refinement mappings from TPaxosAP to EagerVoting and from EagerVoting to Consensus. We also verify them using the TLC model checker.

**Key words**: Paxos; PaxosStore; consensus protocols; TLA+; refinement mapping; model checking

分布式共识(Consensus)问题是分布式计算领域中的核心问题.它要求多个参与者就某个值达成一致[1] [2] .Paxos(也称 Basic Paxos 或 Single-Decree Paxos)是解决分布式共识问题的经典协议[3] [4] .通过并行执行多个独立的 Paxos 实例,Multi-Paxos 允许参与者就一系列值(也称序列)达成一致[3] [4] .Paxos/Multi-Paxos 及其变体被广泛用于需要提供强一致性的分布式系统中,如 Google 的 Chubby 分布式锁服务系统[1] [5] 与 Spanner 分布式数据库系统[6] 、微软的 Autopilot 自动数据中心管理系统[7] 以及腾讯的 PaxosStore 分布式存储系统[8] .本文关注 PaxosStore 系统所实现的 Paxos 变体.我们称之为 TPaxos.

在 TPaxos 中,每个参与者(即,数据副本节点)都同时具有提议者(Proposer)、接受者(Acceptor)与学习者(Learner)三种角色.TPaxos 的新颖之处在于它的"统一性":它为每个参与者维护统一的状态类型并采用统一格式的消息进行通信.在 TPaxos 中,参与者通过互相交换统一类型的消息,不断更新状态,最终达成共识.相比而言,经典的 Paxos 协议(及其众多变体)为不同的角色维护不同的状态类型,并且需要使用四种消息类型,以区分准备阶段(Prepare Phase;包含 Phase1a 与 Phase1b 两个子阶段)与接受阶段(Accept Phase;包含 Phase2a 与 Phase2b 两个子阶段)中的两轮双向通信.这种设计方案可以带来精简而高效的系统实现[8] .然而,这也带来了 TPaxos 与 Paxos 之间的诸多差异,给理解 TPaxos 造成了障碍.其次,虽然腾讯开源了 TPaxos 协议的核心代码(包括论文中的伪代码与实际系统中的 C++代码[9] ),TPaxos 仍缺少抽象而精确的形式化规约.最后,据我们所知,TPaxos 的正确性尚未经过必要的数学论证或者形式化工具的检验.

针对上述情况,本文有三个主要贡献:

- (第 2 节)第一,我们论证如何从 Paxos 出发,逐步推导出 TPaxos.首先,我们通过合并三种角色,统一参与者的状态类型.其次,基于统一的状态类型,我们可以统一消息内容.然后,通过合并某些子阶段,并避免使用消息状态信息区分协议阶段,我们可以得到一种比 TPaxos 更为精简的统一消息类型.最后,我们指出 TPaxos 在消息处理阶段所作的优化,从而推导出最终的 TPaxos 版本.
- (第 3 节)第二,我们给出了 TPaxos 的 TLA+[10] [11] [12] 形式化规约.与已有的 Paxos 的 TLA+规约[13] 类似,TPaxos 采用"已发送消息集合"作为通信信道以及消息发送与接收动作的抽象.这种抽象使得该规约不依赖于代码层面的具体通信机制.更重要的是,在开发 TLA+规约的时候,我们发现 TPaxos 协议描述中存在至关重要但并未充分阐明的微妙之处:在消息处理阶段,参与者(作为接受者角色)是先作出"不再接受具有更小编号的提议"的承诺(Promise)还是先接受(Accept)提议?这导致对 TPaxos 的两种不同理解,并促使我们提出 TPaxos 的一种变体,称为 TPaxosAP.在 TPaxosAP 中,参与者先接受提议后作承诺.虽然在 TLA+规约层面,TPaxosAP 与 TPaxos(先作承诺后接受提议)相差甚小,但它却体现了不同的投票机制.具体来讲,Paxos 协议的投票机制 Voting[13] 仍适用于 TPaxos,却不能完整刻画 TPaxosAP 的行为.在本节,我们将举例说明 TPaxosAP 与 TPaxos 的不同之处,并论证 TPaxosAP 的正确性.
- (第 4 节)第三,我们采用精化(refinement)[14] [15] 技术论证了 TPaxos 与 TPaxosAP 的正确性.对于

TPaxos,我们仍采用 Paxos 所使用的 Voting[13] 机制,建立了从 TPaxos 到 Voting 的精化关系.对于 TPaxosAP,我们首先提出了一种新的投票机制,称作 EagerVoting.EagerVoting 允许参与者在接受提议的同时,作出比 Paxos/TPaxos 更"激进"的"不再接受具有更小编号的提议"的承诺.然后,我们建立了从 TPaxosAP 到 EagerVoting 以及从 EagerVoting 到 Consensus 的精化关系,并使用 TLC 模型检验工具[16]验证它们的正确性.

本文组织如下:第 1 节介绍预备知识.第 2 节介绍 TPaxos 协议,并论证如何逐步从 Paxos 推导出 TPaxos.第 3 节先给出 TPaxos 的 TLA+规约,然后提出它的一种变体 TPaxosAP 并论证其正确性.第 4 节构建从 TPaxos 与 TPaxosAP 到 Consensus 的精化关系.第 5 节使用 TLC 模型检验工具验证精化关系并展示实验结果.第 6 节讨论相关工作.第 7 节总结全文,并讨论可能的未来工作.

## 1 预备知识

我们先介绍 TLA+形式化规约语言然后介绍分布式共识问题与经典 Paxos 协议(包括协议描述、TLA+规约与精化).

### 1.1 TLA+简介

TLA+是 Leslie Lamport 基于时序逻辑 TLA(Temporal Logic of Actions)[12] 开发的一种形式化规约语言,适用于描述并发及分布式系统.TLA+将系统建模成状态机.一个状态机由它可能的初始状态(initial states)与一组动作(actions)来刻画.在 TLA+中,一个状态是系统中所有变量的实例化.一个行为是状态的序列.一个动作是新旧状态之间的转移,刻画了系统执行指令前后的变化.例如,对于指令 $x := x+1$,TLA+采用动作 $x' = x+1$ 进行描述.TLA+使用一个包含带撇变量(表示新状态的值)与不带撇变量(表示旧状态的值)的公式来描述动作.

一般而言,一个系统可以表达为 TLA 中的一个形如 $Spec \triangleq Init \wedge \left[Next\right]_{vars} \wedge L$ 的时序公式.其中,*Init* 和 *Next* 都是状态谓词(结果为布尔值),对系统状态进行刻画.*Init* 刻画了系统的初始状态,*Next* 是所有动作的析取式,描述了系统的所有动作.*vars* 是包含系统所有变量的元组,$\Box$ 是"总是(always)"的时序操作符.表达式 $\left[Next\right]_{vars}$ 为真当且仅当 *Next* 为真(即某个动作为真,意味着该动作对应的指令被系统执行),或者 *vars* 在新旧状态中保持不变,后者是 TLA+的一个特性,有利于构建系统间的精化关系.$L$ 用于检测系统的活性(Liveness).

在 TLA+中,$L$ 通常是 *WF* 的析取式,*WF* 指弱公平性(weak fairness),$WF_{vars}(A)$刻画了动作 $A$ 的公平性,即要求如果动作 $A$ 从某个时刻开始是持续可执行的,则 $A$ 最终将被执行.$WF_{vars}(A)$的定义如下:

$$WF_{var}(A) \triangleq \Box\left(\Box ENABLED\langle A\rangle_{vars} \Rightarrow \Diamond\langle A\rangle_{vars}\right)$$

其中,$\langle A\rangle_{vars}$ 表示动作 $A$ 被执行,$ENABLED\langle A\rangle_{vars}$ 表示动作 $A$(在当前状态下)是可执行的,$\Diamond$ 是"最终(eventually)"的时序操作符.

TLA+在 TLA 的基础上,加入了一阶谓词逻辑以及 ZF 集合论,从而支持丰富的数据类型与表达式.图 1 总结了本文使用到的逻辑与集合操作符(operator).文献[17] 给出了完整的 TLA+操作符列表.

TLA+规约以模块(module)的形式组织在一起.在每个模块中,我们可以声明常量(CONSTANTS)与变量(VARIABLES)或者提出定理(THEOREM).一个模块 $M$ 可以通过扩展其它模块 $M_1,\ldots,M_n$ 的方式引入声明、定义与定理;在模块 $M$ 中,写作 EXTENDS $M_1,\ldots,M_n$ .模块也可以被实例化.考虑 $M$ 模块中的实例化语句:

$$IM_1 \triangleq \text{INSTANCE } M_1 \text{ WITH } p_1 \leftarrow e_1,\ldots,p_n \leftarrow e_n$$

其中,$p_i$包含了$M_1$中的所有常量与变量,$e_i$是 $M$ 中的合法表达式.该语句是通过将$M_1$中的$p_i$替换为 $M$ 中相应的$e_i$来进行$M_1$的实例化,我们可以使用 $IM_1!F$ 来访问模块$M_1$中的表达式 $F$.此外,TLA+中的隐式替换规则允许我们在 $e_j$ 与 $p_j$ 相同时省略 $p_j \leftarrow e_j$ 子句.

TLC 是 TLA+的模型检验工具,它通过遍历 TLA+规约的有穷状态空间来验证规约是否满足具体的性质.然而有些分布式系统的状态并非有穷,例如,考虑一个包含多个服务器的分布式系统,服务器(定义为 *Server*)的数量可以任意大,甚至无穷.为了能够验证这一类系统规约,TLC 通过模型(model)的方式限制了这些系统的状态

数,即 TLC 可以在模型中实例化服务器为有穷集合,如 $Server \triangleq \{s_1, s_2, s_3\}$.如果对于值 $\{s_1, s_2, s_3\}$ 进行任意重排(如将 $s_1$ 替换为 $s_2$, $s_2$ 替换为 $s_3$, $s_3$ 替换为 $s_1$)不影响 TLC 的验证结果,我们可以将 *Server* 标记为对称集(Symmetry Set)[18] 以减少 TLC 需要遍历的状态空间.

在 TLA+中,精化(refinement)关系就是逻辑蕴含(logical implication)关系.例如,考虑模块 *AbsModule* 中的

| | 操作符 | 含义 |
|---|---|---|
| 逻辑 | CHOOSE $x \in S : p$ | 选择集合 $S$ 中满足条件 $p$ 的元素 $x$<br>(通常用于符合条件的 $x$ 是唯一的情况下) |
| 集合 | SUBSET $S$ | $S$ 的幂集 |
| | $\{e : x \in S\}$ | 将 $e$ 作用在 $S$ 中所有元素得到的集合<br>比如 $\{x^2 : x \in S\}$ |
| | $\{x \in S : p\}$ | $S$ 中满足条件 $p$ 的元素构成的集合 |
| 函数 | $f[e]$ | 函数 $f$ 作用在参数 $e$ 上 |
| | $[x \in S \mapsto e]$ | 对于 $x \in S$, 使得 $f[x] = e$ 的函数 $f$ |
| | $[f \text{ EXCEPT } ![e_1] = e_2]$ | 函数 $\hat{f} : \hat{f}[e] = \begin{cases} e_2, & \text{if } e = e_1 \\ f[e], & \text{otherwise} \end{cases}$ |
| | $[f \text{ EXCEPT } ![c] = e]$,<br>其中 $e$ 包含符号 @ | $e$ 中的 @ 表示 $f[c]$ |
| 记录 | $e.h$ | 记录 $e$ 的域 $h$ |
| | $[h_1 \mapsto e_1, \ldots, h_n \mapsto e_n]$ | 域 $h_i$ 为 $e_i$ 的记录 |
| | $[h_1 : S_1, \ldots, h_n : S_n]$ | 满足域 $h_i$ 属于 $S_i$ 的所有记录构成的集合 |
| | $[r \text{ EXCEPT } !.h = e]$ | 记录 $\hat{r} : [h_1 : S_1, \ldots, h : e, \ldots, h_n : S_n]$ |
| | $[r \text{ EXCEPT } !.h = e]$,<br>其中 $e$ 包含符号 @ | $e$ 中的 @ 表示 $r.h$ |
| 元组 | $e[i]$ | 元组 $e$ 中的第 $i$ 个分量 |
| 动作操作符 | $e'$ | 动作的新状态中 $e$ 的值 |
| | UNCHANGED $e$ | $e$ 不变:$e' = e$ |
| | $[A]_e$ | 动作 $A$ 成立或者 $e$ 不变:$A \vee (e' = e)$ |
| 时序操作符 | $\Box F$ | $F$ 在所有情况下均成立 ($\Box$ 表示"always") |
| | $\Diamond F$ | $F$ 最终成立 ($\Diamond$ 表示"eventually") |
| | $WF_e(A)$ | 动作 $A$ 的弱公平性 (weak fairness) |

Figure 1: A summary of TLA+ operators used in this paper

图 1: 本文所使用的 TLA+操作符

*AbsSpec* 规约以及模块 *ImplModule* 中的 *ImplSpec* 规约. *AbsSpec* 包含变量 $x_1, \cdots, x_m, y_1, \cdots, y_n$, *ImplSpec* 包括变量 $x_1, \cdots, x_m, z_1, \cdots, z_p$.令 *X*、*Y* 与 *Z* 分别代表变量组 $x_1, \cdots, x_m$、$y_1, \cdots, y_n$ 与 $z_1, \cdots, z_p$.为了验证 *ImplSpec* 实现/精化了 *AbsSpec*(也称 *ImplSpec* 是 *AbsSpec* 的精化;即 $ImplSpec \Rightarrow AbsSpec$),我们需要证明,对于每一个满足 *ImplSpec* 的行为,都存在一种关于变量组 *Y* 的赋值方式,使得赋值后所得的行为(考虑变量组 *X* 与 *Y*)满足 *AbsSpec*[14] [15] .具体而言,对于每个 $y_i$,我们仅使用 *X* 与 *Z* 定义表达式 $\overline{y_i}$,在 *AbsSpec* 中做替换 $y_i \leftarrow \overline{y_i}$,得到新的规约 $\overline{AbsSpec}$,然后证明 *ImplSpec* 实现/蕴含了 $\overline{AbsSpec}$.这里,替换 $y_i \leftarrow \overline{y_i}$ 被称为精化映射(refinement mapping).为了使用 TLC 检验在精化映射 $y_i \leftarrow \overline{y_i}$ 下,*ImplSpec* 实现了 *AbsSpec*,我们在模块 *ImplModule* 中添加定义 $AbsSub \triangleq INSTANCE\ AbsModule\ WITH\ y_1 \leftarrow \overline{y_1}, \cdots, y_n \leftarrow \overline{y_n}$,并检验定理 $THEOREM\ ImplSpec \Rightarrow AbsSub!AbsSpec$.

### 1.2 分布式共识

我们使用"三种角色模型"定义分布式共识问题[3] [19] .提议者(Proposers)可以提出值(values),接受者(Acceptors)负责选择值,学习者(Learners)需要获悉被选中(chosen)的值.分布式共识问题的安全性(safety)需求包括:

- 非平凡性(Nontriviality).只有被提出过的值才能被选中.
- 一致性(Consistency).最多只有一个值被选中.

TLA+模块 *Consensus*[13] 抽象地描述了分布式共识问题.常量 *Value* 表示所有可能被提出的值构成的集合,

它建模了提议者角色.变量 *chosen* 表示已被选中的值构成的集合,它建模了接受者角色.在初始状态下,*chosen* 为空.该规约只有一个可能的动作:如果 *chosen* 为空,则从 *Value* 中任选一个值 *v*,加入 *chosen* 中(即,选中 *v*).不变式 *Inv* 表达了分布式共识问题的安全性,其中第三条合取式是一致性的定义,描述了 *chosen* 中的元素个数不超过 1,即最多只有一个值被选中.显然,规约 *Spec* 满足 *Inv*,即有 *THEOREM Spec* $\Rightarrow$ *Inv* 成立.

MODULE *Consensus*

EXTENDS *Naturals*, *FiniteSets*

CONSTANT *Value* 被提出值的集合

VARIABLE *chosen* 被选中值的集合

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \land chosen = \{\} \land \exists v \in Value : chosen' = \{v\}$$

$$Spec \triangleq Init \land \Box[Next]_{chosen}$$

$$Inv \triangleq \land chosen \subseteq Value \land IsFiniteSet(chosen) \land Cardinality(chosen) \leq 1$$

$$Liveness \triangleq \Diamond(chosen \neq \{\})$$

除了安全性,分布式共识问题还包括活性,*Liveness* 表达了活性的具体定义,即最终会有某个值被选中.

### 1.3 Paxos协议

Paxos 是在异步消息传递系统中解决分布式共识问题的重要协议.它采用传统的非拜占庭(non-Byzantine)故障模型[3] .在该模型中:

- 每个参与者(包括提议者、接受者与学习者)以各自的速度异步执行.参与者可能终止执行(fail by stopping),也可能重启(restart).
- 消息可能会延时到达、重复或丢失.但是,消息不会被篡改.

在 Paxos 中,提议者为每个提议值 *v* 附加一个(全局唯一的)提议编号 *b*.我们用 $\langle b,v\rangle$ 表示提议.接受者按规则接受提议.如果某个提议被"足够多"的接受者接受了,则称该提议以及它包含的值被选中.这里的"足够多"通常是指接受者中的任意一个多数派(majority).因此,Paxos 可以容忍不超过半数的接受者失效. "多数派"是一种特殊的议会系统(quorum system).设集合 $\mathcal{Q} \subseteq 2^A$ 是由接受者集合 $A$ 的子集构成的集合.如果 $\mathcal{Q}$ 满足以下两个条件,则称 $\mathcal{Q}$ 为一个议会系统:

- $\mathcal{Q}$ 构成 $A$ 的集合覆盖(set cover): $\bigcup_{Q\in\mathcal{Q}} Q = A$ .
- 任意两个议会(quorum)相交不为空: $\forall Q_1, Q_2 \in \mathcal{Q} : Q_1 \cap Q_2 \neq \varnothing$ .

Paxos 包含两个阶段,每个阶段由两个子阶段构成[3] :

- 准备(Prepare)阶段(也称第一阶段 Phase1)
  - 子阶段 Phase1a:提议者选取一个提议编号 *b*,然后向所有接受者[1]发送编号为 *b* 的 Prepare 请求.
  - 子阶段 Phase1b:如果一个接受者收到编号为 *b* 的 Prepare 请求,并且编号 *b* 大于它之前回复过的所有 Prepare 请求的编号,那么它就将自己接受过的编号最大的提议(包括编号与值)回复给相应的提议者,并承诺不会再接受任何编号小于 *b* 的提议.
- 接受(Accept)阶段(也称第二阶段 Phase2)

1 在原始 Paxos 协议描述中,只要求向接受者中的某个议会发送请求.为了叙述方便,本文改为向所有接受者发送请求.显然,该改动不影响协议的正确性.

- 子阶段 Phase2a:如果提议者收到来自接受者中的某个议会对编号为 $b$ 的 Prepare 请求的回复,那么它就向所有接受者发送一个针对提议 $\langle b,v\rangle$ 的 Accept 请求.其中,$v$ 由收到的回复决定:如果回复中不包含任何提议,则 $v$ 可以是任意值;否则,$v$ 是这些回复中编号最大的提议对应的值.
- 子阶段 Phase2b:如果一个接受者收到针对提议 $\langle b,v\rangle$ 的 Accept 请求,只要它尚未对编号大于 $b$ 的 Prepare 请求作过回复,它就可以接受该提议.

### 1.4 Paxos协议的TLA+规约

#### 1.4.1 常量

Paxos 的 TLA+规约[13] 包含三个常量:

- *Value*:所有可能的提议值构成的集合(如, $\{v_1, v_2, v_3\}$).我们定义 $None \triangleq CHOOSE\ v : v \notin Value$ 表示不属于 *Value* 的某个值.
- *Acceptor*:所有接受者构成的集合(如,$\{a_1, a_2, a_3\}$).
- *Quorum*:由接受者形成的议会系统.*QuorumAssumption* 给出了议会系统需要满足的条件.

我们用自然数表示可能的提议编号,即 $Ballot \triangleq Nat$ ,并用 –1 表示相关变量初始化时用到的最小提议编号.

```
MODULE Paxos
EXTENDS Integers

CONSTANT Value, Acceptor, Quorum
ASSUME QuorumAssumption ≜
          ∧ ∀ Q ∈ Quorum : Q ⊆ Acceptor
          ∧ ∀ Q1, Q2 ∈ Quorum : Q1 ∩ Q2 ≠ {}
Ballot ≜ Nat
None ≜ CHOOSE v : v ∉ Value   一个不属于集合 Value 的特殊值
Paxos 通过消息传递进行通信, 我们定义了所有消息的类型, 以下是消息定义.
Message ≜ [type : {"1a"}, bal : Ballot]
          ∪ [type : {"1b"}, acc : Acceptor, bal : Ballot,
             mbal : Ballot ∪ {−1}, mval : Value ∪ {None}]
          ∪ [type : {"2a"}, bal : Ballot, val : Value]
          ∪ [type : {"2b"}, acc : Acceptor, bal : Ballot, val : Value]

VARIABLE maxBal,
         maxVBal,   二元组 (maxVBal[a], maxVVal[a]) 是 a 接受过的
         maxVVal,   编号最大的提议, 如果 a 没有接受过任何提议, 则为 (-1, None)
         msgs       已发送消息的集合
vars ≜ ⟨maxBal, maxVBal, maxVVal, msgs⟩
TypeOK ≜ ∧ maxBal ∈ [Acceptor → Ballot ∪ {−1}]
         ∧ maxVBal ∈ [Acceptor → Ballot ∪ {−1}]
         ∧ maxVVal ∈ [Acceptor → Value ∪ {None}]
         ∧ msgs ⊆ Message
Init ≜ ∧ maxBal = [a ∈ Acceptor ↦ −1]
       ∧ maxVBal = [a ∈ Acceptor ↦ −1]
       ∧ maxVVal = [a ∈ Acceptor ↦ None]
       ∧ msgs = {}
Send(m) ≜ msgs′ = msgs ∪ {m}
```

#### 1.4.2 变量

Paxos 的 TLA+规约包含四个变量:

- *maxBal*:*maxBal[a]*表示接受者 $a$ 在 Phase1b 与 Phase2b 两个子阶段见到过的最大提议编号,也是 $a$ 承诺可以接受的提议的最小编号.

- *maxVBal* 与 *maxVVal*:*maxVBal[a]*表示接受者 *a* 接受过提议的最大编号,*maxVVal[a]*[1]是对应提议中的提议值.我们使用符号⟨ *maxVBal[a]*,*maxVVal[a]* ⟩表示该提议.
- *msgs*:所有已发送消息构成的集合.参与者通过向该集合添加消息或者从该集合读取消息模拟消息的发送(广播)与接收动作.*Send(m)*将消息 *m* 加入到 *msgs* 中.为了对应 Paxos 协议的四个子阶段,该规约定义了四种消息类型,分别为"*1a*"、"*1b*"、"*2a*"与"*2b*".

*TypeOK* 定义了各个变量的类型.*Init* 给出了初始状态下各个变量的取值.

### 1.4.3 动作

Paxos 规约定义了四种动作,分别对应于协议的四个子阶段:

动作 $Phase1a(b)$ 发送类型为"1a" 的消息, 携带的提议编号为 $b$.

$$
\begin{aligned}
Phase1a(b) \triangleq\ & \land Send([type \mapsto \text{“1a”},\ bal \mapsto b]) \\
& \land \text{UNCHANGED } \langle maxBal,\ maxVBal,\ maxVVal\rangle
\end{aligned}
$$

一旦收到携带编号为 $b$ 的"1a" 消息, 接受者能执行动作 $Phase1b(a)$ 当且仅当 $b>maxBal[a]$. 该动作更新 $maxBal[a]$ 至 $b$ 并且将提议 $\langle maxVBal[a], maxVVal[a]\rangle$ 回复给提议者.

$$
\begin{aligned}
Phase1b(a) \triangleq\ & \land \exists\, m \in msgs : \\
& \qquad \land m.type = \text{“1a”} \\
& \qquad \land m.bal > maxBal[a] \\
& \qquad \land maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
& \qquad \land Send([type \mapsto \text{“1b”},\ acc \mapsto a,\ bal \mapsto m.bal, \\
& \qquad\qquad mbal \mapsto maxVBal[a],\ mval \mapsto maxVVal[a]]) \\
& \land \text{UNCHANGED } \langle maxVBal,\ maxVVal\rangle
\end{aligned}
$$

动作 $Phase2a(b, v)$ 能被执行需要两个前置条件:(i) 尚未执行动作 $Phase2a(b, v)$;(ii) 收到了来自一个议会针对提议编号 $b$ 的"1b" 消息, 这两个前置条件对应于前两条合取式. 第二条合取式是整个算法的核心, 其保证了 $SafeAt(b, v)$ 性质的成立.
该动作发送了类型为"2a" 的消息, 携带提议 $\langle b, v\rangle$, 请求接受者接受该提议.

$$
\begin{aligned}
Phase2a(b,\ v) \triangleq\ & \\
& \land \neg\exists\, m \in msgs \ : m.type = \text{“2a”} \land m.bal = b \\
& \land \exists\, Q \in Quorum : \\
& \qquad \text{LET } Q1b \triangleq \{m \in msgs : \land m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\qquad \land m.acc \in Q \\
& \qquad\qquad\qquad\qquad\qquad \land m.bal = b\} \\
& \qquad\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
& \qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \qquad\qquad \land \lor Q1bv = \{\} \\
& \qquad\qquad\qquad \lor \exists\, m \in Q1bv : \\
& \qquad\qquad\qquad\qquad \land m.mval = v \\
& \qquad\qquad\qquad\qquad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal \\
& \land Send([type \mapsto \text{“2a”},\ bal \mapsto b,\ val \mapsto v]) \\
& \land \text{UNCHANGED } \langle maxBal,\ maxVBal,\ maxVVal\rangle
\end{aligned}
$$

一旦接受者收到了"2a" 消息 $m$, 该接受者能执行该动作当且仅当 $m.bal \geq maxBal[a]$. 接受者将接受 $m$ 携带的提议 $\langle b, v\rangle$, 更新二元组 $(maxVBal[a], maxVVal[a])$ 至 $(b, v)$, 同时也会更新 $maxBal[a]$ 至 $b$.

$$
\begin{aligned}
Phase2b(a) \triangleq\ & \\
& \exists\, m \in msgs : \land m.type = \text{“2a”} \\
& \qquad\qquad \land m.bal \geq maxBal[a] \\
& \qquad\qquad \land maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
& \qquad\qquad \land maxVBal' = [maxVBal \text{ EXCEPT } ![a] = m.bal] \\
& \qquad\qquad \land maxVVal' = [maxVVal \text{ EXCEPT } ![a] = m.val] \\
& \qquad\qquad \land Send([type \mapsto \text{“2b”},\ acc \mapsto a, \\
& \qquad\qquad\qquad bal \mapsto m.bal,\ val \mapsto m.val])
\end{aligned}
$$

- *Phase1a(b)*:提议者选取提议编号 *b*,发起 Prepare 请求.消息类型为 *m.type* ↦ "1a",并携带提议编号 *m.bal* ↦ *b* .
- *Phase1b(a)*:接受者 *a* 收到类型为"1a"的消息 *m*.如果 *m* 中携带的提议编号 *m.bal* 大于 *maxBal*[*a*],则接

[1] Paxos 的原始 TLA+规约中使用了变量名 *maxVal*.为了与 *maxVBal* 对应,本文改用 *maxVVal*.

受者 *a* 将 *maxBal[a]*更新为 *m.bal*,并将它接受过的最大编号提议(即, ⟨ *maxVBal[a],maxVVal[a]* ⟩ )回复给提议者.消息类型为"1b",并携带收到的提议编号 *m.bal*.

- *Phase2a(b,v)*:提议者发起针对提议⟨*b*,*v*⟩ 的 Accept 请求.该动作有两个前置条件:(1)第一个合取子句要求,针对提议编号 *b*,提议者尚未发起过 Accept 请求,即 *msgs* 中不存在类型为"2a"且编号为 *b* 的消息.(2)第二个合取子句要求,存在接受者中的某个议会 *Q*,*Q* 中的每个接受者都回复过针对提议编号 *b* 的 Prepare 请求.前置条件成立时,提议者便根据 *Q* 中每个接受者回复的消息确定 *b* 值(方法如 Paxos 协议的 Phase2a 子阶段所述),然后通过类型为"2a"的消息发起针对提议⟨*b*,*v*⟩ 的 Accept 请求.
- *Phase2b(a)*: 接受者 *a* 接受某个提议.该动作的前置条件是存在类型为"2a"的消息 *m*,并且 *m* 中携带的提议编号 *m.bal* 大于等于 *maxBal[a]*.前置条件成立时,接受者 *a* 将 *maxBal[a]*更新为 *m.bal*[1],并且接受消息 *m* 中携带的提议⟨ *m.bal,m.val* ⟩,然后将该提议封装在类型为"2b"的消息中发送出去.

1.4.4 行为

*Next* 定义了次态关系.*Spec* 定义了完整的行为规约:

$$
\begin{array}{l}
Next \triangleq \lor \exists\, b \in Ballot : \lor Phase1a(b) \\
\qquad\qquad\qquad\qquad \lor \exists\, v \in Value : Phase2a(b,\, v) \\
\qquad\quad \lor \exists\, a \in Acceptor : Phase1b(a) \lor Phase2b(a) \\
Spec \triangleq Init \land \Box[Next]_{vars}
\end{array}
$$

### 1.5 Paxos的精化

Paxos 协议的核心在于接受者何时可以接受提议,以及这些提议需要满足何种条件.一方面,接受者 *a* 只有在提议编号 *b* 大于等于它回复过的最大提议编号 *maxBal[a]*(也是它承诺可以接受的提议的最小编号)时,它才可能接受该提议.另一方面,可以被接受的提议⟨*b*,*v*⟩ 需要满足两个关键条件:一是,每个提议编号(这里的 *b*)最多对应一个提议(这里的⟨*b*,*v*⟩ ).该条件称为 *OneValuePerBallot*.二是,⟨*b*,*v*⟩ 是安全的,即对于任何提议⟨*b*′,*v*′⟩ ,其中 $b' \in 0 \ldots b-1, v' \neq v$ , ⟨*b*′,*v*′⟩ 都尚未被选中且将来也不会被选中.该条件称为 *SafeAt(b, v)*.Lamport 使用了 TLAPS(TLA+ Proof System)证明了[20] Paxos 协议正确性关键在于上述两个条件,即只要满足这两个条件,就能够保证 Paxos 的正确性.

为了更好地理解上述核心问题,Lamport 提出了一个更抽象的规约,记为 *Voting*[21] .在 *Voting* 中,每个接受者 *a* 仍然维护 *maxBal[a]*.此外,它还维护它曾接受过的所有提议 *votes*.该规约包含两个动作:在 *IncreaseMaxBal(a,b)* 中,接受者 *a* 可以提高自己的 *maxBal[a]*.该动作对应于 *Phase1a* 与 *Phase1b* 子阶段.在 *VoteFor(a,b,v)*中,接受者 *a* 接受提议⟨*b*,*v*⟩ .其中,第二与第三个合取式保证了 *OneValuePerBallot* 条件.第四个合取式保证了 *SafeAt(b,v)*条件.该动作对应于 *Phase2a* 与 *Phase2b* 子阶段.

由于 *VoteFor(a,b,v)*使用了所有被接受过的提议这一全局信息,*Voting* 可以看作分布式共识问题的集中式解决方案,而 *Paxos* 是 *Voting* 的分布式实现.实际上,Lamport 构建了从 *Paxos* 到 *Voting* 以及从 *Voting* 到 *Consensus* 的精化关系[13] .

1 需要注意的是,自然语言描述的 Paxos 协议[3] 没有要求更新 *maxBal[a]*.然而,可以通过反例说明,如果不同时更新 *maxBal[a]*,则 Paxos 并不能保证论文中的不变式 P2[c] 成立.

```
MODULE Voting
EXTENDS Integers, FiniteSets, TLAPS

CONSTANT Value, Acceptor, Quorum
ASSUME QuorumAssumption ≜ ∧ ∀ Q ∈ Quorum : Q ⊆ Acceptor
                          ∧ ∀ Q1, Q2 ∈ Quorum : Q1 ∩ Q2 ≠ {}
Ballot ≜ Nat
VARIABLES votes, maxBal
TypeOK ≜ ∧ votes ∈ [Acceptor → SUBSET (Ballot × Value)]
         ∧ maxBal ∈ [Acceptor → Ballot ∪ {−1}]

Init ≜ ∧ votes = [a ∈ Acceptor ↦ {}]
       ∧ maxBal = [a ∈ Acceptor ↦ −1]
IncreaseMaxBal(a, b) ≜ ∧ b > maxBal[a]
                       ∧ maxBal' = [maxBal EXCEPT ![a] = b]   作承诺
                       ∧ UNCHANGED votes
VoteFor(a, b, v) ≜
    ∧ maxBal[a] ≤ b                                           遵守承诺
    ∧ ∀ vt ∈ votes[a] : vt[1] ≠ b
    ∧ ∀ c ∈ Acceptor \ {a} :
        ∀ vt ∈ votes[c] : (vt[1] = b) ⇒ (vt[2] = v)
    ∧ ∃ Q ∈ Quorum : ShowsSafeAt(Q, b, v)                     满足 SafeAt 属性
    ∧ votes' = [votes EXCEPT ![a] = votes[a] ∪ {⟨b, v⟩}]      投票
    ∧ maxBal' = [maxBal EXCEPT ![a] = b]                      作承诺

Next ≜ ∃ a ∈ Acceptor, b ∈ Ballot :
          ∨ IncreaseMaxBal(a, b)
          ∨ ∃ v ∈ Value : VoteFor(a, b, v)
Spec ≜ Init ∧ □[Next]⟨votes, maxBal⟩

VotedFor(a, b, v) ≜ ⟨b, v⟩ ∈ votes[a]
DidNotVoteAt(a, b) ≜ ∀ v ∈ Value : ¬VotedFor(a, b, v)
ShowsSafeAt(Q, b, v) ≜
    ∧ ∀ a ∈ Q : maxBal[a] ≥ b                                 作出过承诺
    ∧ ∃ c ∈ −1 .. (b − 1) :
        ∧ (c ≠ −1) ⇒ ∃ a ∈ Q : VotedFor(a, c, v)
        ∧ ∀ d ∈ (c + 1) .. (b − 1), a ∈ Q : DidNotVoteAt(a, d)
CannotVoteAt(a, b) ≜ ∧ maxBal[a] > b
                     ∧ DidNotVoteAt(a, b)
NoneOtherChoosableAt(b, v) ≜
    ∃ Q ∈ Quorum :
        ∀ a ∈ Q : VotedFor(a, b, v) ∨ CannotVoteAt(a, b)
Paxos 的两个关键条件,SafeAt 和 OneValuePerBallot, 满足了这两个条件即可保证 Consistency 属性
SafeAt(b, v) ≜ ∀ c ∈ 0 .. (b − 1) : NoneOtherChoosableAt(c, v)
OneValuePerBallot ≜
    ∀ a1, a2 ∈ Acceptor, b ∈ Ballot, v1, v2 ∈ Value :
        VotedFor(a1, b, v1) ∧ VotedFor(a2, b, v2) ⇒ (v1 = v2)
```

## 2 TPaxos 协议及其推导

本节首先介绍 TPaxos 协议,然后论证如何从经典 Paxos 协议推导出 TPaxos 协议.

### 2.1 TPaxos协议

在 TPaxos 协议中,所有的参与者(即,数据副本节点;假设有 $N$ 个)都同时具有提议者、接受者与学习者三种角色.TPaxos 协议的新颖之处在于它的"统一性":它为每个参与者维护统一的状态类型,并采用统一格式的消息进行通信.相比而言, Paxos 及其众多变体为提议者与接受者维护不同的状态类型(即,维护不同的变量),并且需

要四种消息类型来区分协议的四个子阶段.下面,我们先分别介绍统一的状态类型与消息类型,然后介绍具体协议过程.

在 TPaxos 中,每个提议 $P$ 是一个二元组 $\langle b,v\rangle$,其中 $b$ 表示提议编号,$v$ 表示提议值.每个参与者 $p$ 维护一个大小为 $N$ 的状态向量 $S^p$.其中,参与者 $q$ 对应的向量值,记为 $S_q^p$,表示 $p$ 对 $q$ 的观察状态(*view state*).特殊地,如果 $p=q$,则称 $S_p^p$ 为 $p$ 的当前实际状态(*actual state*).每个状态 $S_q^p$ 记为$(m,P)$:$m$ 表示 $q$ 承诺可以接受提议的最小编号;$P$ 表示 $q$ 最近一次接受的提议 $\langle b,v\rangle$.我们使用 $S_q^p.m$、$S_q^p.P$、$S_q^p.P.b$ 与 $S_q^p.P.v$ 访问相应的状态分量.

在 TPaxos 中,参与者 $p$ 发送给 $q$ 的统一消息类型可记为 $M_{p\to q}=\left(S_p^p,S_q^p\right)$.也就是说,$p$ 会将自身的实际状态 $S_p^p$ 以及它对 $q$ 的观察状态 $S_q^p$ 一起发送给 $q$.

参与者之间通过相互交换此类消息不断更新状态,并在本地判断是否有值已被选中,从而最终达成共识.具体而言,TPaxos 的执行过程与 Paxos 类似,也分为两个阶段(伪代码见图 2):

- 准备阶段
  - Phase1a 子阶段:参与者 $p$ 选取一个大于 $S_p^p.m$ 的提议编号 $b$,更新 $S_p^p.m$ 至 $b$,然后向其它所有参与者 $q\neq p$ 发送消息 $M_{p\to q}=\left(S_p^p,S_q^p\right)$(即,Prepare 请求).
  - 消息处理(OnMessage)子阶段:参与者 $q$ 收到并处理来自 $p$ 的消息 $M_{p\to q}=\left(S_p^p,S_q^p\right)$.
    - 首先,$q$ 根据 $S_p^p$ 更新本地状态 $S_p^q$ 与 $S_q^q$.此处重点关注 $S_q^q$ 的更新:
      1) 作承诺:如果 $S_q^q.m<S_p^p.m$,则将 $S_q^q.m$ 更新为 $S_p^p.m$.这模拟了 Paxos 中的 Phase1b 阶段.
      2) 接受提议:如果 $S_q^q.m<S_p^p.P.b$,则将 $S_q^q.P$ 更新为 $S_p^p.P$.这模拟了 Paxos 中的 Phase2b 阶段.

      消息处理阶段(的核心)是 Phase1b 与 Phase2b 子阶段的合并.需要注意的是,根据 TPaxos 的伪代码,$q$ 先作承诺后接受提议[1].第 3 节将讨论另一种方案,即 $q$ 先接受提议后作承诺.
    - 然后,$q$ 根据本地状态向量 $S^q$ 判断是否有值已被选中;见 *IsValueChosen(q)*.
    - 最后,如果 $S_q^p$ 比更新后的 $S_q^q$ 状态要旧,那么 $q$ 向 $p$ 发送消息 $M_{q\to p}=\left(S_q^q,S_p^q\right)$.
- 接受阶段
  - Phase2a 子阶段:参与者 $p$ 根据本地状态向量 $S^p$ 判断是否可以发起针对提议 $P=\langle b,v\rangle$ 的 Accept 请求.它要求存在某个议会 $Q$,使得对于 $Q$ 中的每个参与者 $q$,都满足 $S_q^p.m=b$.这表明,$p$ 观察到 $Q$ 中每个参与者都回复了编号为 $b$ 的 Prepare 请求.在这种情况下,$p$ 将根据 $S^p$ 确定提议值 $v$:如果对于所有的参与者 $q$,都满足 $S_q^p.P.b=-1$(即,$p$ 未观察到某参与者接受过任何提议),则 $v$ 可以为任意值;否则,$v$ 是所有 $S_q^p.P$ 中最大提议编号对应的提议值.然后,$p$ 更新本地状态 $S_p^p.P$ 为 $\langle b,v\rangle$,并向其它所有参与者 $q\neq p$ 发送消息 $M_{p\to q}=\left(S_p^p,S_q^p\right)$(即,Accept 请求).

    在该阶段,有两点需要特别注意:第一,在 TPaxos 中,$p$ 根据它对所有参与者的观察状态确定 $v$,而不仅限于议会 $Q$ 中的参与者.第 4 节将讨论如何处理这种差别.第二,$p$ 在确定 $v$ 之后,立即将本地状态 $S_p^p.P$ 设置为 $\langle b,v\rangle$.第 3 节将论述它可能带来的问题.
  - 消息处理子阶段:与准备阶段中的消息处理子阶段相同.

### 2.2 从Paxos推导TPaxos

如前所述,TPaxos 采用统一的状态与统一的消息类型.这有助于精简而高效的工程实现.但是,这个特点也带来了 TPaxos 与 Paxos 之间的诸多差异,给理解 TPaxos 造成了障碍.本节分四个步骤论证如何从 Paxos 推导出 TPaxos,使得 TPaxos 可看作 Paxos 的一种自然变体,更易于理解.

---

[1] 此时,第 2)步.$S_q^q.m<S_p^p.P.b$.中的 $S_q^q.m$ 可能已被更新.

**Algorithm 1:** TPaxos in PaxosStore

```
1  Procedure Prepare(b)
2      if S_p^p.m < b then
3          S_p^p.m ← b
4          foreach remote replica node q do
5              send M_{p→q}
6  Procedure Accept(P_i)
7      if |{∀S_q^p ∈ S^p | S_q^p.m = P_i.b}|×2 > |S^p| then
8          if |{∀S_q^p ∈ S^p | S_q^p.P.v ≠ null}| > 0 then
9              P' ← the proposal with maximum P.b in S^p
10             S_p^p.P ← (P_i.b, P'.v)
11         else
12             S_p^p.P ← P_i
13         foreach remote replica node q do
14             send M_{p→q}
15 Procedure OnMessage(M_{p→q})
16     UpdateStates(q, S_p^p)
17     if S_q^p.m < S_q^q.m or S_q^p.P.b < S_q^q.P.b then
18         send M_{q→p}
19     if S_q^q is changed then
20         if IsValueChosen(q) is true then commit
21 Function UpdateStates(q, S_p^p)
22     if S_q^q.m < S_p^p.m then S_q^q.m ← S_p^p.m
23     if S_q^q.P.b < S_p^p.P.b then S_q^q.P ← S_p^p.P
24
25     if S_q^q.m < S_p^p.m then S_q^q.m ← S_p^p.m
26     if S_q^q.m ≤ S_p^p.P.b then S_q^q.P ← S_p^p.P
27 Function IsValueChosen(q)
28     n' ←occurrebce coubt of the most frequebt P.b in S^q
29     return n' × 2 > |S^q|
```

Figure 2: Pseudo-code of TPaxos in PaxosStore

图 2: TPaxos 伪代码

I. **统一状态类型.**TPaxos 之所以能够采用统一的状态类型,是因为在 TPaxos 中,每个参与者都同时具有提议者、接受者与学习者的角色.一方面,由于参与者 $p$ 既是提议者也是接受者,所以 $p$ 需要维护状态 $(m,P)$,其中,$m$ 是 $p$ 承诺可以接受提议的最小编号,$P=\langle b,v\rangle$ 是 $p$ 最近一次接受的提议.另一方面,由于 $p$ 也是学习者,它需要在本地判断是否有值/提议已被选中,所以 $p$ 还需要维护它对其它所有参与者的观察状态.综上所述,在 TPaxos 中,每个参与者 $p$ 需要维护一个大小为 $N$ 的状态向量,即 $S^p$.

II. **统一消息内容.**在状态类型统一的基础上,我们进一步要求,每次 $p$ 与 $q$ 通信时,$p$ 将它的真实状态 $S_p^p$ 发送出去.需要注意的是,该步骤并未统一消息类型.这是因为,协议还需要在消息中添加类似"1a"、"2a"等标志信息,以区分多个子阶段.第 III 步推导将论述如何统一消息类型.另外,这步推导不要求发送 $p$ 对 $q$ 的观察状态 $S_q^p$.TPaxos 采用的消息类型 $M_{p\to q}=\left(S_p^p,S_q^p\right)$ 中的 $S_q^p$ 的作用将在第 IV 步推导中论述.

III. **统一消息类型.** Paxos 使用了消息类型以区分四个子阶段.为了统一这四种消息类型,我们先论证如何统一某些子阶段,然后讨论如何避免通过消息类型区分子阶段.

a) **将子阶段 Phase1b 与 Phase2b 统一为消息处理阶段.**

- 一方面,在 *Paxos* 规约的 *Phase2b(a)*动作中,接受者 $a$ 在接受类型为"2a"并且携带提议 $\langle m.bal,m.val\rangle$ 的消息 $m$ 时,不仅更新了 $\langle maxVBal[a],maxVVal[a]\rangle$,也同时将 *maxBal[a]*更新为 $m.bal$(条件 $m.bal\geq maxBal[a]$成立).也就是说,接受者 $a$ 在接受某提议时,可以同时作出"不再接受具有更小编号的提议"的承诺.
- 另一方面,经过第 II 步的推导,接受者 $q$ 收到的来自 $p$ 的信息 $S_p^p$ 不仅包含提议编号 $m$,还包含 $p$ 最近接受的提议 $P=\langle b,v\rangle$.根据 Paxos 的 Phase1b 阶段,如果条件 $S_q^q.m<S_p^p.m$ 成立,$q$ 就将 $S_q^q.m$ 更新为 $S_p^p.m$.然而,我们注意到,如果条件 $S_q^q.m<S_p^p.P.b$ 成立[1],那么 $q$ 也可以接受提议 $S_p^p.P$.

综上所述,在 Phase1b 与 Phase2b 子阶段,接受者都可以(在各自条件成立的情况下)既作承诺又接受提议.因此,我们可以将这它们统一为一个阶段,即消息处理阶段.需要注意的是,由于在第 II 步推导中,$p$ 不需要发送它对 $q$ 的观察状态 $S_q^p$,因此第 II 步推导得出的消息处理阶段只包含对 $S_q^q$ 的更新,而不包含对 $S_p^q$ 的更新.此外,$q$ 需要将 $S_q^q$ 回复给 $p$.与 TPaxos 最终版本不同的是,该回复

1 注意, $S_q^q.m$ 可能已被更新.

是无条件执行的.这将在第 IV 步推导中论述.

b) **消除消息标志**.经过前面的推导,目前我们得到的协议包含三个阶段,分别是准备阶段、接受阶段与消息处理阶段.为了区分各个阶段,(统一内容的)消息中仍需携带标志信息.下面,我们说明这些标志信息是可以避免的.

- 首先,在 Paxos 中,接受者通过判断收到的消息类型是"1a"还是"2a"来确定进入 Phase1b 还是 Phase2b 子阶段.由于在第 III-a)步推导中,Phase1b 与 Phase2b 被统一成消息处理阶段,所以不需要再区分消息类型"1a"与"2a".
- 其次,在 Paxos 的 *Phase2a(b,v)*阶段,提议者需要收集类型为"1b"且携带提议编号 $b$ 的消息.它们是由接受者发送的针对编号为 $b$ 的 Prepare 请求的回复.在 TPaxos 中,参与者 $p$ 可以通过 $S_q^p.m = b$ 是否成立判断 $q$ 是否发送了这样的回复.
- 最后,在 Paxos 中,接受者使用类型为"2b"的消息将它最新接受的提议通知给学习者.由于在 TPaxos 中,参与者同时具有三种角色,因此不再需要发送该类消息.参与者可以根据本地状态向量判断是否有提议已被选中.

IV. **优化消息处理阶段.**在第 III-III.a)步推导中,参与者在消息处理阶段会将自身真实状态无条件地回复给消息的发送者.这导致任意两个参与者之间都会无休止地互相发送消息,交换各自的状态信息.为了避免这种情况,TPaxos 作了如下优化.首先,$p$ 发送给 $q$ 的消息会携带 $p$ 对 $q$ 的观察状态.也就是说,统一的消息类型变成 $M_{p\to q}=\left(S_p^p,S_q^p\right)$.其次,在 $q$ 的消息处理阶段,只有当观察状态 $S_q^p$ 比更新后的 $S_q^q$ 状态要旧时,$q$ 才回复消息 $M_{q\to p}=\left(S_q^q,S_p^q\right)$ 给 $p$.

## 3 TPaxos 与 TPaxosAP 的 TLA+规约及证明

本节使用 TLA+描述 TPaxos.我们发现 TPaxos 协议描述中存在至关重要但并未充分阐明的微妙之处:在消息处理阶段,参与者是先作出"不再接受具有更小编号的提议"的承诺还是先接受提议?这可能导致对 TPaxos 的两种不同理解,并促使我们提出 TPaxos 的一种变体,称为 TPaxosAP.在 TPaxosAP 中,参与者先接受提议后作承诺.虽然在 TLA+规约层面,TPaxosAP 与 TPaxos(先作承诺后接受提议)相差甚小,但它却体现了不同的投票机制.具体来讲,Paxos 协议的投票机制 Voting[13] 仍适用于 TPaxos,却不能完整刻画 TPaxosAP 的行为.在本节,我们将举例说明 TPaxosAP 与 TPaxos 的不同之处,并论证 TPaxosAP 的正确性.

### 3.1 TPaxos的TLA+规约

#### 3.1.1 常量

与 Paxos 对应,TPaxos 的 TLA+规约也包含三个常量:

- *Value*:所有可能的提议值构成的集合(例如,$\{v_1,v_2,v_3\}$).*None* 表示不属于 *Value* 的某个值.
- *Participant*:所有参与者构成的集合(例如,$\{p_1,p_2,p_3\}$).每个参与者都同时是提议者、接受者与学习者.
- *Quorum*:由参与者形成的议会系统.

为了避免多个参与者使用相同的提议编号发起多个 Accept 请求,我们使用 *Bals(p)*为参与者预分配可用的提议编号[3] (如,*p1* 使用{1,4},*p2* 使用{2,5},*p3* 使用{3,6})[1].其中,*Ballot* 是所有可能的提议编号构成的集合,*NP* 是参与者数目,*PIndex* 为参与者分配了索引.

#### 3.1.2 变量

在 TPaxos 中,每个参与者维护一个状态向量,包含自身的实际状态以及它对其它参与者的观察状态.参与者之间交换具有统一类型的消息,不断更新状态向量,进而达成共识.因此,TPaxos 的 TLA+规约需要维护如下变量:

---

1 *Paxos* 规约未采用预分配提议编号的方式.这有两个原因:第一,在 *Phase1a(b)*与 *Phase2a(b,v)*阶段, 提议者并不改变自身状态,所以可由任意提议者执行.第二, 它在 *Phase2a(b,v)*中通过消息类型限制了最多只会产生一个针对提议编号 $b$ 的 *Accept* 请求.

MODULE *TPaxos*

EXTENDS *Integers*, *FiniteSets*

CONSTANTS
*Participant*, 参与者集合
*Value*, 参与者提出的值集合
*Quorum*

$None \triangleq$ CHOOSE $b : b \notin Value$ 不属于集合 *Value* 的特殊值
$NP \triangleq Cardinality(Participant)$ 参与者数量
$Quorum \triangleq \{Q \in \text{SUBSET } Participant : Cardinality(Q) * 2 \geq NP + 1\}$
ASSUME $QuorumAssumption \triangleq$
$\land \forall Q \in Quorum : Q \subseteq Participant$
$\land \forall Q1, Q2 \in Quorum : Q1 \cap Q2 \neq \{\}$
$Ballot \triangleq Nat$
$Max(m, n) \triangleq$ IF $m > n$ THEN $m$ ELSE $n$
$Injective(f) \triangleq \forall a, b \in \text{DOMAIN } f : (a \neq b) \Rightarrow (f[a] \neq f[b])$
$PIndex \triangleq$ CHOOSE $f \in [Participant \to 1 \,..\, NP] : Injective(f)$
$Bals(p) \triangleq \{b \in Ballot : b\%NP = PIndex[p] - 1\}$ 将编号 $b$ 分配给每个参与者

$State \triangleq [maxBal : Ballot \cup \{-1\},$
$maxVBal : Ballot \cup \{-1\}, maxVVal : Value \cup \{None\}]$
$InitState \triangleq [maxBal \mapsto -1, maxVBal \mapsto -1, maxVVal \mapsto None]$
$Message \triangleq [from : Participant,$
$to : \text{SUBSET } Participant, state : [Participant \to State]]$

VARIABLES
*state*, *state*[*p*][*q*]: 参与者 *p* 对参与者 *q* 的观察状态
*msgs* 所有发送消息的集合
$vars \triangleq \langle state, msgs \rangle$
$TypeOK \triangleq \land state \in [Participant \to [Participant \to State]]$
$\land msgs \subseteq Message$
$Init \triangleq \land state = [p \in Participant \mapsto [q \in Participant \mapsto InitState]]$
$\land msgs = \{\}$
$Send(m) \triangleq msgs' = msgs \cup \{m\}$

- *state*:全局状态矩阵.其中,$state[p][q]$表示参与者 $p$ 对参与者 $q$ 的观察状态(即 $S_q^p$).$state[p]$则表示 $p$ 维护的状态向量(即 $S^p$).每个状态(定义见 *State*)包含三个分量$(maxBal, maxVBal, maxVVal)$,含义与第 2.1 节中的$(m,b,v)$一一对应.
- *msgs*:所有已发送消息构成的集合.*Message* 表示所有可能的消息,它定义了格式统一的消息.需要注意的是,为了简化消息广播过程,我们选择每次发送整个状态向量.消息接收者可以根据需要提取协议规定的部分状态信息进行处理.

*TypeOK* 给出了变量 *state* 与 *msgs* 的类型约束.*Init* 给出了初始状态下变量 *state* 与 *msgs* 的值.

### 3.1.3 动作

TPaxos 包含三个动作,分别是发起准备请求、消息处理以及发起接受请求.

- *Prepare(p,b)*:参与者 $p$ 选取提议编号 $b$,发起 Prepare 请求.它要求 $b$ 是预分配给 $p$ 的编号,并且 $b > state[p][p].maxBal$ .$p$ 先将 $state[p][p].maxBal$ 更新为 $b$,然后广播当前本地状态向量 $state'[p]$.

参与者 $p$ 选择一个大于 $state[p][p].maxBal$ 的编号值 $b$ 开始 Prepare 阶段
$Prepare(p, b) \triangleq$
$\land b \in Bals(p)$
$\land state[p][p].maxBal < b$
$\land state' = [state \text{ EXCEPT } ![p][p].maxBal = b]$
$\land Send([from \mapsto p, to \mapsto Participant, state \mapsto state'[p]])$

- *OnMessage(q)*:参与者 $q$ 处理收到的消息 $m \in msgs$ .消息 $m$ 的发送者记为 $p \triangleq m.from$ .表达式 $m.state[p]$从消息 $m$ 中抽取 $p$ 发送的自身真实状态.$q$ 先根据 $m.state[p]$更新本地状态向量,包括 $q$ 的

真实状态 $state[q][q]$以及 $q$ 对 $p$ 的观察状态 $state[q][p]$;见 $UpdateState(q,p,pp \triangleq m.state[p])$.其中,前三条更新观察状态,后三条更新真实状态,如第 2.1 节所述,包括"先作承诺(将 $state[q][q].maxBal$ 更新为 $pp.maxBal$)后接受(分别将 $state[q][q].maxVBal$ 与 $state[q][q].maxVVal$ 更新为 $pp.maxVBal$ 与 $pp.maxVVal$)"两个步骤.然后,如果 $p$ 对 $q$ 的观察状态 $m.state[q]$比更新后的 $state'[q][q]$状态要旧,则 $q$ 广播当前本地状态向量 $state'[q]$.最后,为了避免 $q$ 不必要地重复处理消息 $m$,该规约将 $q$ 从 $m.to$ 中移除.

```
参与者 q 收到来自 p 的消息, 根据消息中携带的 pp 信息来更新自身状态 state[q], 该函数被函数
OnMessage(q) 调用.(pp = m.state[p], pp 可能不等于此时 p 的真实状态 state[p][p])
UpdateState(q, p, pp) ≜
    LET maxB ≜ Max(state[q][q].maxBal, pp.maxBal)
    IN  state' = [state EXCEPT
                  ![q][p].maxBal = Max(@, pp.maxBal),   作承诺
                  ![q][p].maxVBal = Max(@, pp.maxVBal),
                  ![q][p].maxVVal = IF state[q][p].maxVBal < pp.maxVBal
                                    THEN pp.maxVVal ELSE @,
                  ![q][q].maxBal = maxB,   先承诺后接受
                  ![q][q].maxVBal = IF maxB ≤ pp.maxVBal   接受
                                    THEN pp.maxVBal ELSE @,
                  ![q][q].maxVVal = IF maxB ≤ pp.maxVBal   接受
                                    THEN pp.maxVVal ELSE @]
参与者 q 收到消息并处理
OnMessage(q) ≜
    ∃ m ∈ msgs :
      ∧ q ∈ m.to
      ∧ LET p ≜ m.from
        IN  UpdateState(q, p, m.state[p])
      ∧ LET qm ≜ [from ↦ m.from, to ↦ m.to \ {q}, state ↦ m.state]
            nm ≜ [from ↦ q, to ↦ {m.from}, state ↦ state'[q]]   新消息
        IN  IF ∨ m.state[q].maxBal < state'[q][q].maxBal
               ∨ m.state[q].maxVBal < state'[q][q].maxVBal
            THEN msgs' = (msgs \ {m}) ∪ {qm, nm}
            ELSE msgs' = (msgs \ {m}) ∪ {qm}
```

- *Accept(p,b,v)*:参与者 $p$ 发起针对提议 $\langle b,v\rangle$ 的 Accept 请求.
  - 第一个前置条件要求提议编号 $b$ 是参与者 $p$ 的预分配编号,从而避免多个参与者使用相同的提议编号 $b$ 发起多个 Accept 请求.
  - 第二个前置条件避免参与者接受违背其承诺的提议.下面,我们通过示例论述该条件的必要性.假设有五个参与者 *p1*、*p2*、*p3*、*p4* 与 *p5*;见图 3.首先,*p1* 执行 *Prepare(1)*.*p2*、*p3*、*p4* 与 *p5* 收到该 Prepare 请求后更新状态并回复 *p1*.此时所有参与者的状态如图 3 第一行所示.接着,*p2* 执行 *Prepare(2)*.*p1* 和 *p3* 收到该 Prepare 请求后更新状态并回复 *p2*;见图 3 第二行.然后,*p2* 执行 *Accept(2, $v_1$)*.*p1* 收到该 Accept 请求并更新状态;见图 3 第三行.此时,*p1* 观察到存在议会 *p3*、*p4*、*p5*,它们的 *maxBal* 为 *1*.如果没有前置条件 $2 = state[p][p].maxBal \le b = 1$,则 *p1* 可以执行 *Accept(1, $v_1$)*,并接受提议 $\langle 1,v_1\rangle$.这违背了它"不再接受编号小于 $state[p][p].maxBal = 2$ 的提议"的承诺.
  - 前三个前置条件共同保证 *OneValuePerBallot* 成立,证明见第 3.3 节.其中,如果第三个前置条件不成立,即 $state[p][p].maxVBal = b$,则表示 $p$ 已经执行过一个 *Accept(p,b,v)*.然而,仅有第三个前置条件并不能避免 $p$ 多次执行 *Accept(p,b,v)*.这是因为,$p$ 可能会在之后的 *OnMessage* 动作中接受具有更大编号的提议,从而使得 $state[p][p].maxVBal > b$.

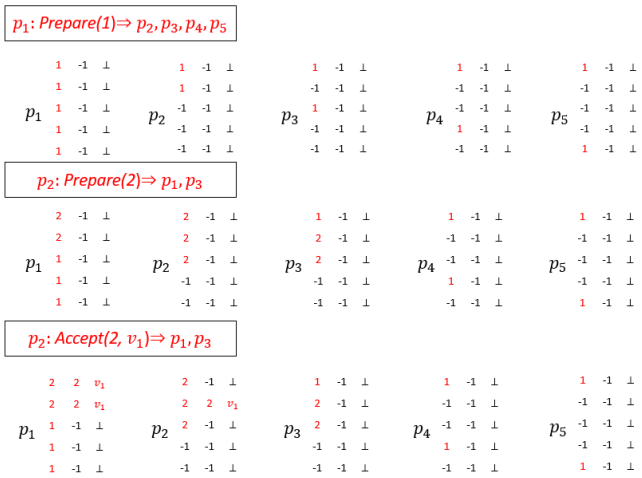


Figure 3: An illustration for the second precondition *state[p][p].maxBal* ≤ *b* of *Accept(p,b,v)*

图 3: 针对 *Accept(p,b,v)*的第二个前置条件 *state[p][p].maxBal* ≤ *b* 的示例

- 第四个前置条件要求存在某个议会 $Q \in Quorum$ ,使得 $Q$ 中的每个参与者 $q$ 都回复了针对 $b$ 的 Prepare 请求,即有 *state[p][q].maxBal=b*.
- 第五条合取式描述了根据本地状态向量 $state[p]$ 确定值 $v$ 的方法.第六条合取式更新 $p$ 的真实状态,即 $p$ 接受了提议 $\langle b,v \rangle$ .第七条合取式表示向其它参与者广播本地状态向量 $state'[p]$.

参与者 $p$ 选择提议编号 $b$($b$ 应该等于 Prepare 阶段的 $b$) 和值 $v$,$v$ 的选择和 Paxos 选择方式是相同的.

$$
\begin{aligned}
&Accept(p,\ b,\ v)\ \triangleq \\
&\quad \land\ b \in Bals(p) \\
&\quad \land\ state[p][p].maxBal \leq b \quad \text{对应于 } Voting \text{ 中第一条合取式} \\
&\quad \land\ state[p][p].maxVBal \neq b \quad \text{对应于 } Voting \text{ 中第二条合取式} \\
&\quad \land\ \exists\, Q \in Quorum : \quad \text{收集到了" 足够" 的针对 } Prepare(p,\ b) \text{ 的回复} \\
&\qquad\quad \land\ \forall\, q \in Q : state[p][q].maxBal = b \\
&\quad \text{从本地所有状态选值} \\
&\quad \land\ \lor\ \forall\, q \in Participant : state[p][q].maxVBal = -1 \quad \text{选择自己的值} \\
&\qquad \lor\ \exists\, q \in Participant : \quad v \text{ 是提议编号最大的提议对应的值} \\
&\qquad\quad \land\ state[p][q].maxVVal = v \\
&\qquad\quad \land\ \forall\, r \in Participant : state[p][q].maxVBal \geq state[p][r].maxVBal \\
&\quad \land\ state' = [state\ \text{EXCEPT}\ ![p][p].maxVBal = b, \\
&\qquad\qquad\qquad\qquad\quad ![p][p].maxVVal = v] \\
&\quad \land\ Send([from \mapsto p,\ to \mapsto Participant,\ state \mapsto state'[p]])
\end{aligned}
$$

### 3.1.4 行为

*Next* 定义了次态关系.*Spec* 定义了完整的行为规约:

$$
\begin{aligned}
&Next\ \triangleq\ \exists\, p \in Participant : \lor\ OnMessage(p) \\
&\qquad\qquad\qquad\qquad\qquad \lor\ \exists\, b \in Ballot : \lor\ Prepare(p,\ b) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad \lor\ \exists\, v \in Value : Accept(p,\ b,\ v) \\
&Spec\ \triangleq\ Init \land \Box[Next]_{vars}
\end{aligned}
$$

## 3.2 TPaxosAP的TLA+规约

TPaxosAP 与 TPaxos 的唯一不同在于,在消息处理阶段 *OnMessage(q)*的 *UpdateState(q,p,pp)*中,参与者 $q$ 先

接受提议⟨*pp.maxVBal*, *pp.maxVVal*⟩后作承诺.需要注意的是,在 TLA+规约中, 判断 $q$ 能否接受提议的条件 $state[q][q].maxBal \le pp.maxVBal$ 中使用的是当前值 $state[q][q].maxBal$,而不是像 TPaxos 那样使用了先作承诺而更新后的 *maxB*.

```
参与者 q 收到来自 p 的消息, 根据消息中携带的 pp 信息来更新自身状态 state[q], 该函数被函数
OnMessage(q) 调用.(pp = m.state[p], pp 可能不等于此时 p 的真实状态 state[p][p])
UpdateState(q, p, pp) ≜
  state' = [state EXCEPT
      ![q][p].maxBal = Max(@, pp.maxBal),                          作承诺
      ![q][p].maxVBal = Max(@, pp.maxVBal),
      ![q][p].maxVVal = IF state[q][p].maxVBal < pp.maxVBal
                        THEN pp.maxVVal ELSE @,
      ![q][q].maxBal = Max(@, pp.maxBal),                          作承诺
      ![q][q].maxVBal = IF state[q][q].maxBal ≤ pp.maxVBal         接受
                        THEN pp.maxVBal ELSE @,
      ![q][q].maxVVal = IF state[q][q].maxBal ≤ pp.maxVBal         接受
                        THEN pp.maxVVal ELSE @]
```

下面,我们举例说明 TPaxosAP 与 TPaxos 的不同之处.假设某参与者 $q$ 收到了来自 $p$ 的消息,消息内容包含 $\langle m,b,v\rangle \triangleq \langle state[p][p].maxBal, state[p][p].maxVBal, state[p][p].maxVVal\rangle$,且 $m > b$.如果 $p$ 的状态"领先"于 $q$,比如 $p$ 比 $q$ 多执行过几次 *Prepare* 与 *Accept*,则对 TPaxos 与 TPaxosAP 来说,$q$ 都可以在 *OnMessage(q)*中既作承诺又接受提议.具体而言,在 TPaxosAP 中,$\langle state[q][q].maxBal, state[q][q].maxVBal, state[q][q].maxVVal\rangle$ 被更新为 $\langle m,b,v\rangle$.这可以看作 $q$ 先将其更新为$\langle b,b,v\rangle$,然后再更新为$\langle m,b,v\rangle$,相当于 $q$ 先处理了针对提议$\langle b,v\rangle$的 Accept 请求,然后处理了针对提议编号 $m$ 的 Prepare 请求.然而,在 TPaxos 中,$q$ 会将它的真实状态更新为$\langle m,-,-\rangle$(– 表示相应分量保持不变).这相当于 $q$ 先处理了针对提议编号 $m$ 的 Prepare 请求,从而导致它不能再接受提议$\langle b,v\rangle$.

我们将在第 4 节中论述 TPaxosAP 和 TPaxos 代表两种不同的投票机制.从上述例子得出,TPaxosAP 与 TPaxos 在实际应用中没有较大区别.

### 3.3 TPaxos与TPaxosAP的正确性

我们证明 TPaxos 与 TPaxosAP 满足第 1.5 节介绍的两个条件 *OneValuePerBallot* 与 *SafeAt(b, v)*,从而保证了一致性(Consistency)条件.*SafeAt(b, v)*由 *Accept* 动作的第四和第五个合取式(确定值 $v$)保证,其证明与 Lamport 使用 TLAPS 证明 Paxos 的子阶段 Phase2a 保证了 *SafeAt(b,v)*[21] 类似.下面,我们证明 TPaxos 与 TPaxosAP 满足 *OneValuePerBallot*.该证明仅用到 *Accept* 动作的前三个前置条件,因此对 TPaxos 与 TPaxosAP 都适用.我们先介绍两个简单的引理.

- (引理 1)对任意参与者 $p$,它接受过的提议编号 *state[p][p].maxVBal* 是(非严格)单调递增的.
- (引理 2)对任意参与者 $p$,它承诺可以接受提议的最小编号始终大于等于它接受过的提议编号,即满足 $state[p][p].maxBal \ge state[p][p].maxVBal$.

*OneValuePerBallot* 保证对于任意提议编号 $b$,最多对应一个提议$\langle b,v\rangle$,即最多执行一次 *Accept(_,b,_)*(_表示该分量可为任意值).注意到 *Accept(_,b,_)*的第一个前置条件 $b \in Bals(p)$ 要求,只有 $b$ 的拥有者 $p$ 才能使用 $b$ 执行 *Accept(p,b,_)*,故只需要保证参与者 $p$ 最多执行一次 *Accept(p,b,_)*即可保证 *OneValuePerBallot*.

假设在时刻 $t$,参与者 $p$ 执行了 *Accept(p,b,_)*,此时 $state[p][p].maxVBal = b$.由 *Accept(p,b,_)*的第三个前置条件 $state[p][p].maxVBal \ne b$ 可知,$p$ 不能在 $state[p][p].maxVBal = b$ 的情况下执行 *Accept(p,b,_)*.根据引理 1,在任意时刻 $t' \ge t$,都有 $state[p][p].maxVBal \ge b$.当 $state[p][p].maxVBal > b$ 时,根据引理 2 可得,$state[p][p].maxBal \ge state[p][p].maxVBal > b$.由于 *Accept(p,b,_)*的第二个前置条件 $state[p][p].maxBal \le b$ 的限制,$p$ 在时刻 $t' \ge t$ 也不能执行 *Accept(p,b,_)*.综上所述,对于提议编号 $b$,它的拥有者 $p$ 最多只能使用 $b$ 执行一次 *Accept(p,b,_)*,从而保证了 *OneValuePerBallot*.

## 4 TPaxos 与 TPaxosAP 的精化

本节建立从 TPaxos 到 Consensus 以及 TPaxosAP 到 Consensus 的精化关系.对于 TPaxos,我们仍采用 Paxos 所使用的 Voting[13] 机制(第 1.5 节),建立了从 TPaxos 到 Voting 的精化关系.对于 TPaxosAP,我们发现 Voting 并不能完整刻画 TPaxosAP 的行为.因此,我们首先提出了一种新的投票机制,称作 EagerVoting.EagerVoting 允许参与者在接受提议的同时,作出比 Paxos/TPaxos 更"激进"的"不再接受具有更小编号的提议"的承诺.然后,我们建立了从 TPaxosAP 到 EagerVoting 以及从 EagerVoting 到 Consensus 的精化关系.

需要注意的是,在 *TPaxos* 与 *TPaxosAP* 规约的 *Accept(p,b,v)*动作中,参与者 *p* 在前置条件满足的情况下,需要根据本地状态向量 *state[p]*确定值 *v*.然而,在 *Voting*(第 1.5 节)与 *EagerVoting*(将在第 4.2.1 节介绍)的 *VoteFor(a,b,v)*动作中,参与者需要根据某个议会的状态(而不是所有参与者)确定值 *v*.这并不会影响协议的正确性.但是,不难论证,存在某些情况使得两种方案确定的 *v* 值不同,这给构建从 *TPaxos* 到 *Voting* 以及从 *TPaxosAP* 到 *EagerVoting* 的精化关系造成了障碍.为此,在本节,我们修改 *TPaxos* 与 *TPaxosAP* 的 *Accept(p,b,v)*动作,改用"从议会确定值 *v*"的方案[1].修改后的 *Accept(p,b,v)*动作如下所示.

```
参与者 p 选择提议编号 b(b 应该等于 Prepare 阶段的 b) 和值 v,v 的选择和 Paxos 选择方式是相同的.
Accept(p, b, v) ≜
  ∧ b ∈ Bals(p)
  ∧ state[p][p].maxBal ≤ b    对应于 Voting 中第一条合取式
  ∧ state[p][p].maxVBal ≠ b   对应于 Voting 中第二条合取式
  从本地一个议会中的状态选值
  ∧ ∃ Q ∈ Quorum :   收集到了" 足够" 的针对 Prepare(p, b) 的回复
    ∧ ∀ q ∈ Q : state[p][q].maxBal = b
    ∧ ∨ ∀ q ∈ Participant : state[p][q].maxVBal = −1   自己的值
      ∨ ∃ q ∈ Participant :   v 是提议编号最大的提议对应的值
          ∧ state[p][q].maxVVal = v
          ∧ ∀ r ∈ Participant : state[p][q].maxVBal ≥ state[p][r].maxVBal
  ∧ state′ = [state EXCEPT ![p][p].maxVBal = b,
                           ![p][p].maxVVal = v]
  ∧ Send([from ↦ p, to ↦ Participant, state ↦ state′[p]])
```

### 4.1 TPaxos的精化

*TPaxos* 采取了与 *Paxos* 相同的投票机制,都可以用 *Voting*[21] 规约(第 1.5 节)来刻画.本节构建从 *TPaxos* 到 *Voting* 的精化关系.*TPaxos* 中的动作与 *Voting* 中的动作存在着明确的对应关系.具体而言:

- 在 *TPaxos* 的 *Preapare(p,b)*中,参与者 *p* 将 *state[p][p].maxBal* 增加至更大的提议编号 *b*.这对应于 *Voting* 中的 *IncreaMaxBal(a,b)*动作(此处 *a=p*).具体而言,*Prepare(p,b)*中的前置条件 $state[p][p].maxBal < b$ 对应于 *IncreaMaxBal(a,b)* 中的前置条件 $maxBal[a] < b$.另外,*Prepare(p,b)* 中的状态更新(将 $state[p][p].maxBal$ 更新为 *b*)也对应于 *IncreaMaxBal(a,b)*中的状态更新(将 $maxBal[a]$ 更新为 *b*).
- 在 *TPaxos* 的 *Accept(p,b,v)*中,参与者 *p* 在确定值 *v* 之后,随即接受了提议 ⟨*b*,*v*⟩.这对应于 *Voting* 中的 *VoteFor(a,b,v)*动作(此处 *a=p*).具体而言:
  - *Accept(p,b,v)* 中的第二个前置条件 $state[p][p].maxBal \le b$ 与第三个前置条件 $state[p][p].maxVBal \ne b$ 分别对应于 *VoteFor(a,b,v)*中的第一个前置条件 $maxBal[a] \le b$ 与第二个前置条件 $\forall vt \in votes[a] : vt[1] \ne b$.
  - *VoteFor(a,b,v)*中的第三个前置条件 $\forall c \in Acceptor \setminus \{a\} : \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v)$ 与第二个前置条件共同保证 *OneValuePerBallot*.在 *Accept(p,b,v)*中,*OneValuePerBallot* 由它的前三个前置条件共同保证.

---

1 我们将如下两个相关问题列入未来工作:第一,如何修改 *Voting* 的 *VoteFor* 动作,使其采用"从全体参与者确定值 *v*"的方案?第二,如何在采用不同("确定值 *v*"的)方案的 *Voting* 之间建立精化关系?

- *Accept(p,b,v)*"从议会 $Q \in Quorum$ 确定值 *v*"的方法与 *VoteFor(a,b,v)*中的 *ShowsSafeAt(Q,b,v)*对应.
- *Accept(p,b,v)*中的状态更新(分别将 $state[p][p].maxVBal$ 与 $state[p][p].maxVVal$ 更新为 *b* 与 *v*)对应于 *VoteFor(a,b,v)*中对 *votes* 的更新(将提议 $\langle b,v\rangle$ 加入到 $votes[a]$中),表示参与者 *a*(即 *p*)接受了提议 $\langle b,v\rangle$ .
- *VoteFor(a,b,v)*会将 $max[a]$更新为 *b*.对于 TPaxos,参与者 *p* 可以执行 *Accept(p,b,v)*,表明 *p* 已执行过 *Prepare(p,b)* 并更新了 $state[p][p].maxBal = b$ .根据第 3.1.3 节对前置条件 $state[p][p].maxBal \le b$ 的分析可知,在 *p* 执行 *Accept(p,b,v)*时, $state[p][p].maxBal = b$ 仍然成立.

■ 现在考虑 *TPaxos* 中的 *OnMessage(q)*动作(第 3.1 节).参与者 *q* 收到了来自 $p \triangleq m.from$ 的消息 $m \in msgs$ .记 $pp \triangleq m.state[p]$为 *m* 中携带的 *p* 的真实状态,即 $(pp.maxBal, pp.maxVBal, pp.maxVVal)$ .在 *UpdateState(q,p,pp)*中,*q* 需要根据 *pp* 更新自身真实状态 $state[q][q]$.这里可能有两种更新结果:第一种是,只"作承诺",即将 $state[q][q].maxBal$ 更新至 $pp.maxBal$.此时,"接受提议"的条件不再成立.这种情况对应于 *Voting* 中的 *IncreaMaxBal(a,b)*动作(此处 *a=p*,*b=pp.maxBal*);第二种是,"作承诺"且"接受提议",即将 $state[q][q].maxBal$ 更新至 $pp.maxBal$,并且分别将 $state[q][q].maxVBal$ 与 $state[q][q].maxVVal$ 更新为 $pp.maxVBal$ 与 $pp.maxVVal$. 这要求 $state'[q][q].maxBal = pp.maxBal = pp.maxVBal$ 成立( $state'[q][q].maxBal$ 为"作承诺"更新后的值).因此,这种情况对应于 *Voting* 中的 *VoteFor(a,b,v)*动作(此处 *a=p*,*b=pp.maxVBal*,*v=pp.maxVVal*).

```
------------------------ MODULE TPaxosWithVotes ------------------------
EXTENDS TPaxos
VARIABLE votes   votes[q]: 参与者 q 接受过的提议集合
varsV ≜ ⟨vars, votes⟩
--------------------------------------------------------------------------
InitV ≜
        ∧ Init
        ∧ votes = [q ∈ Participant ↦ {}]
TypeOKV ≜
          ∧ votes ∈ [Participant → SUBSET (Ballot × Value)]
          ∧ TypeOK
--------------------------------------------------------------------------
PrepareV(p, b) ≜
          ∧ Prepare(p, b)
          ∧ votes' = votes
AcceptV(p, b, v) ≜
        ∧ Accept(p, b, v)
        ∧ votes' = [votes EXCEPT ![p] = @ ∪ {⟨b, v⟩}]   收集提议 ⟨b, v⟩
OnMessageV(q) ≜
        ∧ OnMessage(q)
        ∧ IF state'[q][q].maxVBal ≠ state[q][q].maxVBal   接受了新提议
          THEN votes' = [votes EXCEPT ![q] = @ ∪   收集接受的新提议
                                {⟨state'[q][q].maxVBal, state'[q][q].maxVVal⟩}]
          ELSE UNCHANGED votes
--------------------------------------------------------------------------
NextV ≜ ∃ p ∈ Participant :
            ∨ OnMessageV(p)
            ∨ ∃ b ∈ Ballot : ∨ PrepareV(p, b)
                             ∨ ∃ v ∈ Value : AcceptV(p, b, v)
SpecV ≜ InitV ∧ □[NextV]_varsV
--------------------------------------------------------------------------
maxBal ≜ [p ∈ Participant ↦ state[p][p].maxBal]
V ≜ INSTANCE Voting WITH Acceptor ← Participant
                         votes ← votes, maxBal ← maxBal
THEOREM SpecV ⇒ V!Spec
```

为了构建从 *TPaxos* 到 *Voting* 的精化关系,我们需要在 *TPaxos* 规约中模拟 *Voting* 中的变量 *maxBal* 与 *votes*,即参与者承诺可以接受提议的最小编号以及已接受的提议构成的集合;见模块 *TPaxosWithVotes*.从上面的分析可以得出,*Voting* 中的 $maxBal[p]$对应于 *TPaxos* 中的 $state[p][p].maxBal$ .因此,针对 *maxBal*,我们定义精化映射

为 $maxBal \triangleq [p \in Participant \mapsto state[p][p].maxBal]$.对于 *votes*,我们在 *TPaxos* 的基础上添加辅助变量[15] [22] *votes*(在 *TPaxosWithVotes* 模块中,故与 *Voting* 中的 *votes* 冲突),为每个参与者 *p* 收集它已接受过的提议 $votes[p]$.具体而言,在 *Prepare(p,b)*中,参与者 *p* 未接受任何提议,所以在 *PrepareV(p,b)*中,*votes* 保持不变.在 *Accept(p, b, v)* 中,参与者 *p* 接受了提议 $\langle b,v\rangle$ ,所以在 *AcceptV(p,b,v)*中,我们将 $\langle b,v\rangle$ 添加到 $votes[p]$.在 *OnMessage(q)*中,如果 $state[q][q].maxVBal$ 发生了改变,则表明 *q* 接受了新的提议[1].在 *OnMessageV(q)* 中,我们将该提议添加到 $votes[q]$.最后,*TPaxosWithVotes* 模块给出了从 *TPaxos* 到 *Voting* 的精化映射(即用辅助变量替换 *Voting* 中的对应变量): $V \triangleq INSTANCE\ Voting\ WITH\ Acceptor \leftarrow Participant, maxBal \leftarrow maxBal, votes \leftarrow votes$ .

```
MODULE EagerVoting
EXTENDS Sets

CONSTANT Value, Acceptor, Quorum
ASSUME QuorumAssumption ≜
          ∧ ∀ Q ∈ Quorum : Q ⊆ Acceptor
          ∧ ∀ Q1, Q2 ∈ Quorum : Q1 ∩ Q2 ≠ {}
Ballot ≜ Nat

VARIABLES votes, maxBal
TypeOK ≜ ∧ votes ∈ [Acceptor → SUBSET (Ballot × Value)]
         ∧ maxBal ∈ [Acceptor → Ballot ∪ {−1}]

VotedFor(a, b, v) ≜ ⟨b, v⟩ ∈ votes[a]
DidNotVoteAt(a, b) ≜ ∀ v ∈ Value : ¬VotedFor(a, b, v)
ShowsSafeAt(Q, b, v) ≜
      ∧ ∀ a ∈ Q : maxBal[a] ≥ b  承诺过
      ∧ ∃ c ∈ −1 .. (b − 1) :
          ∧ (c ≠ −1) ⇒ ∃ a ∈ Q : VotedFor(a, c, v)
          ∧ ∀ d ∈ (c + 1) .. (b − 1), a ∈ Q : DidNotVoteAt(a, d)

Init ≜
      ∧ votes = [a ∈ Acceptor ↦ {}]
      ∧ maxBal = [a ∈ Acceptor ↦ −1]
IncreaseMaxBal(a, b) ≜
      ∧ maxBal[a] < b
      ∧ maxBal′ = [maxBal EXCEPT ![a] = b]  作承诺
      ∧ UNCHANGED votes
EagerVoting 和 Voting 的唯一区别在于: 在 Voting 中, 我们只更新 maxBal 为 b,
即 maxBal′ = [maxBal EXCEPT ![a] = b].
VoteFor(a, b, v) ≜
        ∧ maxBal[a] ≤ b  不违背承诺
        ∧ ∀ vt ∈ votes[a] : vt[1] ≠ b
        ∧ ∀ c ∈ Acceptor \ {a} :
            ∀ vt ∈ votes[c]  : (vt[1] = b) ⇒ (vt[2] = v)
        ∧ ∃ Q ∈ Quorum : ShowsSafeAt(Q, b, v)  提议 ⟨b, v⟩ 是安全的, 能够投票
        ∧ votes′ = [votes EXCEPT ![a] = votes[a] ∪ {⟨b, v⟩}]  投票
        ∧ ∃ c ∈ Ballot :
            ∧ c ≥ b
            ∧ maxBal′ = [maxBal EXCEPT ![a] = c]  作承诺

Next ≜
      ∃ a ∈ Acceptor, b ∈ Ballot :
          ∨ IncreaseMaxBal(a, b)
          ∨ ∃ v ∈ Value : VoteFor(a, b, v)
Spec ≜ Init ∧ □[Next]⟨votes, maxBal⟩
```

### 4.2 TPaxosAP的精化

我们先介绍适用于 TPaxosAP 的投票机制 EagerVoting,然后建立从 TPaxosAP 到 EagerVoting 以及从

1 这是因为 *OneValuePerBallot* 成立.

EagerVoting 到 Consensus 的精化关系.

4.2.1 EagerVoting 规约

根据第 4.1 节的分析,在 *TPaxos* 中,每个参与者 *p* 要么仅提高它承诺不再接受的最小提议编号 *state[p][p].maxBal*,要么在接受提议 $\langle b,v\rangle$ 的同时也将 *state[p][p].maxBal* 更新为 *b*.这与 *Voting* 规约是相符的.然而,*Voting* 并不能完全刻画 *TPaxosAP* 的行为.如第 3.2 节所述,*TPaxosAP* 与 *TPaxos* 的行为在 *OnMessage(q)*消息处理阶段的状态更新 *UpdateState(q,p,pp)* 方面有所不同:在 *TPaxosAP* 中,参与者先接受提议 $\langle pp.maxVBal, pp.maxVVal\rangle$ (即,更新 *state[q][q].maxVBal* 与 *state[q][q].maxVVal*) 后作承诺 (即,将 *state[p][p].maxBal* 提高至 *pp.maxBal*).这可能导致更新后的 *state[p][p].maxBal* 不等于 *pp.maxVBal*,从而违反了 *Voting* 规约.这种情况是有可能发生的.考虑如下示例:

假设有三个参与者 *p1*、*p2*、*p3*,*p1* 先执行 *Prepare(1)*.*p2*、*p3* 收到该 Prepare 请求后更新状态并回复 *p1*.接着,*p1* 执行 *Accept(1, $v_1$)*以及 *Prepare(2)*,并发送携带 $\langle 2,1,v_1\rangle$ 的 Prepare 请求.如果 *p2* 与 *p3* 收到该请求并将自身状态更新为 $\langle 2,1,v_1\rangle$,则违反了 *Voting* 规约.

为了弥补 *Voting* 的不足,我们提出一种新的适用于 TPaxosAP 的投票机制,称为 *EagerVoting*. *EagerVoting* 与 *Voting* 的唯一不同在于:在 *VoteFor(a,b,v)*动作中, 我们允许参与者 *a* 在接受提议 $\langle b,v\rangle$ 的同时,将 $maxBal[a]$提高到比 *b* 更大的提议编号 $c \geq b$.

直观地讲,*Voting* 与 *EagerVoting* 是等价的.一方面,由于 *Voting* 的每个动作都是 *EagerVoting* 所允许的,因此 *Voting* 的每个行为也都是 *EagerVoting* 所允许的.另一方面,如果 *EagerVoting* 的动作 *VoteFor(a,b,v)*在更新 $maxBal[a]$时选择的提议编号 *c* 等于 *b*,则该动作与 *Voting* 中的 *VoteFor(a,b,v)*等价;如果 *c* 大于 *b*,则该动作可以看作 *Voting* 中的 *VoteFor(a,b,v)*与 *IncreaseMaxBal(a,c)*的组合.因此,*EagerVoting* 的每个行为都可以被 *Voting* 中的行为所模拟[1].

4.2.2 从 EagerVoting 到 Consensus 的精化

为了构建从 *EagerVoting* 到 *Consensus* 的精化关系,我们需要在 *EagerVoting* 中模拟 *Consensus* 规约中的变量 *chosen*,即协议已选中的值构成的集合;见 *EagerVotingWithChosen*.一个提议 *b*,*v* 以及它包含的值被选中当且仅当该存在某个议会 $Q \in Quorum$ 的接受者都接受了该提议;见 *ChosenAt(b,v)*.辅助变量 *chosen* 则收集了所有被选中的值. 最后,该模块给出了从 *EagerVoting* 到 *Consensus* 的精化映射 $C \triangleq INSTANCE\ Consensus\ WITH\ Acceptor \leftarrow Acceptor, chosen \leftarrow chosen$,即用辅助变量 *chosen* 替换 *Consensus* 中的变量 *chosen*.

```
MODULE EagerVotingWithChosen
EXTENDS EagerVoting

ChosenAt(b, v) ≜
    ∃ Q ∈ Quorum : ∀ a ∈ Q : VotedFor(a, b, v)
chosen ≜ {v ∈ Value : ∃ b ∈ Ballot : ChosenAt(b, v)}
Consistency ≜ chosen = {} ∨ ∃ v ∈ Value : chosen = {v}   |chosen| ≤ 1
C ≜ INSTANCE Consensus WITH chosen ← chosen, Acceptor ← Acceptor
THEOREM Refinement ≜ Spec ⇒ C!Spec
```

4.2.3 从 TPaxosAP 到 EagerVoting 的精化

第 4.1 节描述的 *TPaxos* 和 *Voting* 之间的对应关系在 *TPaxosAP* 和 *EagerVoting* 之间仍然成立.需要注意的是,根据第 4.2.1 节的论述,*TPaxosAP* 的 *OnMessage(q)*动作对"先接受提议后作承诺"的改动对应于

[1] 如果要使用精化技术严格证明 EagerVoting 可以被 Voting 所以模拟,需要找到从 EagerVoting 到 Voting 的精化映射.我们将其列为未来工作.

*EagerVoting* 的 *VoteFor(a,b,v)*动作对"允许将 $maxBal[a]$ 提高到 $c \geq b$ "的改动.因此, *TPaxosAP* 中的动作与 *EagerVoting* 中的动作也存在着明确的对应关系.从 *TPaxosAP* 到 *EagerVoting* 的精化关系与从 *TPaxos* 到 *Voting* 的精化关系相同,构建方法也相同,此处不再赘述.

## 5 模型检验

本节使用 TLC 模型检验工具验证 TPaxos 与 TPaxosAP 算法的正确性,并且验证了从 TPaxos 到 Voting 及从 TPaxosAP 到 EagerVoting 的精化关系的正确性.

### 5.1 实验设置

在所有的实验中[1],我们调整了参与者集合 Participant,提议值集合 Value,提议编号集合 Ballot 的大小,并将前两者设置为对称集[10] 以提高 TLC 的验证效率.我们使用了 10 个线程进行了实验,以下是我们的实验统计结果:已遍历(BFS 方式遍历)的系统状态图的直径,TLC 已检验的所有状态的数量,TLC 已检验的不同状态的数量以及检验时间(格式为 hh:mm:ss).当 TLC 已检验的不同状态数的数量超过某个阈值(设置为 1 亿[2]),我们就人为地停止该次实验.

实验所用的机器配置为:

- 机器 1:2.40 GHz GPU, 6 核以及 64GB 内存,TLC [23] 版本号为 1.6.0.负责 TPaxos 相关实验.
- 机器 2:2.10 GHz GPU, 6 核以及 64GB 内存,TLC 版本号 1.6.0.负责 TPaxosAP 相关实验.

### 5.2 验证结果

#### 5.2.1 TPaxos 与 TPaxosAP 满足 Consistency

图 4 与图 5 分别给出了在不同配置下验证 TPaxos 及 TPaxosAP 满足一致性的结果.总体而言,在相同配置下,这两种协议的模型检验结果相似.在不同的配置下,参与者数量和提议编号个数对协议的规模影响较大.相对而言,提议值的个数对协议的规模影响较小.

| TLC 模型<br>(参与者数量, 提议值个数, 投票轮数) | 状态图直径 | 状态数 | 不同状态数 | 检验时间<br>$(hh:mm:ss)$ |
|---|---|---|---|---|
| $(2,2,2)$ | 20 | 27809 | 7991 | $0:00:01$ |
| $(2,2,3)$ | 31 | 69954174 | 12107912 | $0:07:03$ |
| $(2,4,2)$ | 20 | 27945 | 7991 | $0:00:02$ |
| $(2,4,3)$ | 31 | 69966456 | 12107912 | $0:17:35$ |
| $(3,2,2)$ | 21 | 322114689 | 100000019 | $0:45:04$ |
| $(3,2,3)$ | 18 | 270946706 | 100000022 | $0:42:09$ |
| $(3,4,2)$ | 22 | 319890347 | 100000023 | $2:38:37$ |
| $(3,4,3)$ | 17 | 279673321 | 100000031 | $2:28:30$ |

Figure 4: Model checking results of verifying that TPaxos satisfies Consitency

图 4: TPaxos 满足一致性的验证结果

1 基于 Lamport 给出的从 Voting 到 Consensus 的 TLAPS 证明[21] ,我们使用 TLAPS 定理证明系统[42] 证明了从 EagerVoting 到 Consensus 的精化关系的正确性[44] .

2 以 3 个提议编号, 3 个提议值和 3 个参与者为例,此时变量 *state* 总共有 27 个分量,粗略估计状态数最多有 3 的 27 次方(约为万亿量级).

| TLC 模型 (参与者数量, 提议值个数, 投票轮数) | 状态图直径 | 状态数 | 不同状态数 | 检验时间 $(hh:mm:ss)$ |
|---|---|---|---|---|
| $(2,2,2)$ | 20 | 27809 | 7991 | $0:00:01$ |
| $(2,2,3)$ | 31 | 69954296 | 12107912 | $0:06:52$ |
| $(2,4,2)$ | 20 | 27945 | 7991 | $0:00:02$ |
| $(2,4,3)$ | 31 | 69966578 | 12107912 | $0:18:20$ |
| $(3,2,2)$ | 21 | 329426700 | 100602814 | $0:44:04$ |
| $(3,2,3)$ | 19 | 274590666 | 101397482 | $0:42:03$ |
| $(3,4,2)$ | 21 | 323929358 | 100147908 | $2:40:06$ |
| $(3,4,3)$ | 17 | 278887421 | 100000024 | $2:28:05$ |

Figure 5: Model checking results of verifying that TPaxosAP satisfies Consistency

图 5: TPaxosAP 满足一致性的验证结果

5.2.2 TPaxos 与 TPaxosAP 满足 Liveness

TPaxos 与 TPaxosAP 的活性要求只有一个提议者,即只允许某个参与者 $p$ 来执行动作 *Prepare(p, b)*及 *Accept(p, b, v)*.以 TPaxos 为例,对于 TPaxos 的任何一个阶段,一旦其异常终止(即,没有提议被选中)参与者 $p$ 将会选择增大提议编号 $b$ 重新开始新的 TPaxos 过程.$b$ 将最终大于所有参与者维护的值 *maxBal*,从而,两个阶段将执行完成并且有提议最终被选中.

在 TLA+中,一般使用公平性来验证活性.表达式 *Fairness* 做出了相应的限制.参与者 $p$ 为唯一的提议者,*MaxBallot* 是最大的提议编号.参与者 $p$ 选择 *MaxBallot* 发起 *Prepare(p, MaxBallot)*和 *Accept(p, MaxBallot, v)* 动作,第五条合取式保证了议会 $Q$ 能接受 Prepare 请求和 Accept 请求(参与者 $p$ 是根据本地状态判断能否进入 Accept 阶段,条件 $p \in Q$ 允许 $p$ 和 $Q$ 中的参与者通信以更新本地状态).表达式 *Liveness* 定义了 TPaxos 的活性,即最终一定会有值被选中.

$$
\begin{aligned}
&Fairness \triangleq \\
&\quad \wedge \exists p \in Participant : \\
&\qquad \wedge MaxBallot \in Bals(p) \\
&\qquad \wedge \mathrm{WF}_{vars}(Prepare(p, MaxBallot)) \\
&\qquad \wedge \forall v \in Value : \mathrm{WF}_{vars}(Accept(p, MaxBallot, v)) \\
&\qquad \wedge \exists Q \in Quorum : \\
&\qquad\quad \wedge p \in Q \\
&\qquad\quad \wedge \forall q \in Q : \mathrm{WF}_{vars}(OnMessage(q))
\end{aligned}
$$

$$Liveness \triangleq \Diamond(chosen \neq \{\})$$

图 6 与图 7 分别给出了多种配置下验证 TPaxos 与 TPaxosAP 满足活性的结果.由于使用公平性来验证活性的方法增加了对动作的约束,因此状态数相对而言较少.

| TLC 模型 (参与者数量, 提议值个数, 投票轮数) | 状态图直径 | 状态数 | 不同状态数 | 检验时间 $(hh:mm:ss)$ |
|---|---|---|---|---|
| $(2,2,2)$ | 15 | 385 | 275 | $0:00:01$ |
| $(2,2,3)$ | 22 | 27366 | 13684 | $0:00:02$ |
| $(2,4,2)$ | 15 | 735 | 523 | $0:00:01$ |
| $(2,4,3)$ | 22 | 54192 | 27042 | $0:00:03$ |
| $(3,2,2)$ | 34 | 206443509 | 62284985 | $2:27:08$ |
| $(3,4,2)$ | 34 | 444157073 | 134403809 | $6:21:41$ |

Figure 6: Model checking results of verifying that TPaxos satisfies Liveness

图 6: TPaxos 满足活性的验证结果

| TLC 模型 (参与者数量, 提议值个数, 投票轮数) | 状态图直径 | 状态数 | 不同状态数 | 检验时间 $(hh:mm:ss)$ |
|---|---|---|---|---|
| $(2,2,2)$ | 15 | 385 | 275 | 0 : 00 : 01 |
| $(2,2,3)$ | 22 | 27372 | 13684 | 0 : 00 : 02 |
| $(2,4,2)$ | 15 | 735 | 523 | 0 : 00 : 01 |
| $(2,4,3)$ | 22 | 54204 | 27042 | 0 : 00 : 02 |
| $(3,2,2)$ | 34 | 206248747 | 62215595 | 1 : 17 : 38 |
| $(3,4,2)$ | 34 | 442881157 | 133987469 | 16 : 28 : 41 |

Figure 7: Model checking results of verifying that TPaxosAP satisfies Liveness

图 7: TPaxosAP 满足活性的验证结果

5.2.3 TPaxos 与 TPaxosAP 的精化

图 8 与图 9 分别给出了多种配置下验证从 TPaxos 到 Voting 与从 TPaxosAP 到 EagerVoting 精化关系的正确性.由于该组实验检验的性质包含时序操作符,不再是定义在单个状态上的不变式,验证算法较为复杂,所以需要消耗更多的时间.

| TLC 模型 (参与者数量, 提议值个数, 投票轮数) | 状态图直径 | 状态数 | 不同状态数 | 检验时间 $(hh:mm:ss)$ |
|---|---|---|---|---|
| $(2,2,2)$ | 20 | 27809 | 7991 | 0 : 00 : 02 |
| $(2,2,3)$ | 31 | 69954174 | 12107912 | 0 : 12 : 32 |
| $(2,4,2)$ | 20 | 27945 | 7991 | 0 : 00 : 02 |
| $(2,4,3)$ | 31 | 69966456 | 12107912 | 0 : 55 : 24 |
| $(3,2,2)$ | 21 | 321158078 | 100000010 | 0 : 48 : 36 |
| $(3,2,3)$ | 17 | 268108874 | 100000034 | 1 : 10 : 50 |
| $(3,4,2)$ | 22 | 316078356 | 100000012 | 12 : 56 : 45 |
| $(3,4,3)$ | 17 | 278338521 | 100000015 | 11 : 08 : 10 |

Figure 8: Model checking results of verifying that TPaxos refines Voting

图 8: TPaxos 精化 Voting 的验证结果

| TLC 模型 (参与者数量, 提议值个数, 投票轮数) | 状态图直径 | 状态数 | 不同状态数 | 检验时间 $(hh:mm:ss)$ |
|---|---|---|---|---|
| $(2,2,2)$ | 20 | 27809 | 7991 | 0 : 00 : 01 |
| $(2,2,3)$ | 31 | 69954296 | 12107912 | 0 : 15 : 42 |
| $(2,4,2)$ | 20 | 27945 | 7991 | 0 : 00 : 03 |
| $(2,4,3)$ | 31 | 69966578 | 12107912 | 0 : 59 : 24 |
| $(3,2,2)$ | 22 | 315969776 | 100254890 | 2 : 09 : 05 |
| $(3,2,3)$ | 17 | 267572227 | 100000032 | 2 : 11 : 25 |
| $(3,4,2)$ | 22 | 318518236 | 100000012 | 13 : 13 : 31 |
| $(3,4,3)$ | 17 | 278245333 | 100000036 | 11 : 31 : 35 |

Figure 9: Model checking results of verifying that TPaxosAP refines EagerVoting

图 9: TPaxosAP 精化 EagerVoting 的验证结果

## 6 相关工作

Paxos[3] [4] 协议衍生出了许多变体[24] [25] [26] ,如 Disk Paxos[27] 、Cheap Paxos[28] 、Fast Paxos[19] 、Generalized Paxos[29] 、Stoppable Paxos[30] 、Vertical Paxos[31] 、Byzantine Paxos[24] 、EPaxos(Egalitarian Paxos)[32] 、Raft[33] 、CASPaxos[34] 等.本文关注腾讯开发的分布式存储系统 PaxosStore 中实现的 TPaxos 变体.TPaxos 的新颖之处在于它的"统一性":它为每个参与者维护统一的状态类型,并采用类型统一的消息进行通信.我们从 Paxos 出发,论证了如何逐步推导出 TPaxos.基于这种推导,我们可以将 TPaxos 看作 Paxos 的一种自

然变体.

使用形式化规约语言描述分布式协议并使用相应的模型检验工具进行验证可以有效提高协议的可信度[35].近年来,研究者使用 TLA+/TLC 描述并验证了 Paxos 协议及其多种变体.Lamport 等人使用 TLA+分别描述了 Paxos[13]、Fast Paxos[19]、Disk Paxos[27]以及 Byzantine Paxos[36],并使用 TLC 在一定规模上验证了它们的正确性. Moraru 在博士论文中给出了 EPaxos 的 TLA+规约.最近,Sutra 发现并纠正了其中的错误[37].Ongaro 给出了 Raft 协议的 TLA+规约[38].在本文,我们使用 TLA+描述了 TPaxos.在开发规约的时候,我们发现 TPaxos 协议描述中存在至关重要但并未完全阐明的微妙之处:在消息处理阶段,参与者是先作出"不再接受具有更小编号的提议"的承诺还是先接受提议?这促使我们发现了 TPaxos 的一种变体,称之为 TPaxosAP.与 TPaxos 相比,TPaxosAP 改动很小,但却体现了一种不同于 Voting[13]的投票机制.

精化(Refinement)技术[14] [15]有助于理解 Paxos 各种变体的正确性以及它们之间的关系.Lampson[24]提出了一个抽象的 Paxos 协议(Abstract Paxos;记为 AP).AP 刻画了 Paxos 协议的核心,但是由于它使用了全局信息,无法直接在分布式系统中实现.接着,作者使用精化/模拟(refinement/simulation)技术分别建立了 Paxos、Disk Paxos 以及 Byzantine Paoxs 与 AP 的关系.在 AP 的视角下,它们都可以看作 AP 的具体实现.Lamport 等人提出了抽象的投票机制 Voting[13],用于刻画接受者"作承诺"与"接受提议"两种核心行为.Voting 可以看作分布式共识问题的集中式解决方案,而 Paxos 是 Voting 的分布式实现.实际上,Lamport 等人给出了从 Paxos 到 Voting 以及从 Voting 到 Consensus 的精化映射[13].同样基于 Voting,Lamport 还使用精化技术从 Paxos 推导出了 Byzantine Paxos[24],并使用 TLC 验证了它们之间的精化关系.Maric[39]等人在 HO(Heard-Of)模型[40]下研究了多种 Paxos[39]变体.作者首先为它们建立了一个共同的抽象,也称为 Voting.根据 Voting 行为的差异,这些 Paxos 变体可以被归为三类.然后,作者构建了这三类变体之间的精化关系,并使用定理证明器 Isabelle/HOL 进行了形式化验证.在本文,我们分别建立了从 TPaxos 以及 TPaxosAP 到 Consensus 的精化关系.特别地,在 TPaxosAP 方面,我们首先在 Voting 的基础上提出了一种适用于 TPaxosAP 的投票机制 EagerVoting,然后建立了从 TPaxosAP 到 EagerVoting 以及从 EagerVoting 到 Consensus 的精化关系.

## 7 总结与未来工作

本文深入研究了 PaxosStore 系统中的 TPaxos 协议[8]. TPaxos 的新颖之处在于它的"统一性":它为每个参与者维护统一的状态类型,并采用统一类型的消息进行通信.首先,我们从经典的 Paxos 协议出发,论证如何逐步推导出 TPaxos 协议.基于这种推导,我们可以将 TPaxos 看作 Paxos 的一种自然变体.其次,我们给出了 TPaxos 的 TLA+规约.我们发现 TPaxos 协议描述中存在至关重要但并未完全阐明的微妙之处.这促使我们提出了它的一种变体,称为 TPaxosAP.最后,我们分别建立了从 TPaxos 以及 TPaxosAP 到 Consensus 的精化关系.特别地,在 TPaxosAP 方面,我们在 Voting 的基础上提出了一种适用于 TPaxosAP 的投票机制 EagerVoting,并使用 TLC 模型检验工具验证了从 TPaxosAP 到 EagerVoting 以及从 EagerVoting 到 Consensus 的精化关系的正确性.

目前,我们正在使用 TLAPS[11] [41] [42]定理证明系统开发机器可检验的证明.另外,我们计划对 PaxosStore 进行更全面的研究.比如,我们将采用形式化方法(如形式化规约、精化技术、模型检验与定理证明等)研究共识层中另外两个模块的正确性:实现了 Multi-Paxos[3]功能的 PaxosLog 机制以及允许故障切换(failover)的一致性读写协议.

**References**

[1] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. J. ACM 32, 2 (April 1985), 374–382. DOI:https://doi.org/10.1145/3149.214121

[2] Maurice Herlihy. Wait-free synchronization. ACM Trans. Program. Lang. Syst. 13, 1 (January 1991), 124–149. DOI:https://doi.org/10.1145/114005.102808

[3] Leslie Lamport. Paxos made simple[J]. ACM Sigact News, 2001, 32(4): 18-25.

[4] Leslie Lamport. The part-time parliament. ACM Trans. Comput. Syst. 16, 2 (May 1998), 133–169. DOI:https://doi.org/10.1145/279227.279229

[5] Tushar D. Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: an engineering perspective. In Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing (PODC '07). Association for Computing Machinery, New York, NY, USA, 398–407. DOI:https://doi.org/10.1145/1281100.1281103

[6] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's Globally Distributed Database. ACM Trans. Comput. Syst. 31, 3, Article 8 (August 2013), 22 pages. DOI:https://doi.org/10.1145/2491245

[7] Michael Isard. Autopilot: automatic data center management. SIGOPS Oper. Syst. Rev. 41, 2 (April 2007), 60–67. DOI:https://doi.org/10.1145/1243418.1243426

[8] Jianjun Zheng, Qian Lin, Jiatao Xu, Cheng Wei, Chuwei Zeng, Pingan Yang, and Yunfan Zhang. PaxosStore: high-availability storage made practical in WeChat. Proc. VLDB Endow. 10, 12 (August 2017), 1730–1741. DOI:https://doi.org/10.14778/3137765.3137778

[9] Jianjun Zheng. The PaxosStore System. https://github.com/Tencent/paxosstore (Accessed Sep 5, 2019)

[10] Leslie Lamport. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[11] Leslie Lamport. The TLA+ Hyperbook, 2019. http://lamport.azurewebsites.net/tla/hyperbook.html (Accessed Sep 5, 2019)

[12] Leslie Lamport. The temporal logic of actions. ACM Trans. Program. Lang. Syst. 16, 3 (May 1994), 872–923. DOI:https://doi.org/10.1145/177492.177726

[13] Leslie Lamport, Stephan Merz, and Doligez D. A TLA + specification of Paxos and its refinement. https://github.com/tlaplus/Examples/tree/master/specifications/Paxos (Accessed Sep 5, 2019)

[14] Abadi Martin, and Leslie Lamport. The existence of refinement mappings[J]. Theoretical Computer Science, 1991, 82(2): 253-284.

[15] Leslie Lamport, Stephan Merz. Auxiliary variables in TLA+[J]. arXiv preprint arXiv:1703.05121, 2017.

[16] Yu Yuan, Panagiotis Manolios, and Leslie Lamport. Model checking TLA+ specifications[C]//Advanced Research Working Conference on Correct Hardware Design and Verification Methods. Springer, Berlin, Heidelberg, 1999: 54-66.

[17] Leslie Lamport. Summary of tla+. http://lamport.azurewebsites.net/tla/summary-standalone.pdf (Accessed Sep 5, 2019)

[18] Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao, Yongle Zhang, Pranay U. Jain, and Michael Stumm. Simple testing can prevent most critical failures: an analysis of production failures in distributed data-intensive systems. In Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation (OSDI'14). USENIX Association, USA, 249–265.

[19] Leslie Lamport. Fast paxos[J]. Distributed Computing, 2006, 19(2): 79-103.

[20] Leslie Lamport, Stephan Merz. A TLA+ specification of Paxos Consensus algorithm described in Paxos Made Simple and a TLAPS-checked proof of its correctness. https://github.com/tlaplus/v2-tlapm/blob/master/examples/paxos/Paxos.tla (Accessed Sep 5, 2019)

[21] Leslie Lamport, Stephan Merz. A TLA+ specification of Voting algorithm and a TLAPS-checked proof of its correctness. https://github.com/tlaplus/v2-tlapm/blob/master/examples/consensus/Voting.tla (Accessed Sep 5, 2019)

[22] Howard Heidi, and Richard Mortier. A Generalised Solution to Distributed Consensus[J]. arXiv preprint arXiv:1902.06776, 2019.

[23] Microsoft Research. The TLA Toolbox. http://lamport.azurewebsites.net/tla/toolbox.html (Accessed Sep 5, 2019)

[24] Leslie Lamport. Byzantizing Paxos by refinement[C]//International Symposium on Distributed Computing. Springer, Berlin, Heidelberg, 2011: 211-224.

[25] Butler Lampson. The ABCD's of Paxos. In Proceedings of the twentieth annual ACM symposium on Principles of distributed computing (PODC '01). Association for Computing Machinery, New York, NY, USA, 13.DOI:https://doi.org/10.1145/383962.383969

[26] Robbert Van Renesse and Deniz Altinbuken. Paxos Made Moderately Complex. ACM Comput. Surv. 47, 3, Article 42 (February 2015), 36 pages. DOI:https://doi.org/10.1145/2673577

[27] Gafni Eli, Leslie Lamport. Disk paxos[J]. Distributed Computing, 2003, 16(1): 1-20.

[28] Leslie Lamport, and Mike Massa. Cheap paxos[C]//International Conference on Dependable Systems and Networks, 2004. IEEE, 2004: 307-314.

[29] Leslie Lamport. Generalized consensus and paxos[J]. Tech. Rep. MSR-TR-2005-33, Microsoft Research, 2005.

[30] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Stoppable paxos[J]. TechReport, Microsoft Research, 2008.

[31] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Vertical paxos and primary-backup replication. In Proceedings of the 28th ACM symposium on Principles of distributed computing (PODC '09). Association for Computing Machinery, New York, NY, USA, 312–313. DOI:https://doi.org/10.1145/1582716.1582783

[32] Iulian Moraru, David G. Andersen, and Michael Kaminsky. There is more consensus in Egalitarian parliaments. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13). Association for Computing Machinery, New York, NY, USA, 358–372. DOI:https://doi.org/10.1145/2517349.2517350

[33] Ongaro Diego, and John Ousterhout. In search of an understandable consensus algorithm[C]//2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14). 2014: 305-319.

[34] Rystsov Denis. CASPaxos: Replicated State Machines without logs[J]. arXiv preprint arXiv:1802.07000, 2018.

[35] Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis. Model checking: algorithmic verification and debugging. Commun. ACM 52, 11 (November 2009), 74–84. DOI:https://doi.org/10.1145/1592761.1592781

[36] Leslie Lamport. The PlusCal Code for Byzantizing Paxos by Refinement[J]. TechReport, Microsoft Research, 2011.

[37] Sutra Pierre. On the correctness of Egalitarian Paxos[J]. arXiv preprint arXiv:1906.10917, 2019.

[38] Ongaro Diego. A TLA + specification of raft. https://github.com/ongardie/raft.tla (Accessed Sep 5, 2019)

[39] Maric Ognjen, Christoph Sprenger, and David Basin. Consensus refined[C]//2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. IEEE, 2015: 391-402.

[40] Charron-Bost Bernadette, and André Schiper. The heard-of model: computing in distributed systems with benign faults. Distributed Computing 22.1 (2009): 49-71.

[41] Microsoft Research. TLAPS website. http://tla.msr-inria.inria.fr/tlaps/ (Accessed Sep 5, 2019)

[42] Kaustuv Chaudhuri, Damien Doligez, Leslie Lamport, Stephan Merz. A TLA+ Proof System. ArXiv, 2008,abs/0811.1914.

[43] Wang J, Zhang M, Wu Y, Chen K, Zheng W. Paxos-like consensus algorithms: a review[J]. Journal of Computer Research and Development, 2019, 56(04): 692-707.

[44] Yi X, Wei H, Huang Y, Qiao L, Lu J. TLAPS proof for the refinement from EagerVoting to Consensus. https://github.com/Starydark/PaxosStore-tla/blob/master/specification/EagerVoting.tla (Accessed Sep 5, 2019)

**附中文参考文献**:

[43] 王江,章明星,武永卫,陈康,郑纬民.类 Paxos 共识算法研究进展.计算机研究与发展,2019,56(4):692-707.