软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(6):1748−1778 [doi: 10.13328/j.cnki.jos.006248]

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

支持乱序执行的 Raft 协议*

谷晓松[1], 魏恒峰[1], 乔 磊[2], 黄 宇[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, Email: hfwei@nju.edu.cn; 乔磊, Email: fly2moon@aliyun.com

摘 要: PolarFS 是阿里巴巴开发的分布式文件系统,它实现了分布式共识协议 Raft 的一种变体,称为 ParallelRaft. ParallelRaft 突破了 Raft 中顺序提交、顺序执行的限制,允许状态机乱序执行用户命令.然而文献表明:ParallelRaft 并未开源,仅有简短的文字描述,更缺乏严格的形式化规约.更进一步,它的正确性也尚未经过必要的数学论证或形式化检验.旨在为 ParallelRaft 提供严格的形式化规约并证明其正确性,主要贡献包括:首先,为了理清 ParallelRaft 与 Raft 之间的关系,提出了允许乱序提交、顺序执行的 ParallelRaft-SE(sequential execution)协议,并建立了从 ParallelRaft-SE 到 Multi-Paxos 的精化关系;其次,现有的 ParallelRaft 描述忽略了可能会违反状态一致性的“幽灵日志”问题,因此在 ParallelRaft-SE 的基础上提出了 ParallelRaft-CE(concurrent execution)协议.ParallelRaft- CE 限制了 ParallelRaft-SE 在乱序提交阶段的并行度,避免了“幽灵日志”问题,支持乱序执行,并保证乱序执行下的状态机一致性.证明了 ParallelRaft-CE 的正确性.最后,使用 TLA+给出了 ParallelRaft-SE 和 ParallelRaft-CE 的形式化规约,并对协议参与者数量较小的情形,使用 TLC 模型检验与模拟测试工具验证了从 ParallelRaft-SE 到 Multi-Paxos 的精化关系以及 ParallelRaft-CE 的正确性.

关键词: Raft;ParallelRaft;Multi-Paxos;共识协议;TLA+;精化关系;模型检验

中图法分类号: TP311

中文引用格式: 谷晓松,魏恒峰,乔磊,黄宇.支持乱序执行的 Raft 协议.软件学报,2021,32(6):1748−1778. http://www.jos.org.cn/1000-9825/6248.htm

英文引用格式: Gu XS, Wei HF, Qiao L, Huang Y. Raft with out-of-order executions. Ruan Jian Xue Bao/Journal of Software, 2021,32(6):1748−1778 (in Chinese). http://www.jos.org.cn/1000-9825/6248.htm

Raft with Out-of-order Executions

GU Xiao-Song[1], WEI Heng-Feng[1], QIAO Lei[2], HUANG Yu[1]

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

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

Abstract: PolarFS is a distributed file system developed by Alibaba Inc. with ultra-low latency and high availability. It implemented a variant of the Raft consensus protocol, called ParallelRaft. ParallelRaft breaks Raft's strict serialization restrictions in the commitment and execution of log entries and enables state machines to commit and execute log entries in an out-of-order way. However, ParallelRaft is not open-sourced. It has only a brief description, lacking formal specification. Moreover, the correctness of ParallelRaft has not been manually proven or formally checked. The purpose of the study is to provide a precise formal specification for ParallelRaft and to prove

* 基金项目: 国家自然科学基金(61932021, 61772258); 五〇二所空间先进计算与电子信息专业实验室开放基金(OBCandETL-2020-04)

Foundation item: National Natural Science Foundation of China (61932021, 61772258); Space Advanced Computing and Electronic Information Laboratory of BICE (OBCandETL-2020-04)

本文由“形式化方法与应用”专题特约编辑邓玉欣教授推荐.

收稿时间: 2020-08-30; 修改时间: 2020-10-26, 2021-01-18; 采用时间: 2021-01-27; jos 在线出版时间: 2021-02-07

its correctness. Specifically, the following main contributions are accomplished. First, to clarify the relationship between Raft and ParallelRaft, ParallelRaft-SE (sequential execution) is proposed, which allows out-of-order commitment but prohibits out-of-order executions. Also, a refinement mapping from ParallelRaft-SE to Multi-Paxos is established. Second, it is discovered that ParallelRaft, according to its brief description in the literature, neglects the so-called “ghost log entries” phenomenon, which may violate the consistency among state machines. Therefore, based on ParallelRaft-SE, ParallelRaft-CE (concurrent execution) is proposed. ParallelRaft-CE avoids the “ghost log entries” phenomenon and ensures the consistency among state machine when executing concurrently by limiting parallelism in the commitment of log entries. The correctness of ParallelRaft-CE is proved manually. Finally, the formal specifications of ParallelRaft-SE and ParallelRaft-CE using TLA+ (TLA stands for temporal logic of actions) are provided, and the refinement mapping from ParallelRaft-SE to Multi-Paxos and the correctness of ParallelRaft-CE are verified using the TLC model checker when the number of participants of the protocols is small.

Key words: Raft; ParallelRaft; Multi-Paxos; consensus protocols; TLA+; refinement; model checking

分布式共识问题是分布式计算领域的核心问题,它要求多个参与者对某个值或一系列值(也称序列)达成一致[1,2].分布式系统通常采用共识协议提供所需的强一致性,如开源 Ceph[3]、Google 公司的 Spanner[4]、Oracle 公司的 MySQL[5]、腾讯的 PaxosStore[6]以及阿里巴巴(Alibaba)的 PolarDB[7]等分布式数据存储系统.

Multi-Paxos(Paxos)[8,9]与 Raft[10]是解决分布式共识问题的两种经典协议,它们都基于复制状态机(replicated state machine)[11]模型,通过投票、日志复制等机制,保证多副本节点上的状态一致性.每个用户命令都将经历“提交”(commit)与“执行”(execute)两个阶段:如果某命令收到来自多数派(Majority)服务器的投票,则称该命令被“提交”.只有被提交的命令才能被执行.对于多个命令,这两个阶段都可以以顺序或乱序的方式进行.在乱序方式下,较大日志编号所对应的命令可能会在较小日志编号所对应的命令之前被提交或者被执行.比如,Raft 要求顺序提交、顺序执行,Multi-Paxos 则允许乱序提交,但它仍要求顺序执行.

PolarDB 使用了分布式文件系统 PolarFS[7].为了提高系统性能,PolarFS 基于 Raft 实现了允许乱序提交、乱序执行的 ParallelRaft 共识协议.然而文献中并未给出 ParallelRaft 的严格规约,尤其是缺少对乱序执行机制的完整描述.此外,ParallelRaft 尚未经过必要的数学证明或形式化验证.本文旨在为 ParallelRaft 提供严格的形式化规约,并基于精化关系[12]以及数学论证证明其正确性.具体而言,本文的主要贡献如下:

- 首先,为了理清 ParallelRaft 与 Raft 之间的关系,我们基于 Raft 提出了允许乱序提交,但要求顺序执行的 ParallelRaft-SE(sequential execution)协议.ParallelRaft 可以看作 ParallelRaft-SE 的乱序执行版本.建立了从 ParallelRaft-SE 到 Multi-Paoxs 之间的精化关系(表明 ParallelRaft-SE 是 Multi-Paxos 的一种实现),从而证明了 ParallelRaft-SE 的正确性;
- 其次,我们发现文献中描述的 ParallelRaft 乱序执行机制忽略了可能会破坏状态一致性的“幽灵日志”问题.已有研究表明:在顺序执行方式下(如 Raft 或 Multi-Paxos),“幽灵日志”问题不会破坏副本间的状态一致性.相反地,我们将论证:在乱序执行方式下,它可能造成副本间状态不一致.更进一步地指出,针对 Raft 和 Multi-Paxos 的解决方案无法解决乱序执行方式下的状态机的一致性问题;
- 在 ParallelRaft-SE 的基础上提出了支持乱序执行的 ParallelRaft-CE(concurrent execution)协议.通过限制 ParallelRaft-SE 在乱序提交阶段的并行度,ParallelRaft-CE 避免了“幽灵日志”问题.我们证明了 ParallelRaft-CE 的正确性;
- 最后,使用 TLA+[13–15]语言给出了 ParallelRaft-SE 与 ParallelRaft-CE 的形式化规约,并对协议参与者数量较小的情形,使用 TLC[16]模型检验工具验证了从 ParallelRaft-SE 到 Multi-Paxos 的精化关系以及 ParallelRaft-CE 的正确性.

本文第 1 节介绍预备知识,包括 TLA+形式化规约语言、分布式共识问题以及 Multi-Paxos 与 Raft 共识协议.第 2 节介绍 ParallelRaft-SE 协议,并建立从 ParallelRaft-SE 到 Multi-Paxos 的精化关系.第 3 节分析在乱序执行方式下,“幽灵日志”问题对状态一致性的影响,并指出现有解决方案的不足.第 4 节介绍允许乱序执行且避免了“幽灵日志”问题的 ParallelRaft-CE 协议.第 5 节简要证明 ParallelRaft-CE 的正确性.第 6 节使用 TLC 模型检验工具,验证从 ParallelRaft-SE 到 Multi-Paxos 的精化关系以及 ParallleRaft-CE 的正确性.第 7 节讨论相关工作.

第 8 节总结全文并讨论可能的未来工作.完整的 TLA+规约与模型检验结果参见 GitHub 仓库[17].

1 预备知识

1.1 TLA+简介

TLA+是由 Leslie Lamport 基于时序逻辑(temporal logic of action)[15]开发的形式化规约语言[13],尤其适合于描述并发系统或分布式协议.

一个 TLA+规约包含一组变量、初始状态(initial state)和一组动作(action),通常表示为 $Spec=Init\wedge[Next]_{vars}$,其中,*vars* 是所有变量的集合.状态(state)是对所有变量的赋值.*Init* 谓词定义了系统的初始状态,*Next* 是所有动作的析取式,定义了状态之间的转换关系.$[Next]_{vars}$ 为真当且仅当 *Next* 为真(某个动作为真,即某个动作被执行)或者所有变量的值保持不变.行为(behavior)是由状态构成的序列.TLA+使用不带撇的变量表示当前状态中变量的值,用带撇的变量表示新状态中的值.这样,动作使用一个包含带撇变量与不带撇的变量的公式描述.例如,动作 $x'=x+1$ 表示变量 x 在新状态中的值比旧状态中的值大 1.

TLA+支持一阶谓词逻辑以及 ZF(zermelo-fraenkel)集合论,可以表达丰富的数据类型[18,19].图 1 总结了本文用到的逻辑与集合操作符(operator).文献[20]给出了完整的 TLA+操作符列表.

	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 (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 中的所有元素得到的集合, 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
	$[f \text{ EXCEPT } ![e_1] = e_2]$	函数 $\bar{f} : \bar{f}[e] = \begin{cases} e_2, & \text{if } e = e_1 \\ f[e], & \text{otherwise} \end{cases}$
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 $\bar{r}\,[h_1 : S_1, \ldots, h : e, \ldots, h_n : S_n]$
	$[r \text{ EXCEPT } !h = e]$, 其中 e 包含符号 @	e 中的 @ 表示 $r.h$
元组	$e[i]$	元组 e 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”)

Fig.1 A summary of the TLA+ operators used in this paper

图 1 本文使用的 TLA+操作符

TLA+允许以模块(module)的方式进行相互引用.每个模块中,可以声明常量(CONSTANTS)与变量(VARIABLES),定义操作符(operator)或者提出定理(THEOREM)[18,19].一个模块可以通过扩展(EXTEND)命令来引入其他模块中的声明、定义与定理.引入的模块可以实例化.比如,模块 M 引入了模块 M_1.

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

其中,p_i 包含模块 M_1 中的所有常量与变量,e_i 是由 M 中的常量与变量定义的合法表达式.该语句将 M_1 中的 p_i 替换为相应的 e_i.我们可以通过 $IM_1!F$ 访问模块 M_1 中的表达式 F.当 e_i 与 p_j 相同时,TLA+的隐式替换规则允许我们省略 $p_j \leftarrow e_i$[18,19].

TLC[16]是 TLA+的模型检验工具,它可以遍历所有可能的系统行为,检查所有的状态,验证系统是否满足特

定性质.然而有些分布式系统包含无穷的状态,比如 TLA+规约中常常包含自然数,而自然数是无穷的.为了验证这样的系统,TLA+引入模型(model).模型所有的集合都是有穷的,因此系统的状态数目是有穷的.模型检验常常面临组合爆炸的问题,为此,TLC 可以利用模型中的对称性缩减状态空间.例如,假设 CONSTANTS *Server* 定义了系统中所有进程的集合.在进行模型检验时,我们需要将它实例化为一个有穷的集合,比如 $Server=\{S_1,S_2,S_3\}$.如果 S_1,S_2,S_3 之间的任意排列(比如 S_1 替换 S_2,S_2 替换 S_3,S_3 替换 S_1)都不会影响系统规约满足给定的性质,那么我们可以将 *Server* 设置为对称集(symmetry set)[13,21].

在 TLA+中,精化(refinement)关系[12]用来描述模块之间的逻辑蕴含(logical implication)关系[18].精化关系由精化映射(refinement mapping)定义.比如,一个由模块 *ImplModule* 中的 *ImplSpec* 规约到模块 *AbsModule* 中的 *AbsSpec* 规约的精化映射 ϕ 将 *AbsSpec* 中的每个变量 v 对应于一个表达式 $\overline{v}$,$\overline{v}$ 由 *ImplSpec* 中的变量定义.对于 *ImplSpec* 的每一个状态 s,精化映射 ϕ 定义了 *AbsSpec* 的状态 $\overline{s}$,$\overline{s}$ 中的每个变量 v 的值由 s 中的 $\overline{v}$ 定义.若 σ 为 *ImplSpec* 的行为 $s_1\rightarrow s_2\rightarrow\ldots$,我们定义 *AbsSpec* 的行为 $\overline{\sigma}:\overline{s}_1\rightarrow\overline{s}_2\rightarrow\ldots$.*ImplSpec* 在精化映射 ϕ 下实现/精化了 *AbsSpec*(*ImplSpec*⇒*AbsSpec*)当且仅当对于任何一个满足 *ImplSpec* 的行为 σ,行为 $\overline{\sigma}$ 满足规约 *AbsSpec*[14].为了使用 TLC 检验在精化映射 ϕ 下,*ImplSpec* 到的精化关系,我们在模块 *ImplModule* 中添加了定义 *AbsSub*≜INSTANCE *AbsModule*,并检验定理 THEOREM *ImplSpec*⇒*AbsSub*!*AbsSpec*[18].

1.2 分布式共识

分布式共识要求多个副本服务器节点保持状态一致.每个服务器节点可以建模为一个复制状态机,通过执行用户命令进行状态转换.

通常使用复制日志的机制实现复制状态机.每个服务器保存一份日志.日志由按顺序编号(通常是自然数)的日志项组成.每个日志项存储了一条来自用户的命令.服务器每次按顺序从日志中读取下一条已经取得共识的命令,在状态机上执行,并将结果返回给用户.由于服务器按照编号顺序执行一条命令,我们称此为顺序执行.传统的分布式共识协议(Multi-Paxos,Raft)都是顺序执行.在此条件下,我们假设副本服务器上的复制状态机具有相同的初始状态,那么只需要保证日志之间的一致性,就可以保证服务器之间的状态一致.因此,分布式共识问题可以转化为保证不同副本服务器节点上日志之间的一致性.具体而言,它要求:

- 非平凡性(nontriviality):只能对用户发出的命令取得共识;
- 一致性(consistency):每个位置最多只能对一条命令取得共识.

服务器通过运行共识协议保证日志的一致性.本文考虑异步消息传递系统中的共识协议,其故障模型(failure model)如下.

- 服务器可能终止执行(fail by stop),但不会出现拜占庭错误[8];
- 消息可能被延时、乱序到达、丢失或重复,但消息的内容不会被篡改.

1.3 Paxos协议

Paxos 协议是解决分布式共识问题的经典协议,它允许一组服务器对单个值(即单个日志项)取得共识,是 Multi-Paxos 的基础.Paxos 中定义了 3 种角色:提议者(proposer)、从节点(acceptor)和学习者(learner).提议者提出值(value),接受者选择值,学习者学习被选中的值.Paxos 包括两个阶段,每个阶段又包括两个子阶段[8].

- 准备(prepare)阶段(也称第 1 阶段,Phase1)
  - 子阶段 Phase1a:提议者选择一个全局唯一的编号 b(通常是自然数),向所有接受者发送编号为 b 的 Prepare 请求;
  - 子阶段 Phase1b:接受者收到编号为 b 的 Prepare 请求.如果它之前收到过编号大于 b 的 Prepare 请求,则忽略该编号为 b 的请求;否则,接受者将自己接受过的提议(包括编号与值)中编号最大的回复给提议者;
- 接受(accept)阶段(也称第 2 阶段,Phase2)
  - 子阶段 Phase2a:提议者收到来自超过半数的接受者发送的针对编号为 b 的 Prepare 请求的回复.

如果回复中不包含任何提议,则提议者可以任选一个值(通常是该提议者收到的用户命令);否则,提议者选择编号最大的回复对应的值.设提议者选择的值为 v,则提议者向所有接受者发送内容为$\langle b,v\rangle$的 Accept 请求;

- 子阶段 Phase2b:接受者收到编号为 b 的 Accept 请求.如果接受者没有收到过编号大于 b 的请求,则可以接受该 Accept 请求;否则,接受者忽略该 Accept 请求.

1.4 Multi-Paxos协议

Multi-Paxos 针对每个日志项运行一个独立的 Paxos 实例,从而支持副本服务器对日志(日志项的序列)达成共识.由于各个 Paxos 实例是并行独立运行的,Multi-Paxos 允许乱序提交用户命令.也就是说,它允许副本服务器先就编号较大的日志项对应的用户命令达成共识,而无需等待之前的日志项完成共识.

在实际实现时,常常通过将 phase1 的消息分批(batching)的方式提高性能.此时,系统包含一个恢复阶段,恢复阶段本质上是将不同实例的 phase1 消息集中处理的过程.

模块 *MultiPaxos* 描述了 Multi-Paxos 协议.其中包括 3 个常量.

- *Acceptors*:所有接受者构成的集合;
- *Value*:所有可能的提议值构成的集合;
- *Nil*:特殊记号,不属于 *Value*.

我们使用自然数表示可能的提议编号以及每个实例的编号,即 *Ballots*≜*Nat*,*Instances*≜*Nat*.*Quorums* 定义了一种特殊的议会系统(quorum system):它的每个元素都是由超过半数的接受者构成的集合.

Multi-Paxos 的规约中包括 6 个变量.

- *ballot*:*ballot*[*a*]表示接受者 *a* 记录的最大的提议编号,也是 *a* 可以接受的最小的提议编号;
- *vote*:*vote*[*a*][*i*][*b*]表示接受者 *a* 对编号为 *i* 的实例在 *a* 的提议编号为 *b* 时接受的提议值.如果 *a* 在提议编号为 *b* 时,没有对编号为 *i* 的实例接受任何值,则 *vote*[*a*][*i*][*b*]=*Nil*;
- *leaderVote*:*leaderVote*[*b*][*i*]为提议者在提议编号为 *b* 时,为编号为 *i* 的实例提出的值与提议编号 *b* 组成的二元组.初始状态下,对所有的提议编号 *b* 与实例编号 *i*,*leaderVote*[*b*][*i*]=$\langle -1,Nil\rangle$;
- 1*amsgs*,1*bmsgs*,2*amsgs*:不同类型的消息集合.

---------- **MODULE** MultiPaxos ----------

EXTENDS *Integers, FiniteSets*

CONSTANTS *Acceptors, Nil, Value*

Ballots==*Nat*

Instances==*Nat*

Quorums=={*Q*\in **SUBSET** *Acceptors*: *Cardinality*(*Q*)>*Cardinality*(*Acceptors*)\2}

Max(*s*)==**CHOOSE** *x*\in *s*:\for all *y*\in *s*:*x*\geq *y*

VARIABLES *ballot, vote, leaderVote,* 1*amsgs,* 1*bmsgs,* 2*amsgs*

协议定义的主要动作包括:

- *Phase*1*a*(*b*):对应 Paxos 的 *Phase*1*a* 阶段.提议者选取提议编号 *b*,并且向其他节点发送 Prepare 请求,提议的编号为 *b*;
- *Phase*1*b*(*a*,*b*):对应 Paxos 的 *Phase*1*b* 阶段.节点 *a* 收到编号为 *b* 的投票请求.若 *b*>*ballot*[*a*],*a* 设置 *ballot*[*a*]为 *b*,并且对于每一个实例编号 *i*,*a* 将它接受过的提议编号最大的提议值与提议编号回复给提议者.

---

*Phase*1*a*(*b*)==

/\1*amsgs*′=1*amsgs*\cup {⟨⟨*b*⟩⟩}

/**UNCHANGED** ⟨⟨*ballot*,*vote*,*leaderVote*,1*bmsgs*,2*amsgs*⟩⟩

MaxAcceptorVote(*a*,*i*)==
LET *maxBallot*==Max({*b*\in *Ballots*:*vote*[*a*][*i*][*b*]#*Nil*}\cup {−1})
v==**IF** *maxBallot*>−1 **THEN** *vote*[*a*][*i*][*maxBallot*] **ELSE** *Nil*
IN ⟨⟨*maxBallot*,*v*⟩⟩

*Phase*1*b*(*a*,*b*)==
/*ballot*[*a*]<*b*
/\⟨⟨*b*⟩⟩\in 1*amsgs*
/*ballot*′=[*ballot* **EXCEPT** ![*a*]=*b*]
/\1*bmsgs*′=1*bmsgs*\cup
{⟨⟨*b*,{⟨⟨*i*,*MaxAcceptorVote*(*a*,*i*)⟩⟩:*i*\in *Instances*},*a*⟩⟩}
/**UNCHANGED** ⟨⟨*vote*,*leaderVote*,1*amsgs*,2*amsgs*⟩⟩

IncreaseBallot(*a*,*b*)==
/*ballot*[*a*]<*b*
/*ballot*′=[*ballot* **EXCEPT** ![*a*]=*b*]
/**UNCHANGED** ⟨⟨*vote*,*leaderVote*,1*amsgs*,1*bmsgs*,2*amsgs*⟩⟩

- *Merge*(*b*):与 *Propose*(*b*,*i*),*Phase2a*(*b*,*i*)动作一起对应于 Paxos 的 *Phase2a* 阶段.在 *Merge* 动作中,提议者根据收到的投票回复更新自己的日志,但是不发起 Accept 请求.当发起提议编号为 *b* 的 Prepare 请求的提议者收到一个多数派的回复时,对每一个实例编号 *i*,根据回复(方法如 Paxos 协议的 Phase2a 阶段描述)选取提议值,并保存在 *leaderVote*[*b*][*i*]中.值得注意的是:只有当 *leaderVote*[*b*][*i*]之前没有被修改过时(*leaderVote*[*b*][*i*]=⟨−1,*Nil*⟩)时才能更新.因为 Multi-Paxos 中每个提议编号只能有一个提议者发起投票,且对于每个实例编号,它只能提出一个提议值;
- *Propose*(*b*,*i*):提议者对实例编号 *i* 提出提议值.若 *leaderVote*[*b*][*i*]=⟨−1,*Nil*⟩,那么提议者选取一个合法的提议值 *v*,更新 *leaderVote*[*b*][*i*]=⟨*b*,*v*⟩,并发起请求;否则,提议者直接将 *leaderVote*[*b*][*i*]发送给接受者;
- *Phase2a*(*b*,*i*):提议者根据 *Merge*(*b*)和 *Propose*(*b*,*i*)中对实例编号 *i* 确定的提议值,发起 Accept 请求;
- *Vote*(*a*,*b*,*i*):对应于 Paxos 的 *Phase2b* 阶段.接受者 *a* 收到提议编号为 *b*、实例编号为 *i* 的 Accept 请求时,若 *b* 不小于 *a* 的提议编号(*b*>*ballot*[*a*]),则 *a* 接受 Accept 请求,并更新 *vote*[*a*][*i*][*b*]为对应的提议值.

1*bMsgs*(*b*,*Q*)=={*m*\in 1*bmsgs*:*m*[3]\in *Q*/*m*[1]=*b*}

MaxVote(*b*,*i*,*Q*)==
LET *entries*==**UNION** {*m*[2]:*m*\in 1*bMsgs*(*b*,*Q*)}
ientries=={*e*\in *entries*:*e*[1]=*i*}
maxBal==Max({*e*[2][1]:*e*\in *ientries*})
IN CHOOSE *v*\in *Value*\cup {*Nil*}:*E e*\in *ientries*:
/*e*[2][1]=*maxBal*/*e*[2][2]=*v*

lastInstance(*b*,*Q*)==**LET** *entries*==**UNION** {*m*[2]:*m*\in 1*bMsgs*(*b*,*Q*)}

```
                    valid=={e\in entries:e[2][1]/=−1}
                IN IF valid={·} THEN −1 ELSE Max({e[1]:e\in valid})
Merge(b)==/\\E Q\in Quorums:
            /\\A a\in Q:\E m\in 1bMsgs(b,Q):m[3]=a
            /\leaderVote'=[leaderVote EXCEPT ![b]=[i\in Instances↦
              IF (/\i\in 0..lastInstance(b,Q)
                /\leaderVote[b][i][1]=−1)
              THEN ⟨⟨b,MaxVote(b,i,Q)⟩⟩
              ELSE leaderVote[b][i]]]
            /\UNCHANGED ⟨⟨vote,ballot,1amsgs,1bmsgs,2amsgs⟩⟩

Propose(b,i)==/\leaderVote[b][i][1]=−1
              /\\E Q\in Quorums:
              /\\A a\in Q:\E m\in 1bMsgs(b,Q):m[3]=a
              /\LET maxV==MaxVote(b,i,Q)
                    safe==IF maxV/=Nil THEN {maxV} ELSE
                                            Value \cup {Nil}
                IN \E v\in safe: leaderVote'=[leaderVote EXCEPT
                                            ![b][i]=⟨⟨b,v⟩⟩]
            /\UNCHANGED ⟨⟨vote,ballot,1amsgs,1bmsgs,2amsgs⟩⟩

Phase2a(b,i)==
  /\leaderVote[b][i][1]=b
  /\2amsgs'=2amsgs\cup {⟨⟨b,i,leaderVote[b][i]⟩⟩}
  /\UNCHANGED ⟨⟨ballot,vote,leaderVote,1amsgs,1bmsgs⟩⟩

Vote(a,b,i)==
  /\ballot[a]\leq b
  /\ballot'=[ballot EXCEPT ![a]=b]
  /\\E m\in 2amsgs:
    /\m[2]=i/\m[1]=b
    /\vote'=[vote EXCEPT ![a][i][b]=m[3][2]]
  /\UNCHANGED ⟨⟨leaderVote,1amsgs,1bmsgs,2amsgs⟩⟩
----------------------------------------------------------------------------------------------------
Next 定义了次态关系,Spec 定义了完整的行为规约.
----------------------------------------------------------------------------------------------------
Next==
  \/\E a\in Acceptors,b\in Ballots:IncreaseBallot(a,b)
  \/\E b\in Ballots:Phase1a(b)
  \/\E a\in Acceptors,b\in Ballots:Phase1b(a,b)
  \/\E b\in Ballots:Merge(b)
  \/\E b\in Ballots,i\in Instances:Propose(b,i)
```

\/\E *b*\in *Ballots*,*i*\in *Instances*:*Phase2a*(*b*,*i*)

\/\E *a*\in *Acceptors*,*b*\in *Ballots*,*i*\in *Instances*:*Vote*(*a*,*b*,*i*)

Spec==*Init*/\[·][*Next*]_⟨⟨*leaderVote*,*ballot*,*vote*,1*amsgs*,1*bmsgs*,2*amsgs*⟩⟩

1.5 Raft协议

Raft 是一种更易于理解的分布式共识协议[10],它加强了日志项之间的串行性,简化了协议的设计.Raft 中的每个节点都维护一个递增的变量,称为任期(term).任期本质上是节点共同维护的逻辑时钟,通过任期,节点可以发现过时的消息.具体而言,节点间发送消息时,需携带自身当前的任期.如果节点收到的消息携带的任期值小于该节点自身当前的任期值,则拒绝该消息;否则,则更新自身的任期值.节点向日志添加新的日志项时,会将自身当前的任期值也保存在日志项中,这成为该日志项的任期.

Raft 中的节点有 3 种角色:主节点(leader)、从节点(follower)和候选节点(candidate).初始状态下,所有的节点都是从节点.Raft 协议主要包括两部分:选举主节点以及主节点向从节点同步日志.正常情况下,主节点通过定时向其他节点发送心跳信号来维护自己的主节点身份.当从节点一段时间内没有收到心跳信号时,便转变为候选节点.候选节点首先增大自身任期,然后将任期发送给所有节点,以发起选举.收到选举请求后,节点将比较自身任期和和选举请求携带的任期.如果自身任期较大,或者自己在本任期内已为其他候选节点投过票,则拒绝此次选举请求.收到多数派投票的候选节点将成为新的主节点.Raft 的多数派投票机制确保选举的安全性(election safety)[10]:每个任期内,最多只有一个主节点.为了保证新任主节点的完全性(leader completeness),即它的日志应包含所有已被提交的日志项,Raft 引入了如下规则:如果候选节点的日志比自身的日志旧[10],那么节点将拒绝该选举请求.节点间可以通过比较日志中编号最大的日志项的编号与任期(分别称为 *lastIndex* 和 *lastTerm*)判断日志的新旧[10].

主节点按照日志编号顺序向从节点同步日志项,从节点需要按照编号顺序接受主节点的日志项.在收到编号更小的日志项之前,从节点不能接受编号更大的日志项.主节点与从节点间通过确认(ack)机制维护下一个可以接受的编号.与 Multi-Paxos 中不同实例独立发送和接受日志项不同,Raft 在所有的日志项之间通过编号建立了全序关系.通过这样的限制,Raft 中节点的日志不会出现空洞,同时保证了节点间日志的一致性(log matching property)[10]:如果两个节点上的日志在同一位置拥有相同的日志项,那么之前所有位置上的日志项也必定相同.根据日志的一致性,如果某个日志项已被提交,那么日志中所有编号更小的日志项也已被提交.

2 ParallelRaft-SE 协议及精化

Raft 要求顺序提交、顺序执行用户命令,这种限制使它不适用于高并发系统[7].因此,阿里巴巴公司基于 Raft 提出了 ParallelRaft,它允许乱序提交、乱序执行用户命令[7].为了理清 ParallelRaft 与 Raft 之间的关系,我们提出了 ParallelRaft-SE.ParallelRaft-SE 允许乱序提交,但仍要求顺序执行用户命令.因此,可以将 ParallelRaft-SE 视为顺序执行版本的 ParallelRaft.为了证明 ParallelRaft-SE 的正确性,我们建立了从它到 Multi-Paxos 的精化关系.

2.1 ParallelRaft-SE协议

ParallelRaft-SE 沿用了 Raft 的角色(主节点、从节点、候选节点)、任期的概念以及选主机制.ParallelRaft-SE 协议主要包括 3 部分:主节点向从节点同步日志、主节点的选取和日志恢复.

在 ParallelRaft-SE 中,主节点可以并发地向从节点发送多个日志项.从节点收到日志项后立即接受并确认,而无需等待编号更小的日志项.因此,ParallelRaft-SE 支持日志的乱序接受和提交.

ParallelRaft-SE 的选举机制与 Raft 基本相同,都需要保证选举的安全性.不同的是:由于 ParallelRaft-SE 不具有 Raft 的串行性,节点的日志中可能有空洞,因此在选举过程中,ParallelRaft-SE 无法通过节点比较日志的新旧保证新任主节点的完全性.因此,ParallelRaft-SE 加入了日志恢复阶段.

ParallelRaft-SE 的日志恢复过程的基本思想是:新任主节点收集其他节点的日志并运行 Paxos 协议,恢复那

些可能已被提交但新任主节点缺失的日志项.

经过恢复,新任主节点满足完全性,此时可以向从节点同步日志项.

ParallelRaft-SE 的规约中使用了如下的常量与变量(仅介绍相对于 Multi-Paxos 模块新引入的部分).

- *Server* 是所有参与共识的节点的集合;
- *Follower*,*Candidate*,*Leader* 是服务器的 3 种不同状态;
- *r*1*amsgs*,*r*1*bmsgs*,*r*2*amsgs*,*r*2*bmsgs*,*r*3*amsgs*,*negMsgs* 是不同类型的消息的集合;
- *currentTerm*[*i*]为节点 *i* 所记录的最大的任期;
- *currentState*[*i*]为节点 *i* 的状态,在任何时候都是 *Follower*,*Candidate*,*Leader* 之一;
- *vote*[*i*][*n*][*t*]表示节点 *i* 在任期 *t* 接受的序号为 *n* 的日志项.引入 *vote* 是为了构建 ParallelRaft-SE 到 Multi-Paxos 的精化,实际协议中不需要 *vote*;
- *leaderLog* 记录了每个任期的主节点的日志.*leader*Log 同样是为了构建 ParallelRaft-SE 到 Multi-Paxos 的精化,实际协议中不需要.*leader*Log[*t*][*n*]表示任期 *t* 的主节点的日志中编号为 *n* 的日志项.每个日志项是形如⟨*t*′,*v*,*b*⟩的三元组.其中,*t*′是日志项的任期;*v* 是提议值;*b* 是一个布尔值,当且仅当日志项被提交时,*b* 为真;
- log[*i*][*n*]为节点 *i* 的日志中编号为 *n* 的日志项.

---------- **MODULE** ParallelRaft-SE ----------

EXTENDS *Integers*,*FiniteSets*,*Sequences*,*TLC*

CONSTANTS *Server*,*Follower*,*Candidate*,*Leader*,*Nil*,*Value*

Quorums=={*i*\in **SUBSET**(*Server*):*Cardinality*(*i*)*2>*Cardinality*(*Server*)}

Index==*Nat*

Term==*Nat*

VARIABLE *r*1*amsgs*,*r*1*bmsgs*,*r*2*amsgs*,*r*2*bmsgs*,*r*3*amsgs*,*negMsgs*,
currentTerm,*currentState*,*vote*,*leaderLog*,*log*

serverVars==⟨⟨*currentTerm*,*currentState*⟩⟩

vars==⟨⟨*r*1*amsgs*,*r*1*bmsgs*,*r*2*amsgs*,*r*2*bmsgs*,*r*3*amsgs*,*negMsgs*,*log*,
serverVars,*leaderLog*,*vote*⟩⟩

主要的动作包括:

- *Timeout*(*i*):从节点或候选节点 *i* 因未收到来自主节点的消息而超时,则增大自身任期(*currentTerm*′[*i*]=*currentTerm*[*i*]+1),将状态转变为选举者(*currentState*′[*i*]=*Candidate*);
- *RequestVote*(*i*):候选节点 *i* 向其他节点发送自身任期,发起选举请求;
- *HandleRequestVote*(*i*):节点 *i* 收到选举请求 *m*,如果 *m* 携带的任期大于 *i* 的任期(*m*[1]>*currentTerm*[*i*]),则 *i* 升级任期,接受该选举请求,并将自己的日志发送给候选节点;否则,*i* 拒绝该选举请求.

Timeout(*i*)==
/*currentState*[*i*]\in {*Follower*,*Candidate*}
/*currentTerm*′=[*currentTerm* **EXCEPT** ![*i*]=*currentTerm*[*i*]+1]
/*currentState*′=[*currentState* **EXCEPT** ![*i*]=*Candidate*]
/**UNCHANGED** ⟨⟨*r*1*amsgs*,*r*1*bmsgs*,*log*,*r*2*amsgs*,*r*2*bmsgs*,*r*3*amsgs*,
negMsgs,*leaderLog*,*vote*⟩⟩

```
RequestVote(i)==
  /\currentState[i]=Candidate
  /\r1amsgs'=r1amsgs\cup {⟨⟨currentTerm[i],i⟩⟩}
  /\UNCHANGED ⟨⟨serverVars,r1bmsgs,log,r2amsgs,r2bmsgs,r3amsgs,
                negMsgs,leaderLog,vote⟩⟩

HandleRequestVoteRequest(i)==
  /\\E m\in r1amsgs:
    LET j==m[2]
        grant==m[1]>currentTerm[i]
        entries=={⟨⟨n,log[i][n]⟩⟩:n\in Index}
    IN
      \/\grant
        /\UpdateTerm(i,m[1])
        /\r1bmsgs'=r1bmsgs\cup {⟨⟨m[1],entries,i,j⟩⟩}
        /\UNCHANGED negMsgs
      \/\\neg grant
        /\negMsgs'=negMsgs\cup {⟨⟨currentTerm[i],j⟩⟩}
        /\UNCHANGED ⟨⟨currentState,currentTerm,r1bmsgs⟩⟩
  /\UNCHANGED ⟨⟨log,r1amsgs,r2amsgs,r2bmsgs,r3amsgs,vote,leaderLog⟩⟩
```

- *BecomeLeader*(*i*):收到多数派投票的候选节点 *i* 成为主节点,并根据收到的日志恢复可能缺失的日志项.对每个日志编号 *n*,考察它收到的所有编号为 *n* 的日志项,选择其中任期最大的日志项,将该日志项的任期修改为 *i* 自身的任期.为了构建从 ParallelRaft-SE 到 Multi-Paxos 的精化关系,恢复过程将修改 *leaderLog*,而非直接修改 *log*[*i*].之后,*i* 可以向自己发送 *RequestSync* 请求,完成日志更新;
- *RequestSync*(*i*):主节点 *i* 向其他节点同步日志项.

```
Merge(entries,term)==
  LET committed=={e\in entries:e[3]=TRUE}
      chosen==
        CASE committed={·}→CHOOSE x\in entries:
                             \A y\in entries:x[1]\geq y[1]
        [·] committed/={·}→CHOOSE x\in committed:TRUE
      safe==chosen[2]
  IN ⟨⟨term,safe,chosen[3]⟩⟩

BecomeLeader(i)==
  /\currentState[i]=Candidate
  /\\E Q\in Quorums:
    LET voteGranted=={m\in r1bmsgs:m[4]=i/\m[3]\in Q
                        /\m[1]=currentTerm[i]}
```

```
        allLog==UNION {m[2]:m\in voteGranted}
        valid=={e\in allLog:e[2][1]/=−1}
        end==IF valid={·} THEN −1 ELSE Max({e[1]:e\in valid})
    IN
    /\\A q\in Q:\E m\in voteGranted:m[3]=q
    /\leaderLog'=[leaderLog EXCEPT ![currentTerm[i]]=
                    [n\in Index↦IF n\in 0..end THEN
                     Merge({l[2]:l\in {t\in allLog:t[1]=n}},currentTerm[i])
                       ELSE ⟨⟨−1,Nil,FALSE⟩⟩]]
  /\currentState'=[currentState EXCEPT ![i]=Leader]
  /\UNCHANGED ⟨⟨currentTerm,r1amsgs,r2amsgs,r1bmsgs,r2bmsgs,r3amsgs,
                    negMsgs,log,vote⟩⟩

RequestSync(i)==
  /\currentState[i]=Leader
  /\LET sync=={n\in Index:leaderLog[currentTerm[i]][n][1]/=−1}
    IN
    \E n\in sync:r2amsgs'=r2amsgs\cup
      {⟨⟨currentTerm[i],n,leaderLog[currentTerm[i]][n],i⟩⟩}
  /\UNCHANGED ⟨⟨serverVars,log,r1amsgs,r1bmsgs,r2bmsgs,
                    r3amsgs,negMsgs,leaderLog,vote⟩⟩
```

- *HandleRequestSyncRequest*(*i*):*i* 收到 *RequestSync* 请求 *m*,如果 *m* 携带的任期不小于 *i* 自身的任期(*m*[1]> *currentTerm*[*i*]),则 *i* 接受该请求,升级自己的任期,更新 *log*[*i*]与 *vote*,并回复确认;
- *CommitEntry*(*i*):收到多数派节点对某个日志项的确认后,主节点 *i* 将该日志项标记为已提交(*Committed*);
- *RequestCommit*(*i*):主节点将已提交的日志项发送给其他节点.

```
HandleRequestSyncRequest(i)==
  /\\E m\in r2amsgs:
      LET j==m[4]
          grant==m[1]\geq currentTerm[i]
      IN
    /\\/\m[1]>currentTerm[i]
        /\UpdateTerm(i,m[1])
      \/\m[1]\leq currentTerm[i]
        /\UNCHANGED ⟨⟨currentTerm,currentState⟩⟩
    /\\/\grant
        /\log'=[log EXCEPT ![i][m[2]]=m[3]]
        /\vote'=[vote EXCEPT ![i][m[2]][m[1]]=m[3][2]]
        /\r2bmsgs'=r2bmsgs\cup {⟨⟨m[1],m[2],i,j⟩⟩}
        /\UNCHANGED negMsgs
```

```
        \/\\neg grant
          /\negMsgs'=negMsgs\cup {⟨⟨currentTerm[i],j⟩⟩}
          /\UNCHANGED ⟨⟨vote,r2bmsgs,log⟩⟩
    /\UNCHANGED ⟨⟨r1amsgs,r1bmsgs,r2amsgs,r3amsgs,leaderLog⟩⟩

CommitEntry(i)==
    /\\E index\in Index,Q\in Quorums:
      LET syncSuccess=={m\in r2bmsgs:
                              m[4]=i/\m[3]\in Q
                              /\m[1]=currentTerm[i]/\m[2]=index}
      IN
      /\currentState[i]=Leader
      /\\A q\in Q:\E m \in syncSuccess:m[3]=q
      /\leaderLog'=[leaderLog EXCEPT ![currentTerm[i]][index][3]=TRUE]
    /\UNCHANGED ⟨⟨serverVars,log,r1amsgs,r1bmsgs,r2amsgs,r2bmsgs,
                    r3amsgs,negMsgs,vote⟩⟩

RequestCommit(i)==
    /\currentState[i]=Leader
    /\LET committed=={n\in Index:leaderLog[currentTerm[i]][n][3]=TRUE} IN
      \E n\in committed:r3amsgs'=r3amsgs\cup {⟨⟨currentTerm[i],n,i⟩⟩}
    /\UNCHANGED ⟨⟨serverVars,log,r1amsgs,r1bmsgs,r2amsgs,r2bmsgs,
                    negMsgs,leaderLog,vote⟩⟩
```

- *HandleRequestCommit*(*i*):节点 *i* 收到来自主节点的 *RequestCommit* 请求 *m*,如果 *m* 携带的任期不小于 *i* 自身的任期,则 *i* 将相应的日志项标记为已提交;
- *ClientRequest*(*i*):收到用户的命令 *v* 后,主节点 *i* 将 *v* 作为新的日志项添加到日志中.

```
HandleRequestCommitRequest(i)==
    /\\E m\in r3amsgs:
      LET grant==currentTerm[i]\leq m[1]
          j==m[3]
      IN
      /\\/\m[1]>currentTerm[i]
          /\UpdateTerm(i,m[1])
        \/\m[1]\leq currentTerm[i]
          /\UNCHANGED ⟨⟨currentTerm,currentState⟩⟩
      /\\/\grant
          /\log[i][m[2]][1]=m[1]
          /\log'=[log EXCEPT ![i][m[2]][3]=TRUE]
          /\UNCHANGED negMsgs
        \/\\neg grant
```

```
        /\negMsgs'=negMsgs\cup {⟨⟨currentTerm[i],j⟩⟩}
        /\UNCHANGED log
  /\UNCHANGED ⟨⟨serverVars,r1amsgs,r1bmsgs,r2amsgs,r2bmsgs,
                r3amsgs,leaderLog,vote⟩⟩

ClientRequest(i)==
  LET ind=={b\in Index:leaderLog[currentTerm[i]][b][1]/=−1}
      nextIndex==IF ind={·}
        THEN 0
        ELSE Max(ind)+1
  IN
  /\currentState[i]=Leader
  /\\E v\in Value:leaderLog'=[leaderLog EXCEPT ![currentTerm[i]][nextIndex]=
                              ⟨⟨currentTerm[i],v,FALSE⟩⟩]
  /\UNCHANGED ⟨⟨serverVars,log,r1amsgs,r1bmsgs,r2amsgs,r2bmsgs,r3amsgs,
                negMsgs,vote⟩⟩
----------------------------------------------------------------------------------------------------------
Next 定义了次态关系.Spec 定义了完整的行为规约.
----------------------------------------------------------------------------
Next==\\E i\in Server:Timeout(i)
      \\E i\in Server:RequestVote(i)
      \\E i\in Server:HandleRequestVoteRequest(i)
      \\E i\in Server:BecomeLeader(i)
      \\E i\in Server:CommitEntry(i)
      \\E i\in Server:ClientRequest(i)
      \\E i,j\in Server:RequestCommit(i)
      \\E i\in Server:HandleRequestCommitRequest(i)
      \\E i,j\in Server:RequestSync(i)
      \\E i\in Server:HandleRequestSyncRequest(i)

Spec==Init/\[·][Next]_vars
----------------------------------------------------------------------------
```

2.2 精化ParallelRaft-SE到Multi-Paxos

ParallelRaft-SE 支持乱序提交、顺序执行用户命令,这与 Multi-Paxos 相同.此外,ParallelRaft-SE 的日志恢复阶段本质上是使用 Paxos 对可能缺失的日志项重确认,而 Multi-Paxos 也通过 Paxos 发起提议.实际上,我们可以建立从 ParallelRaft-SE 到 Multi-Paxos 的精化关系,从而也证明了 ParallelRaft-SE 的正确性.这种精化关系基于 ParallelRaft-SE 与 Multi-Paxos 的以下相似之处.

- *RequestVote* 对应于 *Phase*1*a*.ParallelRaft-SE 中的任期对应于 Multi-Paxos 中的提议编号;
- *HandleRequestVote* 对应于 *Phase*1*b*.二者都需要通过比较任期/提议编号来决定是否同意选举/接受提议请求,且都需要在回复中包含自己的日志;
- *BecomeLeader* 对应于 *Merge*.收到多数派回复的主节点/提议者运行 Paxos 恢复可能缺失的日志项;
- *RequestSync* 对应于 *Phase*2*a*.在 ParallelRaft-SE 中,主节点完成日志恢复后,向从节点同步日志项.在

Multi-Paxos 中,提议者通过运行 Paxos 选出提议值后,向接受者同步;

- *HandleRequestSync* 对应于 *Vote*.在 ParallelRaft-SE 中,从节点收到同步请求后更新自己的日志.在 Multi-Paxos 中,接受者收到提议者的提议后,接受提议并记录在本地;
- *ClientRequest* 对应于 *Propose*.在 ParallelRaft-SE 中,主节点收到用户命令,将其作为日志项添加到日志的末尾.在 Multi-Paxos 中,若对于某个编号,提议者没有收到提议值,可以提出任意合法的值.因此可以向日志末尾追加合法的日志项.

ParallelRaft-SE 规约给出了 ParallelRaft-SE 和 Multi-Paxos 在常量、变量之间的精化映射:

Acceptors==*Server*

Ballots==Term

Instances==*Index*

ballot==*currentTerm*

leaderVote==[*i*\in *Ballots*↦[*j*\in *Index*↦⟨⟨*leaderLog*[*i*][*j*][1],*leaderLog*[*i*][*j*][2]⟩⟩]]

1*amsgs*=={⟨⟨*m*[1]⟩⟩:*m*\in *r*1*amsgs*}

1*bmsgs*=={⟨⟨*m*[1],{⟨⟨*e*[1],⟨⟨*e*[2][1],*e*[2][2]⟩⟩⟩⟩:*e*\in *m*[2]},*m*[3]⟩⟩:*m*\in *r*1*bmsgs*}

2*amsgs*=={⟨⟨*m*[1],*m*[2],⟨⟨*m*[3][1],*m*[3][2]⟩⟩⟩⟩:*m*\in *r*2*amsgs*}

Spec==*Init*/\[·][*Next*]_*vars*

MP==**INSTANCE** *MultiPaxos*

THEOREM *Refinement*==*Spec*⇒*MP*!*Spec*

---

3 乱序执行模型与“幽灵日志”问题

相对于 Raft 而言,ParallelRaft-SE 支持乱序提交.但是,它仍然要求顺序执行用户命令,不适合于高并发系统.在 PolarFS 文件系统应用场景中,ParallelRaft 需要支持乱序执行,也就是允许状态机先执行编号较大的已提交日志项,而不必等待编号较小的日志项被提交或被执行[7].为了在乱序执行情况下仍能满足状态一致性,需要保证被乱序执行的命令是无冲突的.因此,本节先介绍 ParallelRaft 所采用的乱序执行模型.在分析 ParallelRaft 协议的正确性时我们发现,文献[7]中关于 ParallelRaft 的描述忽略了可能会违反状态一致性的“幽灵日志”问题.本节将分析该问题给乱序执行机制带来的挑战.

3.1 乱序执行的模型

ParallelRaft 的乱序执行模型给出了用户命令之间的冲突判断规则[7].在 PolarFS 文件系统应用场景中,每个用户命令都包含该命令所访问数据的逻辑区块地址(logic block address,简称 LBA).LBA 不重叠的命令不存在冲突,可以乱序执行;相反,LBA 有重叠的命令需要按日志编号顺序执行.

在该模型下,每个命令在被执行前,首先检查该命令是否与编号更小的命令存在冲突.由于日志项是乱序接受(可能尚未被提交)的,日志中可能存在“空洞”.为了在存在空洞的情况下仍能判断当前命令是否可被执行,ParallelRaft 要求每个日志项记录它之前的 K(是待定参数)个日志项的 LBA,称为“向后看缓冲区”(look behind buffer,简称 LBF).因此,只要日志中不存在长度超过 K 的“空洞”,就可以判断任意两个日志项/命令之间是否存在冲突;否则,“空洞”之后的日志项都需要等待.

3.2 “幽灵日志”问题

经过日志恢复后,ParallelRaft 中主节点的日志不存在“空洞”,因此它可以为每个日志项计算 LBF.之后,主节点可以将 LBF 随日志项一起发送给从节点.正确的冲突判断要求每个日志项的 LBF 都准确地记录了它之前 K 个日志项的 LBA.然而,“幽灵日志”现象可能会导致主节点计算出的 LBF 与实际不符.图 2 描述了“幽灵日志”现象,它包含 3 个阶段(图中每个方格表示一个日志项,它记录了该日志项的任期以及所携带的用户命令.日志项从 1 开始编号).

(1) 如图 2(a)所示,第 1 阶段中,s_1 是主节点.s_1 向日志中添加了编号为 1~4 的 4 个日志项.其中,编号为 1 和 2 的日志项被 s_2 和 s_3 接受,这两个日志项被提交;编号为 3 和 4 的日志项还没有被 s_2 和 s_3 接受.s_1 失效;

(2) 第 2 阶段中,s_3 成为主节点.s_3 的主节点在恢复过程中没有收到 s_1 未提交的项(注意,s_3 只需要从多数派节点收集日志项).因此完成恢复完成后,s_3 的日志中不包含 s_1 未提交的日志项.如图 2(a),之后,s_3 向日志中添加了编号为 3~6 的新的日志项,并将编号为 6 的日志项发送给了 s_2,该项被提交.由于这一项是对 y 的修改(y←2),与之前的日志项无冲突,因此 s_3 执行 y←2(乱序执行).之后,s_3 也失效,系统选出新的主节点 s_2;

(3) 如图 2(b)所示,第 3 阶段中,s_3 在恢复过程中的收到了 s_1(此时 s_1 恢复正常)未提交的日志项(编号为 4),并将它发送给 s_3.这一项被提交后,s_3 需要执行.它也是对变量 y 的修改(y←3),根据乱序执行的规则,应该先执行.但是 s_3 已经执行 y←2,因此导致了执行顺序的不一致.

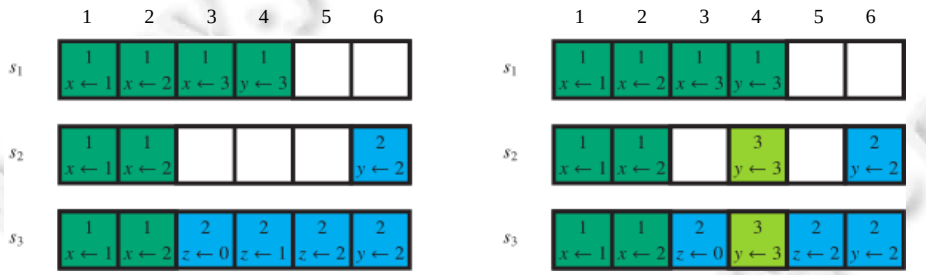

Fig.2 The “Ghost Log Entries” phenonmenon violates consistency

图 2 “幽灵日志”破坏状态一致性

s_1 在第 1 阶段未提交的日志项不在第 2 阶段的主节点 s_3 的日志中,但是之后又出现在了第 3 阶段的主节点 s_2 的日志中.我们称这种现象为“幽灵日志”.这使得 s_3 计算出的 LBF 与实际不符,可能会导致错误的冲突判断,进而影响 ParallelRaft-SE(ParallelRaft)在乱序执行下的正确性.避免出现“幽灵日志”的关键在于,在日志恢复的过程中,第 2 阶段的主节点 s_3 需要明确 s_1 中未提交的日志项的状态:要么将它们重新提交,要么将它们删除并保证之后不会被新的主节点提交.“幽灵日志”现象在顺序执行(如 Multi-Paxos 与 Raft)下也可能出现,但不会影响协议的正确性[22].在顺序执行下,“幽灵日志”问题很容易解决.接下来我们考虑将 Multi-Paxos 的解决方案应用到 ParallelRaft-SE 上,并简要论述 Raft 对“幽灵日志”的解决方法.我们将说明这些方案无法解决乱序执行下的“幽灵日志”问题.

3.3 Multi-Paxos的解决方案

Multi-Paxos 采用了被动式的解决方案.提议者在添加日志项时保存它的生成时间(即生成该日志项的提议者的提议编号).节点可以通过日志项的生成时间检测出“幽灵日志”,并忽略它们.为此,新的提议者完成恢复过程后,向日志中写入一条特殊的空操作日志项,称为栅栏(barrier).只有等栅栏日志项被提交后,提议者才能向日志中添加新的日志项.因此,栅栏日志项标记了日志恢复的终点以及新的提议者添加日志项的起点.如果之后某个日志项的生成时间小于栅栏日志项的生成时间,则可断定它为“幽灵日志”.

如图 3(a)所示,每个日志项中的二元组分别记录了该日志项的提议编号(后续过程中可能更改)与生成时间.s_1 是提议编号 1 的提议者,它向日志中添加了日志编号为 1~4 的日志项,它们的提议编号与生成时间均为 1.其中,日志编号为 3 和 4 的日志项没有被提交.之后,s_1 失效,s_3 成为提议编号 2 的提议者.s_3 收到 s_2 的日志项,完成恢复过程后,向日志中写入一条栅栏日志项(日志编号为 3,在图中用 B 标记),其提议编号与生成时间均为 2.s_3 首先将栅栏日志项发送给 s_2,并将它提交.之后,s_3 响应用户请求,并向日志中添加了日志编号为 4~6 的日志项.其中,日志编号为 6 的日志项被 s_2 接受(因此被提交).s_3 执行该日志项后失效.

s_2 成为提议编号 3 的提议者.如图 3(b)所示:在恢复过程中,s_2 从 s_1 的日志中找到了日志编号为 4(y←3)的日志项并将它添加到日志中.这一项的提议编号为 3,但是它是 s_1 写入日志的,生成时间为 1.而 s_2 的日志中编号为 3 的栅栏日志项的生成时间为 2.由此,s_2 可以判断日志编号为 4 的日志项是“幽灵日志”.s_2 从日志中删除该项,从而解决了“幽灵日志”问题.

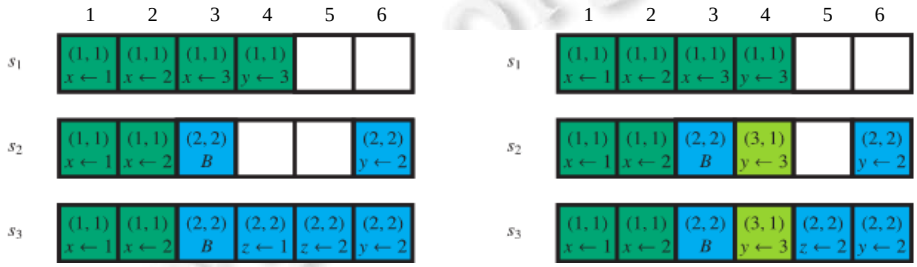

Fig.3 Passive way to address the “Ghost Log Entries” phenomenon

图 3 “幽灵日志”被动式解决方案

3.4 Raft的解决方案

Raft 采用了主动式的解决方案[22]:利用选主机制中的限制条件,使得拥有“幽灵日志”的节点无法成为新的主节点.为此,Raft 中选出的新主节点向日志中添加一条栅栏日志项.只有等栅栏日志项被提交后,新的主节点才能响应用户的请求.根据 Raft 的选举规则,此时,没有栅栏日志项的节点将无法成为主节点(日志较旧,无法获得多数派投票).另一方面,在日志项同步阶段,接收到栅栏日志项的节点将会删除栅栏日志项后的所有日志项,故不存在“幽灵日志”.

如图 4(a)所示,s_1 是任期 1 的主节点,它向日志中写入了编号为 1~4 的 4 个日志项.其中,编号为 3 和 4 的日志项还没有提交,主节点变更为 s_2.如图 4(b)所示,s_2 成为任期 2 的主节点后,先向日志中写入一条栅栏日志项并提交,之后 s_2 才能响应用户请求.即使 s_2 之后失效,根据 Raft 的选主规则,s_1 若想成为主节点,需要先删除多余的日志项(编号为 3 和 4),并添加栅栏日志项.因此,编号为 3 和 4 的两个日志项不会出现在新的主节点的日志中.

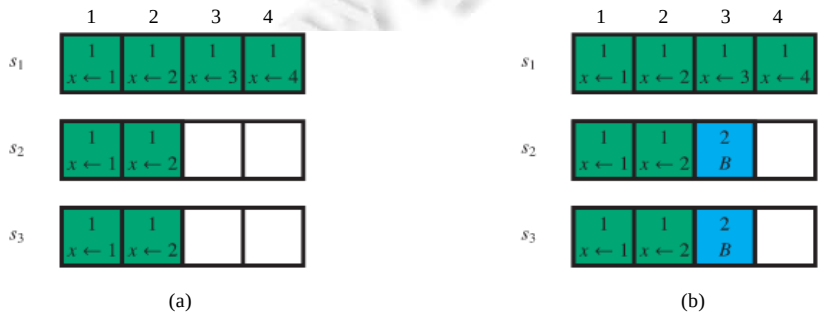

Fig.4 Proactive way to address the “Ghost Log Entries” phenomenon

图 4 “幽灵日志”主动式解决方案

3.5 乱序执行下的挑战

将 Multi-Paoxs 对“幽灵日志”的解决方案直接应用在 ParallelRaft-SE 上,并不能保证 ParallelRaft-SE(或 ParallelRaft)在乱序执行下的正确性.换句话说,在乱序执行下,“幽灵日志”现象并非导致状态不一致的必要条件.比如,主节点 *i* 的日志中包含未提交的日志项.当主节点发生变更后,*i* 乱序地收到了新主节点的日志项.*i* 并不知道日志中未提交的日志项已经过时,从而误以为不存在冲突,导致不一致的行为.

另一方面,Raft 利用它的串行性在选举过程中消除了“幽灵日志”.然而,ParallelRaft-SE 支持乱序提交,日志中可能存在“空洞”.这使得 ParallelRaft-SE 也不能直接应用 Raft 的解决方案.

4 ParallelRaft-CE 协议与规约

为了解决乱序执行下的“幽灵日志”问题,我们基于 ParallelRaft-SE 提出了 ParallelRaft-CE 协议.ParallelRaft-CE 通过限制 ParallelRaft-SE 在乱序提交阶段的并行度,避免了“幽灵日志”问题,保证乱序执行下的状态机的一致性.ParallelRaft-CE 主要包括 3 个部分:日志同步机制、选主机制及其日志恢复机制.

我们先给出 ParallelRaft-CE 的几条关键性质,其中,“同步号”与“主节点候选者”的概念将在接下来的协议描述中介绍.下一节将简要证明这些关键性质以及 ParallelRaft-CE 的正确性.

(1) 节点的任期与同步号单调递增;
(2) 选举的安全性:每个任期最多选出一个主节点;
(3) 设主节点候选者的任期为 *t*,同步号为 *s*,则 *s*<*t* 且系统内不存在任期大于 *s* 的日志项;
(4) 主节点完全性:若主节点的任期为 *t*,那么它的日志中包括所有任期小于 *t* 且被提交的日志项;
(5) 一致性:对任意两个节点,如果它们的同步号都大于某个任期值 *t*,则它们的日志中包含相同的任期为 *t* 的日志项.

4.1 日志同步机制

在 ParallelRaft-CE 中,日志项的发送与接受是部分乱序的:任期相同的日志项可以并发地发送与接受,任期不同的日志项则需按序发送与接受.为此,每个节点维护一个同步号(sync),表示目前节点仅接受任期等于同步号的日志项.当从节点收到主节点的日志项时,会检查该日志项的任期.如果日志项的任期与从节点的同步号相同,则从节点接受该日志项,并回复确认消息;否则,从节点拒绝该日志项,并将自己的同步号发送给主节点.主节点额外维护每个从节点的同步号.当收到从节点的同步号时,主节点升级自己记录的从节点同步号.当从节点对当前任期的每一条日志项都完成确认后,主节点通知从节点将同步号设置为下一任期.这种日志同步机制限制了提交阶段的并行度.

4.2 选主机制

ParallelRaft-CE 的选主机制与 Raft 相似.不同的是,ParallelRaft-CE 通过同步号判断日志的新旧:节点的同步号越大,它的日志就越新.候选节点发起选举时,将自己的同步号也发送给其他节点.只有当接收节点的同步号不大于候选节点的同步号时,才同意该选举请求.

4.3 日志恢复机制

在 ParallelRaft-CE 中,任期相同的日志项可以并发发送与接受.因此,日志中可能存在“空洞”,新的主节点需要对上一任期的日志项进行恢复.恢复的目标是使得主节点的同步号升级为它的任期,此时,系统便已对所有任期小于主节点任期的日志项达成了共识.我们将尚未完成日志恢复的主节点称为主节点候选者(leader candidate).恢复过程完成后,所有任期小于主节点任期的日志项的状态是确定的:已经被提交(或执行)的日志项仍在新主节点的日志中;之前未提交的日志项要么被提交,要么被丢弃且之后不会再被提交.

假设主节点候选者 *l* 的任期为 *t*,同步号为 *s*(根据性质 3,*s*<*t*).根据性质 5 系统已对任期小于 *s* 的日志项达成共识.再根据性质 3,系统中不存在任期超过 *s* 的日志项.因此,*l* 仅需要恢复任期等于 *s* 的日志项.恢复日志项的方

法与 ParallelRaft-SE(或 Multi-Paxos)相同,*l* 需要从多数派节点收集日志项,并从中选择最“新”的日志项.由于日志项的任期是固定的,ParallelRaft-CE 为每个日志项增加一个新的变量:日期(date).日期相当于 Paxos 中日志项的提议编号.日志编号相同,日期越大的日志项就越“新”.

l 需要分 3 个阶段将恢复得到的日志项同步给从节点.设 *l* 记录某从节点的同步号为 *n*.如果 *n*>*s*,则直接进入第 3 阶段;如果 *n*=*s*,则直接进入第 2 阶段;否则,先进入第 1 阶段.

第 1 阶段(*n*<*s*):*l* 先向该从节点发送任期为 *n* 的日志项.当这些项被从节点全部确认后,*l* 通知从节点升级到满足以下条件的下一个同步号 *k*:

(1) *k*>*n*;

(2) *l* 的日志中含有任期为 *k* 的日志项;

(3) *k* 是满足条件 1、条件 2 的最小值.

收到升级同步号请求后,从节点删除日志中所有未确认的任期为 *n* 的日志项,并将同步号升级为 *k*.之后,主节点候选者向从节点发送任期为 *k* 的日志项.持续以上过程,直到从节点的同步号升级为 *s*.进入第 2 阶段.

第 2 阶段(*n*=*s*):*l* 持续向从节点同步日志项,直到任期为 *s* 的所有日志项都被相同的多数派提交,记这个多数派为 *Q*.进入第 3 阶段.

第 3 阶段(*n*>*s*):*l* 将它的同步号升级为自身的任期 *t*(根据性质 3,这是安全的).然后,*l* 通知 *Q* 中的节点将自身同步号升级为 *t*.收到某多数派节点(均来自 *Q*)的确认消息后,恢复过程结束,*l* 正式成为主节点.

4.4 ParallelRaft-CE的TLA+规约

ParallelRaft-CE 使用常量 *LeaderCandidate* 表示主节点候选者角色.其中的变量包括:

- *messages*:节点发送的消息的集合;
- *currentTerm*[*i*]:节点 *i* 记录的最大的任期;
- *currentState*[*i*]:节点 *i* 的状态,为 *Leader*,*LeaderCandidate*,*Follower*,*Candidate* 之一;
- *votedFor*:每个节点一个任期内只能为一个候选节点投票.*votedFor*[*i*]表示 *i* 在本任期(*currentTerm*[*i*])内投票的候选节点.若 *i* 没有为任何节点投票,则 *votedFor*[*i*]=*Nil*;
- *sync*[*i*]:节点 *i* 的同步号;
- *end*[*i*][*t*]:节点 *i* 记录的可接受的任期为 *t* 的日志项的最大编号.ParallelRaft-CE 使用 Paxos 的投票机制确定每一个任期的最大编号;
- log[*i*]:节点 *i* 的日志.*log*[*i*][*k*]为 *i* 的日志中编号为 *k* 的日志项.每个日志项是⟨*t*,*d*,*v*,*b*⟩的四元组.其中:*t* 是日志项的任期,不可修改;*d* 是日期,用于判断日志项的新旧;*v* 是提议值;*b* 是布尔值,当且仅当日志项被提交时,*b* 为真;
- *syncTrack*:状态为 *Leader* 或 *LeaderCandidate* 的节点用来记录其他节点的同步号.*syncTrack*[*i*][*a*]为节点 *i* 记录的节点 *a* 的同步号;
- *elections*,*halfElections* 为历史变量,记录了选举信息.

```
---------------------------- MODULE ParallelRaft-CE ----------------------------
CONSTANTS Server,Follower,Candidate,Leader,LeaderCandidate,Nil,Value,
          RequestVoteRequest,RequestVoteResponse,
          RequestCommitRequest,RequestCommitResponse,
          RequestSyncRequest,RequestSyncResponse,
          UpdateSyncRequest,UpdateSyncResponse

Quorums=={i\in SUBSET (Server):Cardinality(i)*2>Cardinality(Server)}

VARIABLE messages,currentTerm,currentState,votedFor,sync,end,log,syncTrack
```

serverVars==⟨⟨*currentTerm*,*currentState*,*votedFor*,*sync*,*end*⟩⟩

VARIABLE *halfElections*,*elections*
electionVars==⟨⟨*halfElections*,*elections*⟩⟩

vars==⟨⟨*messages*,*log*,*serverVars*,*syncTrack*,*electionVars*⟩⟩

ParallelRaft-CE 的主要动作包括:

- *UpdateTerm*(*i*):*s* 收到消息 *m*,若 *m.mterm*>*currentTerm*[*i*],那么 *i* 升级任期(*currentTerm′*[*i*]=*m.mterm*),转变为从节点(*currentState′*[*i*]=*Follower*),并将投票的记录置空(*votedFor′*[*i*]=*Nil*),因为 *i* 在新的任期还没有为任何节点投票.在 ParallelRaft-CE 中,节点收到任何请求都需要检查任期.当消息的任期大于节点的任期时,节点执行 *UpdateTerm*,之后再响应请求;当消息的任期与节点任期相同时,节点直接处理请求;当消息的任期小于节点任期时,节点拒绝任何请求,并将自己的任期回复给发送者;
- *RequestVote*(*i*):*i* 发起选举.与 ParallelRaft-SE(Multi-Paxos)不同的是,*i* 需要将自己的同步号(*sync*[*i*])发送给其他节点;
- *HandleRequestVoteRequest*(*i*):收到选举请求 *m* 的节点 *i* 比较任期(*currentTerm*[*i*])和 *m.mterm*)与同步号(*sync*[*i*]和 *m.msync*).当 *m.mterm*=*currentTerm*[*i*]且 *sync*[*i*]≥*m.msync* 且 *i* 在本任期内还没有同意其他节点的选举请求,*i* 同意选举请求,并将自己的日志中任期为 *m.msync* 的项以及 *i* 可以接受的任期为 *m.msync* 的日志项的最大编号(*end*[*i*][*m.msync*])发送给选举发起者.

UpdateTerm(*i*)==
  /*E m*\in *messages*:
    /*m.mterm*>*currentTerm*[*i*]
    /\\/*m.mdest*=*i*
      \/*m.mdest*=*Nil*
    /*currentTerm′*=[*currentTerm* **EXCEPT** ![*i*]=*m.mterm*]
    /*currentState′*=[*currentState* **EXCEPT** ![*i*]=*Follower*]
    /*votedFor′*=[*votedFor* **EXCEPT** ![*i*]=*Nil*]
  /**UNCHANGED** ⟨⟨*messages*,*sync*,*log*,*syncTrack*,*electionVars*,*end*⟩⟩

RequestVote(*i*)==
  /*currentState*[*i*]=*Candidate*
  /*Send*([*mtype*↦*RequestVoteRequest*,
        mterm↦*currentTerm*[*i*],
        msync↦*sync*[*i*],
        msource↦*i*,
        mdest↦*Nil*])
  /**UNCHANGED** ⟨⟨*serverVars*,*syncTrack*,*log*,*electionVars*⟩⟩

HandleRequestVoteRequest(*i*)==
  /*E m*\in *messages*:

```
    LET j==m.msource
        syncOK==/\m.msync\geq sync[i]
        grant==/\syncOK
                /\votedFor[i]\in {Nil,j}
                /\currentTerm[i]=m.mterm
    IN
      /\m.mterm\leq currentTerm[i]
      /\\/grant/\votedFor'=[votedFor EXCEPT ![i]=j]
        \/\neg grant/\UNCHANGED votedFor
      /\Send([mtype↦RequestVoteResponse,
              mterm↦currentTerm[i],
              mvoteGranted↦grant,
              mlog↦LET C=={n\in Index:log[i][n].term=sync[i]}
                   IN {⟨⟨n,log[i][n]⟩⟩:n\in C},
              mend↦end[i][m.msync],
              msource↦i,
              mdest↦j])
  /\UNCHANGED ⟨⟨currentTerm,currentState,sync,log,syncTrack,
               electionVars,end⟩⟩
```

- *BecomeLeaderCandidate*(*i*):若候选节点 *i* 从一个多数派(*voteGranted*)的节点收到同意选举的消息,*i* 转变为主节点候选者.*i* 要恢复任期等于自己同步号(*sync*[*i*])的日志项,因此,*i* 首先确定能接受的任期为 *sync*[*i*]的日志项的最大编号,将它保存在 *end*[*i*][*sync*[*i*]]中.方法为:在收到的消息的 *m.mend* 中选择最新的.之后对每个可接受的编号,*i* 在收到的对应编号的日志项中选择最新的,并写入日志.对每个节点 *p*,*i* 初始化 *syncTrack*[*i*][*p*]=*sync*[*i*].

```
Merge(entries,term,date)==
    IF entries={·} THEN [term↦term,
                         date↦date,
                         value↦Nil,
                         committed↦FALSE]
    ELSE
      LET
        committed=={e\in entries:e.committed=TRUE}
        chosen==
          CASE committed={·}→CHOOSE x\in entries:
                                \A y\in entries:x.date\geq y.date
          [·] committed/={·}→CHOOSE x\in committed:TRUE
      IN
        [term↦chosen.term,
```

```
                    date↦date,
                    value↦chosen.value,
                    committed↦chosen.committed]

BecomeLeaderCandidate(i)==
  /\currentState[i]=Candidate
  /\\E P\in Quorums:
    LET voteGranted=={m\in messages:/\m.mtype=RequestVoteResponse
                                    /\m.mdest=i
                                    /\m.msource\in P
                                    /\m.mterm=currentTerm[i]
                                    /\m.mvoteGranted=TRUE}
      allLog==UNION {m.mlog:m\in voteGranted}
      endLine==LET allPoint=={m.mend:m\in voteResponded}
                  IN e==CHOOSE e1\in allPoint:
                                  (\A e2\in allPoint:e1[1]\geq e2[1])
      toRecover=={n\in 0..endLine:log[i][n].committed=FALSE}
      toSync=={⟨⟨n,Merge({l[2]:l\in {t\in allLog:t[1]=n}},sync[i],currentTerm[i])⟩⟩
              :n\in toRecover}
    IN
      /\\A p\in P:\E m\in voteGranted:m.msource=p
      /\log'=[log EXCEPT ![i]=[n\in Index↦IF n\in toRecover THEN
                                            (CHOOSE e\in toSync:e[1]=n)[2]
                                          ELSE log[i][n]]]
      /\end'=[end EXCEPT ![i][sync[i]]=⟨⟨currentTerm[i],end⟩⟩]
  /\currentState'=[currentState EXCEPT ![i]=LeaderCandidate]
  /\syncTrack'=[syncTrack EXCEPT ![i]=[j\in Server↦sync[i]]]
  /\UNCHANGED ⟨⟨messages,currentTerm,votedFor,sync,elections⟩⟩
```

- *RequestSync*(*i*):当 *i* 为主节点或主节点候选者时,对其他每个节点 *p*,*i* 向 *p* 发送任期为 *syncTrack*[*i*][*p*]的日志项.这些日志项可以乱序地发送和接受;
- *HandleRequestSyncRequest*(*i*):节点 *i* 收到同步请求 *m*.当 *m.msync*<*sync*[*i*]或 *m.msync*>*sync*[*i*]时,*i* 拒绝请求并回复 *RequestSyncResponse* 消息,将 *sync*[*i*]告知发送者.当 *m.msync*=*sync*[*i*]时,*i* 同意请求,回复确认,并根据 *m.mend*(可以接受的任期为 *m.msync* 的日志项的最大编号)和 *m.mentries*(任期为 *m.msync* 的日志项)修改 *end*[*i*]和 *log*[*i*].对日志的修改为:删除日志中所有编号大于 *m.mend* 的日志项,并将编号不超过 *m.mend* 的日志项替换为 *m.mentries* 中相应的项.

```
RequestSync(i)==
  /\currentState[i]\in {LeaderCandidate,Leader}
  /\\E s\in 0..sync[i]:
    LET start==Min({n\in Index:log[i][n].term=s})
        end==Max({n\in Index:log[i][n].term=s})
```

```
    IN
      /\Send([mtype↦RequestSyncRequest,
              mterm↦currentTerm[i],msync↦s,
              mstart↦start,mend↦end,
              mentries↦IF start=−1 THEN Nil ELSE
                                    [n\in start..end↦log[i][n]],
              msource↦i,mdest↦Nil])
  /\ UNCHANGED ⟨⟨serverVars,logVars,electionVars,syncTrack⟩⟩

HandleRequestSyncRequest(i)==
  /\\E m\in messages:
      LET j==m.msource
          grant==/\m.mterm=currentTerm[i]
                  /\m.msync=sync[i]
      IN
    /\m.mtype=RequestSyncRequest
    /\m.mterm\leq currentTerm[i]
    /\j/=i
    /\\/\grant
      /\log'=[log EXCEPT ![i]=[n\in Index↦
        IF n<m.mstart THEN log[i][n]
        ELSE IF n\in m.mstart..m.mend
          THEN m.mentries[n]
        ELSE [term↦−1,date↦−1,
              value↦Nil,committed↦FALSE]]]
    /\endPoint'=[endPoint EXCEPT ![i][sync[i]]=⟨⟨currentTerm[i],m.mend⟩⟩]
      \/\\neg grant
        /\UNCHANGED ⟨⟨log,endPoint⟩⟩
    /\Send([mtype↦RequestSyncResponse,mterm↦currentTerm[i],
            msyncGranted↦grant,msync↦sync[i],
            mstart↦m.mstart,mend↦m.mend,
            msource↦i,mdest↦j])
  /\UNCHANGED ⟨⟨currentTerm,currentState,sync,votedFor
```

- *HandleRequestSyncResponse*(*i*):*i* 为主节点或主节点候选者时,收到节点 *p* 的 *RequestSyncResponse* 消息 *m*.若 *m* 是确认消息(*m.msyncGranted*=TRUE),且 *p* 的同步号小于 *i* 的同步号(*m.msync*<*sync*[*i*]),则 *s* 向 *p* 发送 *UpdateSyncRequest* 请求,通知 *p* 升级到下一个同步号.下一个同步号的选举方法如上文所述.若 *m* 是否定消息,则 *i* 修改 *syncTrack*[*i*][*p*](*syncTrack'*[*i*][*p*]=*m.msync*);
- *UpdateSync*(*i*):状态为主节点候选者的节点 *i*,如果存在一个多数派的节点完成对任期为 *sync*[*i*]的日志项的同步,且 *i* 收到它们的确认(*RequestSyncResponse* 消息),那么 *s* 向这些节点发送 *UpdateSyncRequest* 消息,通知它们升级同步号为 *i* 的任期(*currentTerm*[*i*]).

```
---------------------------------------------------------------------------------
HandleRequestSyncResponse(i)==
  /\\E m\in messages:
    LET j==m.msource IN
    /\m.mtype=RequestSyncResponse
    /\m.mdest=i
    /\currentTerm[i]=m.mterm
    /\currentState[i]\in {Leader,LeaderCandidate}
    /\syncTrack'=[syncTrack EXCEPT ![i][j]=m.msync]
    /\\/\m.msyncGranted
      /\m.msync<sync[i]
      /\Send([mtype↦UpdateSyncRequest,
              mterm↦currentTerm[i],
              msync↦Min({sync[i]} \union {k\in Nat:k>m.msync/\
                      Cardinality({n\in Index:log[i][n].term=k})>0}),
              msource↦i,
              mdest↦{j}])
    \//\\neg m.msyncGranted
      /\UNCHANGED messages
  /\UNCHANGED ⟨⟨serverVars,log,electionVars⟩⟩

UpdateSync(i)==
  /\currentState[i]=LeaderCandidate
  /\\E Q\in Quorums:
    LET syncUpdated=={m\in messages:/\m.mtype=RequestSyncResponse
                                    /\m.mterm=currentTerm[i]
                                    /\m.msyncGranted=TRUE
                                    /\m.msync=sync[i]
                                    /\m.msource\in Q
                                    /\m.mdest=i}
    IN
      /\\A q\in Q:(\E m\in syncUpdated:m.msource=q)\/q=i
      /\Send([mtype↦UpdateSyncRequest,
              mterm↦currentTerm[i],
              msync↦currentTerm[i],
              msource↦i,
              mdest↦Q])
  /\UNCHANGED ⟨⟨serverVars,log,syncTrack,electionVars⟩⟩
---------------------------------------------------------------------------------------------------------
```

- *HandleUpdateSyncRequest*(*i*):*i* 收到 *UpdateSyncRequest* 请求 *m*,升级自己的同步号(*sync*′[*i*]=*m.msync*),并将日志中的所有日志项标记为提交.然后回复确认,将升级后的同步号通知发送者.收到 *i* 的确认的

主节点(主节点候选者)根据回复修改对 *i* 的同步号的记录(*syncTrack*);

- *HandleUpdateSyncResponse*(*i*):主节点或主节点候选者 *i* 收到 *UpdateSyncResponse* 回复后,更新对发送者同步号的记录.

```
----------------------------------------------------------------------------
HandleUpdateSyncRequest(i)==
  \E m\in messages:
    LET grant==/\currentTerm[i]=m.mterm
                    /\m.msync>sync[i]
        j==m.msource
    IN
    /\m.mtype=UpdateSyncRequest
    /\i \in m.mdest
    /\m.mterm\leq currentTerm[i]
    /\\/\grant
      /\sync'=[sync EXCEPT ![i]=m.msync]
      /\log'=[log EXCEPT ![i]=[n\in Index↦
                                IF log[i][n].term=sync[i] THEN
                                  log[i][n].committed=TRUE
                                ELSE log[i][n]]]
      \/\\neg grant
        /\UNCHANGED ⟨⟨log,sync⟩⟩
    /\Send([mtype↦UpdateSyncResponse,mterm↦currentTerm[i],
            mupdateSyncGranted↦grant,msync↦sync'[i],
            msource↦i,mdest↦j])
  /\UNCHANGED ⟨⟨currentTerm,currentState,votedFor,end,syncTrack,electionVars⟩⟩

HandleUpdateSyncResponse(i)==
  /\\E m\in messages:
    LET j==m.msource IN
    /\m.mtype=UpdateSyncResponse
    /\m.mdest=i
    /\currentTerm[i]=m.mterm
    /\currentState[i]\in {Leader,LeaderCandidate}
    /\\/\m.mupdateSyncGranted
      /\syncTrack'=[syncTrack EXCEPT ![i][j]=m.msync]
    \/\\neg m.mupdateSyncGranted
      /\UNCHANGED syncTrack
  /\UNCHANGED ⟨⟨messages,serverVars,log,electionVars⟩⟩
----------------------------------------------------------------------------------------------
```

- *BecomeLeader*(*i*):主节点候选者 *i* 执行 *UpdateSync* 后,若有一个多数派 *Q* 的节点的同步号升级为 *currentTerm*[*i*],且 *i* 收到它们的确认(∀*q*∈*Q*:*syncTrack*[*i*][*q*]=*currentTerm*[*i*]),那么 *i* 转变为主节点,并提交日志中的所有日志项;

- *ClientRequest*(*i*):*i* 为主节点时,可以响应用户的请求,将用户的命令插入到日志中.

```
----------------------------------------------------------------------------
BecomeLeader(i)==
  /\currentState[i]=LeaderCandidate
  /\\E Q\in Quorums:\A q\in Q:(q=i\/syncTrack[i][q]=currentTerm[i])
                          /\elections'=elections\union
                          {[eterm|->currentTerm[i],
                            esync|->sync[i],
                            eleader|->i,
                            evotes|->Q,
                            evoterLog|->{log[k]:k\in Q},
                            elog|->log[i]]}
  /\sync'=[sync EXCEPT ![i]=currentTerm[i]]
  /\currentState'=[currentState EXCEPT ![i]=Leader]
  /\log'=[log EXCEPT ![i]=[n\in Index|->
                              IF log[i][n].term=sync[i] THEN
                                log[i][n].committed|->TRUE]
                              ELSE log[i][n]]]
  /\UNCHANGED <<messages,currentTerm,votedFor,end,syncTrack,halfElections>>

ClientRequest(i,v)==
  LET nextIndex==logTail(log[i])+1
      entry==[term|->currentTerm[i],
              date|->currentTerm[i],
              value|->v,
              committed|->FALSE]
  IN
    /\currentState[i]=Leader
    /\log'=[log EXCEPT ![i][nextIndex]=entry]
    /\UNCHANGED <<messages,serverVars,electionVars,syncTrack>>
----------------------------------------------------------------------------------------
Next 定义了次态关系.Spec 定义了完整的行为规约.
----------------------------------------------------------------------------------
Next==\/\E i\in Server:Restart(i)
       \/\E i\in Server:Timeout(i)
       \/\E i\in Server:UpdateTerm(i)
       \/\E i\in Server:RequestVote(i)
       \/\E i\in Server:HandleRequestVoteRequest(i)
       \/\E i\in Server:BecomeLeaderCandidate(i)
       \/\E i\in Server:BecomeLeader(i)
       \/\E i\in Server,v\in Value:ClientRequest(i,v)
```

\/*E i,j*\/\in *Server*:*RequestSync*(*i*)
\/*E i*\/\in *Server*:*HandleRequestSyncRequest*(*i*)
\/*E i*\/\in *Server*:*HandleRequestSyncResponse*(*i*)
\/*E i,j*\/\in *Server*:*UpdateSync*(*i*)
\/*E i*\/\in *Server*:*HandleUpdateSyncRequest*(*i*)
\/*E i*\/\in *Server*:*HandleUpdateSyncResponse*(*i*)

Spec==*Init*/\[·][*Next*]_*vars*

---

5 ParallelRaft-CE 的正确性证明

本节简要论证 ParallelRaft-CE 协议的正确性.ParallelRaft-CE 的正确性包括两部分:一是共识协议的一致性,二是 ParallelRaft-CE 中不会出现“幽灵日志”现象.由此可以得到复制状态机的安全性:节点间执行每个日志项的顺序没有冲突,状态机的状态一致.

性质 1. 节点的任期与同步号单调递增.

根据规约容易得到,任何节点的任期和同步号是单调递增的;

根据规约,节点只会修改任期等于同步号的日志项,而不会增加、删除或修改日志中任期小于同步号的日志项.

性质 2(选举的安全性). 每个任期最多选出一个主节点.

ParallelRaft-CE 中,从节点成为主节点需要两个阶段:一是从节点通过选举成为主节点候选者,二是主节点候选者转变为主节点.其中,选取主节点候选者的方式与 Raft 中选取主节点的方式相同.根据 Raft 选主的安全性,可以得到 ParallelRaft-CE 中每个任期最多选出一个主节点候选者;

主节点候选者可能成功转变为主节点,可能由于失效,或是任期更大的节点发起选举等原因没有转变为主节点,在这种情况下,系统进入下一任期.因此,每个任期最多选出一个主节点.

任期相同、编号相同的日志项一定包含相同的命令.根据选举的安全性,以及对任意编号,每个主节点最多添加一个日志项可以得到.这个性质与 Raft 相同.

性质 3. 设主节点候选者的任期为 t,同步号为 s,则 $s<t$ 且系统内不存在任期大于 s 的日志项.

首先,任何节点的同步号不大于任期.由于节点在发起选举需要增大任期(同步号不变),且在选举过程中任期与同步号不变,因此主节点候选者的同步号小于任期.设主节点候选者 i 的同步号为 s,任期为 $t(s<t)$;

对任意 $s<k<t$,不存在任期为 k 的主节点.使用反证法,假设存在节点 j 为任期 k 的主节点.根据规约,存在多数派节点 Q_1 为 i 投票,j 由主节点候选者转变为主节点,因此存在多数派节点 Q_2 升级同步号为 k.根据多数派的性质,$Q_1\cap Q_2\neq\varnothing$.设 $r\in Q_1\cap Q_2$.若 r 先为 i 投票,那么 r 升级任期为 t.由于 $t>k$,因此 r 之后会拒绝 j 升级同步号的请求,这与 $r\in Q_2$ 矛盾.若 r 先升级同步号为 k,那么由于 $k>s$,之后,r 会拒绝 i 的选举请求(注意,选举投票要比较同步号),这与 $r\in Q_1$ 矛盾.由此可知,不存在这样的 k;

对 $k\geqslant t$,不存在任期为 k 的主节点.若系统中存在任期为 k 的主节点,那么 k 在选举中获得多数派节点 Q 的投票,因此 Q 中节点的任期至少为 k.由于节点的任期递增,因此 i 发起选举时,Q 中的节点的任期要么大于 t,要么已经为任期为 t 的其他节点投过票,因此不会为 i 投票.这与 i 成为任期为 t 的主节点候选者矛盾.

综上,系统中没有出现过任期大于 s 的主节点,因此不存在任期大于 s 的日志项.

性质 4(主节点的完全性). 若主节点的任期为 t,则其日志中包括所有任期小于 t 且被提交的日志项.

ParallelRaft-CE 中,任期为 t_1 的主节点失效后,新的主节点候选者(任期为 t_2)会对任期为 t_1 的日志项进行重确认.在重确认的过程中,所有上一任期的日志项的状态被确定:被提交或者丢弃.重确认过程结束后,主节点候选者成为新的主节点.重确认的过程实质上是对每个位置执行 Paxos 协议的过程,因此可以保证取得共识,且之

前取得共识的日志项不会丢失.根据规约,当完成重确认后,一个多数派的节点升级同步号为 t_2,且由于节点的同步号递增,之后,同步号小于 t_2 的节点无法通过选举成为主节点候选者,因此已经取得共识的任期为 t_1 的日志项不会改变.

性质 5(一致性). 对任意两个节点,如果它们的同步号都大于某个任期 t,则它们的日志中包含相同的任期为 t 的日志项.

ParallelRaft-CE 的主节点根据从节点的同步号同步日志项,这与恢复阶段的正确性(性质 3 与性质 4),保证了节点间日志的一致性.

ParallelRaft-CE 中不会出现“幽灵日志”现象.

在 ParallelRaft-CE 中,新的主节点候选者通过重确认过程对上一主节点的日志项进行重确认.当重确认过程结束后,主节点候选者的同步号升级为任期.由于节点的同步号递增,选举机制保证了同步号落后的节点无法成为主节点.因此,系统不会重新提交之前任期的日志项.

6 模型检验与模拟测试

本节使用 TLC 模型检验工具,在模型检验模式与模拟模式下验证 ParallelRaft-CE 协议的正确性,并验证 ParallelRaft-SE 到 Multi-Paxos 的精化关系.

6.1 实验设置

在所有的实验中,我们调整参与者集合 *Proposer*(*Server*)、提议者集合 *Value*、提议编号集合 *Ballot*(*Term*)的大小,并将前两者设置为对称集[13],以提高 TLC 的验证效率.我们使用 10 个线程进行实验,以下是我们的实验统计结果.

模型检验模式的统计结果包括:已遍历(BFS 方式遍历)的系统状态图的直径、已遍历的状态的数量、已发现的不同状态的数量以及检验花费的时间(单位是 hh:mm:ss).在验证 ParallelRaft-SE 精化 Multi-Paxos 时,我们设置 TLC 已检验的不同状态数量超过 1 亿时,人为停止实验.在验证 ParallelRaft-CE 满足一致性时,我们设置 TLC 已检验的不同状态数超过 2 亿时,人为停止实验.

模拟模式下,我们使用随机种子,设置最大深度为 50.统计结果主要为已遍历的状态的数量.每组实验运行 6 小时.

实验使用的机器配置为: 2.40 GHz 10 核 CPU 以及 64GB 内存,TLC 版本号为 1.7.0

6.2 模型检验验证结果

6.2.1 ParallelRaft-SE 精化 Multi-Paxos

图 5 给出了在不同配置下使用模型检验模式验证 ParallelRaft-SE 精化 Multi-Paxos 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 $(hh:mm:ss)$
(3,2,2)	22	1183766512	104836664	10:05:25
(3,2,3)	23	899846293	102806000	09:54:49
(3,3,2)	22	950017774	100064549	12:49:10
(3,3,3)	22	828085252	100020363	12:40:07
(4,2,2)	24	1045345827	100093827	12:52:38
(4,2,3)	22	1483023804	100044759	09:42:07
(4,3,2)	24	1150819236	100101884	23:21:09
(4,3,3)	21	1452458568	100054689	20:04:08

Fig.5 Model checking results of verifying the refinement from ParallelRaft-SE to Multi-Paxos

图 5 ParallelRaft-SE 精化 Multi-Paxos 的验证结果

精化关系在 ParallelRaft-SE 的 TLA+规约中给出.从实验数据可以发现:参与者数量与提议值个数对实验的

规模以及检验时间的影响较大,而投票轮数对实验的规模影响相对较小.

6.2.2 ParallelRaft-CE 满足一致性

图 6 给出了在不同配置下使用模型检验模式验证 ParallelRaft-CE 满足一致性的验证结果.

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)	26	1274709286	201161468	01:51:02
(2,2,3)	24	1104433959	202169934	01:31:11
(2,3,2)	26	1218607009	200208445	02:27:13
(2,3,3)	24	1108870367	200916171	02:22:13
(3,2,2)	22	1370489381	200340341	03:02:17
(3,2,3)	19	1221006004	200913649	02:40:23
(3,3,2)	22	1369865234	200265437	05:44:01
(3,3,3)	19	1236091812	200041489	05:02:33

Fig.6 Model checking results of verifying that ParallelRaft-CE satisfies consistency

图 6 ParallelRaft-CE 满足一致性的验证结果

ParallelRaft-CE 的一致性形式化定义为:

AllEntries(*i*)=={⟨⟨*n*,*log*[*i*][*n*]⟩⟩:*n*\in *Index*}

Consistency==\A *i*,*j*\in *Server*:

{*e* \in *AllEntries*(*i*):*e*[2].*term*\geq 0/*e*[2].*term*<Min({*sync*[*i*],*sync*[*j*]})}=

{*e*\in *AllEntries*(*j*):*e*[2].*term*\geq 0/*e*[2].*term*<Min({*sync*[*i*],*sync*[*j*]})}

其中,*AllEntries*(*i*)为节点 *i* 的所有日志项(日志编号与日志的二元组).*Consistency* 定义了 ParallelRaft-CE 的一致性,即:任意两个节点,若它们的同步号都大于 *t*,那么它们的日志中有相同的任期为 *t* 的日志项.

ParallelRaft-CE 协议更加复杂,不同状态的数量较多,因此我们设置不同状态数达到 2 亿时停止实验.

6.3 模拟模式验证结果

6.3.1 ParallelRaft-SE 精化 Multi-Paxos

图 7 给出了在不同配置下使用模拟模式验证 ParallelRaft-SE 精化 Multi-Paxos 的实验结果.经过估算,在实验规模较小的情况下(3 个参与者,2 轮投票),状态图的直径为 30 左右;实验规模较大的情况下(4 个参与者,3 轮投票),状态图的直径为 50 左右.实验设置模拟模式的检测深度为 50.

TLC 模型 (参与者数量，提议值个数，投票轮数)	状态数	检验时间 ($hh:mm:ss$)
(3,2,2)	3203570873	06:00:01
(3,2,3)	2723736314	06:00:00
(3,3,2)	3137465261	06:00:00
(3,3,3)	2670655875	06:00:01
(4,2,2)	2911602481	06:00:00
(4,2,3)	2414873632	06:00:00
(4,3,2)	2976053370	06:00:00
(4,3,3)	2298047612	06:00:01

Fig.7 Simulation results of the refinement from ParallelRaft-SE to Multi-Paxos

图 7 模拟模式验证 ParallelRaft-SE 精化 Multi-Paxos 的实验结果

6.3.2 ParallelRaft-CE 满足一致性

图 8 给出了在不同配置下,使用模拟模式验证 ParallelRaft-CE 满足一致性的实验结果.实验同样设置最大深度为 50.

TLC 模型 (参与者数量，提议值个数，投票轮数)	状态数	检验时间 $(hh:mm:ss)$
(2,2,2)	4521671984	06:00:17
(2,2,3)	4209786847	06:00:20
(2,3,2)	4480532710	06:01:05
(2,3,3)	4091547805	06:00:12
(3,2,2)	3181834406	06:02:14
(3,2,3)	2722329411	06:00:10
(3,3,2)	3116899023	06:00:45
(3,3,3)	2690629148	06:04:19

Fig.8 Simulation results of the consistency of ParallelRaft-CE

图 8 模拟模式验证 ParallelRaft-CE 满足一致性的验证结果

7 相关工作

经典分布式共识协议 Multi-Paxos(Paxos)[8,9]衍生出了多种变体[23,24],如 Disk Paxos[25],Cheap Paoxs[26],Fast Paxos[27],Generalized Paxos[28],Stoppable Paxos[29],Vertical Paxos[30],Byzantine Paxos[31],EPaxos[32],TPaxos 等[18]. Multi-Paxos 支持乱序提交、顺序执行用户命令.Raft[10]也可以看作 Multi-Paxos(Paxos)的变体[33],它限制了乱序提交行为,通过顺序提交、顺序执行用户命令简化了协议设计.本文关注分布式文件系统 PolarFS 使用的 ParallelRaft 协议[7],它基于 Raft 实现了乱序提交、乱序执行,更适用于高并发系统.为了理清 ParallelRaft 与 Raft 之间的关系,我们提出了 ParallelRaft-SE.ParallelRaft-SE 是 Multi-Paxos 的一种变体,支持乱序提交、顺序执行用户命令.

利用无冲突命令之间的可交换性,可以提高共识协议的性能.在 Generalized Paxos[28]中,如果命令无冲突,用户可以绕过主节点,直接将命令广播给所有节点,减少通信开销、提高性能.微软公司提出的分布式复制框架 Tribble[34]利用多线程技术并发执行无冲突的命令项,同时保证状态一致性.ParallelRaft[7]根据命令的逻辑区块地址进行冲突判断,无冲突的命令可以乱序执行.我们分析了 ParallelRaft 的乱序执行机制,发现文献[7]中的描述忽略了可能会违反状态一致性的“幽灵日志”问题.因此,我们提出了 ParallelRaft-CE,它通过限制乱序提交阶段的并行度,避免了“幽灵日志”问题.

使用形式化规约语言描述分布式协议,并使用模型检验工具验证预期的性质,可以有效地增强人们对协议可靠性的信心[18].Lamport 使用 TLA+[13,15]描述了 Paxos[35],Fast Paxos[27],Byzantine Paxos[31]等协议[18],并使用模型检验工具 TLC 验证了这些协议(在受限规模下)的正确性.Ongaro 等人提出了 Paxos 的变体 Raft,并给出了 Raft 的 TLA+规约[22].本文给出了 ParallelRaft-SE 和 ParallelRaft-CE 的 TLA+规约,并验证了它们(在受限规模下)的正确性.

精化技术[12]有助于理解各种协议之间的关系.在研究 Paxos 时,Lamport 提出了共识问题的抽象描述 Consensus,给出了分布式共识问题的集中式解决方案 Voting[35],并构建了从 Paxos 到 Voting 以及从 Voting 到 Consensus 的精化关系[35].Yi 等人在研究腾讯公司 PaxosStore 系统中的 TPaxos 时,提出了 Voting 的一种变体 EagerVoting,并构建了从 TPaxos 到 EagerVoting 以及从 EagerVoting 到 Consensus 的精化关系[18].本文构建了从 ParallelRaft-SE 到 Multi-Paxos 的精化关系,证明了 ParallelRaft-SE 的正确性.

8 总结与未来工作

本文的目标是为 PolarFS 文件系统中使用的 ParallelRaft 协议[7](它支持乱序提交、乱序执行用户命令)提供严格的形式化规约并证明其正确性.首先,为了理清 ParallelRaft 与 Raft 之间的关系,我们提出了允许乱序提交、顺序执行的 ParallRaft-SE 协议,并建立了从 ParallelRaft-SE 到 Multi-Paxos 的精化关系;其次,我们发现现有的 ParallelRaft 描述忽略了可能会违反状态一致性的“幽灵日志”问题,并提出了 ParallelRaft-CE 协议.通过限制 ParallelRaft-SE 在乱序提交阶段的并行度,ParallelRaft-CE 避免了“幽灵日志”问题.最后,我们给出了 ParallelRaft-SE 和 ParallelRaft-CE 的 TLA+规约,并对协议参与者数量较小的情形使用 TLC 模型检验工具,验证了从

ParallelRaft-SE 到 Multi-Paxos 的精化关系以及 ParallelRaft-CE 的正确性.

目前,我们正在使用 TLAPS(TLA+Proof System)[14,36]定理证明系统为 ParallelRaft-CE(以及 Raft)开发机械化正确性证明.此外,我们还将从顺序/乱序提交、顺序/乱序执行的角度对已知的分布式共识协议进行综述,并利用形式化方法研究它们之间的关系.

References:

[1] Fischer MJ, Lynch NA, Paterson MS. Impossibility of distributed consensus with one faulty process. Journal of the ACM (JACM), 1985,32(2):374−382. [doi: https://doi.org/10.1145/3149.214121]

[2] Herlihy M. Wait-Free synchronization. ACM Trans. on Programming Languages and Systems (TOPLAS), 1991,13(1):124−149. [doi: https://doi.org/10.1145/114005.102808]

[3] Weil SA. Ceph: Reliable, scalable, and high-performance distributed storage [Ph.D. Thesis]. Santa Cruz: University of California, 2007.

[4] Corbett JC, Dean J, Epstein M, Fikes A, Frost C, Furman JJ, Ghemawat S, Gubarev A, Heiser C, Hochschild P, Hsieh W. Spanner: Google's globally distributed database. ACM Trans. on Computer Systems (TOCS), 2013,31(3):1−22.

[5] https://www.mysql.com/

[6] Zheng J, Lin Q, Xu J, Wei C, Zeng C, Yang P, Zhang Y. PaxosStore: High-availability storage made practical in WeChat. Proc. of the VLDB Endowment, 2017,10(12):1730−1741.

[7] Cao W, Liu Z, Wang P, Chen S, Zhu C, Zheng S, Wang Y, Ma G. PolarFS: An ultra-low latency and failure resilient distributed file system for shared storage cloud database. Proc. of the VLDB Endowment, 2018,11(12):1849−1862.

[8] Lamport L. Paxos made simple. ACM Sigact News, 2001,32(4):18−25.

[9] Lamport L. The part-time parliament. ACM Trans. on Computer Systems (TOCS), 1998,16(2):133−169.

[10] Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. In: Proc. of the 2014 USENIX Annual Technical Conf. ({USENIX}{ATC} 2014). 2014. 305−319.

[11] Schneider, Fred B. Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Computing Surveys, 1990,22(4):299−319.

[12] Abadi M, Lamport L. The existence of refinement mappings. Theoretical Computer Science, 1991,82(2):253−284.

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

[14] Lamport L. The TLA+ hyperbook. 2015. http://research.microsoft.com/enus/um/people/lamport/tla/hyperbook.html

[15] Lamport L. The temporal logic of actions. ACM Trans. on Programming Languages and Systems (TOPLAS), 1994,16(3):872−923.

[16] Yu Y, Manolios P, Lamport L. Model checking TLA+ specifications. In: Proc. of the Advanced Research Working Conf. on Correct Hardware Design and Verification Methods. Berlin, Heidelberg: Springer-Verlag, 1999. 54−66.

[17] https://github.com/HappyCS-Gu/Parallel-Raft-tla

[18] Yi XC, Wei HF, Huang Y, Qiao L, Lü J. TPaxos consensus protocol in PaxosStore: Derivation, specification, and refinement. Ruan Jian Xue Bao/Journal of Software, 2020,31(8):2336−2361 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5964.htm [doi: 10.13328/j.cnki.jos.005964]

[19] Ji Y, Wei HF, Huang Y, Lü J. Specifying and verifying CRDT protocols using TLA+. Ruan Jian Xue Bao/Journal of Software, 2020,31(5):1332−1352 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5956.htm [doi: 10.13328/j.cnki.jos.005956]

[20] Lamport L. Summary of tla+. http://lamport.azurewebsites.net/tla/summary-standalone.pdf

[21] Yuan D, Luo Y, Zhuang X, Rodrigues GR, Zhao X, Zhang Y, Jain PU, Stumm M. Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. In: Proc. of the 11th USENIX Symp. on Operating Systems Design and Implementation (OSDI). 2014. 249−265.

[22] Ongaro D. Consensus: Bridging theory and practice [Ph.D. Thesis]. Stanford University, 2014.

[23] Lampson B. The ABCD's of Paxos. In: Proc. of the 20th Annual ACM Symp. on Principles of Distributed Computing (PODC), Vol.1. 2001. 13.

[24] Van Renesse R, Altinbuken D. Paxos made moderately complex. ACM Computing Surveys (CSUR), 2015,47(3):1−36.

[25] Gafni E, Lamport L. Disk Paxos. Distributed Computing, 2003,16(1):1−20.

[26] Lamport L, Massa M. Cheap Paxos. In: Proc. of the Int'l Conf. on Dependable Systems and Networks. 2004. 307−314.

[27] Lamport L. Fast Paxos. Distributed Computing, 2006,19(2):79−103.

[28] Lamport L. Generalized consensus and Paxos. Technical Report, Microsoft Research, 2005.

[29] Lamport L, Malkhi D, Zhou L. Stoppable Paxos. Technical Report, Microsoft Research, 2008.

[30] Lamport L, Malkhi D, Zhou L. Vertical Paxos and primary-backup replication. In: Proc. of the 28th ACM Symp. on Principles of Distributed Computing. 2009. 312−313.

[31] Lamport L. Byzantizing Paxos by refinement. In: Proc. of the Int'l Symp. on Distributed Computing. 2011. 211−224.

[32] Moraru I, Andersen DG, Kaminsky M. There is more consensus in egalitarian parliaments. In: Proc. of the 24th ACM Symp. on Operating Systems Principles. 358−372.

[33] Wang Z, Zhao C, Mu S, Chen H, Li J. On the parallels between Paxos and Raft, and how to port optimizations. In: Proc. of the 2019 ACM Symp. on Principles of Distributed Computing. 2019. 445−454.

[34] Guo Z, Hong C, Yang M, Zhou D, Zhou L, Zhuang L. Paxos made parallel. Technical Report, Microsoft Research Asia, 2012. 118.

[35] Lamport L, Merz S, Doligez D. A TLA + specification of Paxos and its refinement. 2019. https://github.com/tlaplus/Examples/tree/master/specifications/Paxos

[36] Chaudhuri K, Doligez D, Lamport L, Merz SA. TLA+ proof system. In: Proc. of the LPAR Workshops, CEUR Workshop. 2008. 17−37.

附中文参考文献:

[18] 易星辰,魏恒峰,黄宇,乔磊,吕建.PaxosStore 中共识协议 TPaxos 的推导、规约与精化.软件学报,2020,31(8):2336−2361. http://www.jos.org.cn/1000-9825/5964.htm [doi: 10.13328/j.cnki.jos.005964]

[19] 纪业,魏恒峰,黄宇,吕建.CRDT 协议的 TLA+描述与验证.软件学报,2020,31(5):1332−1352. http://www.jos.org.cn/1000-9825/31/5956.htm [doi: 10.13328/j.cnki.jos.005956]

谷晓松(1997－),男,学士,CCF 学生会员,主要研究领域为分布式数据一致性,分布式系统,形式化方法.

乔磊(1982－),男,博士,研究员,CCF 专业会员,主要研究领域为航天器嵌入式操作系统设计及验证.

魏恒峰(1986－),男,博士,CCF 专业会员,主要研究领域为分布数据一致性,形式化方法.

黄 宇(1982－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为分布式算法,分布式系统,网络化软件系统.