计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2024,33(6):242−250 [doi: 10.15888/j.cnki.csa.009550]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于日志副本的 Raft 共识算法优化①

雷　磊[1], 胡晓鹏[1], 黄　岩[2]

[1](西南交通大学 计算机与人工智能学院, 成都 611756)
[2](云和恩墨 (北京) 信息技术有限公司, 北京 100010)
通信作者: 胡晓鹏, E-mail: xphu@swjtu.edu.cn

摘　要: 在基于三副本策略的分布式存储系统中, 当存储节点上的硬盘出现故障时, 常见的处理方式是等待系统预设的时间. 如果该故障硬盘超时未恢复, 才开始恢复故障硬盘上的副本. 这种处理方式存在的问题是, 当三副本组中存在故障副本时, 如果该副本组再有一个副本所在的硬盘发生故障, 将导致系统无法继续提供服务, 且不能自动恢复. 本文提出一种基于日志副本的改进的 Raft 共识算法, 即 LR-Raft (log replica based Raft), 日志副本没有完整状态机, 可以快速加入集群, 并参与投票与共识, 提升了存在故障硬盘时系统的可用性; 可以解决短时间内三副本中两个副本故障导致集群不可用和丢失数据的问题. 实验结果表明, 在副本组中引入日志副本后, 与原 Raft 相比, LR-Raft 在不同的工作负载下读写时延均明显降低, 吞吐量显著提升.

关键词: 分布式存储系统; Raft 共识算法; 故障处理; 日志副本; 日志压缩优化

引用格式: 雷磊,胡晓鹏,黄岩.基于日志副本的 Raft 共识算法优化.计算机系统应用,2024,33(6):242–250. http://www.c-s-a.org.cn/1003-3254/9550.html

Optimization of Raft Consensus Algorithm Based on Log Replica

LEI Lei[1], HU Xiao-Peng[1], HUANG Yan[2]

[1](School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu 611756, China)
[2](Yunhe Enmo (Beijing) Information Technology Co. Ltd., Beijing 100010, China)

Abstract: In a distributed storage system based on a three-replica strategy, when a hard disk on the storage node fails, a common processing method is to wait for the system's preset time. If the faulty disk doesn't recover within the specified timeout, the recovery of the replicas on the faulty hard disk will begin. The issue with this handling approach is that when there is a faulty replica within the three-replica group, another disk failure in the same group will result in the system being unable to continue providing services and recover automatically. This study introduces an improved Raft consensus algorithm based on log replicas, namely log replica based Raft (LR-Raft). Log replicas do not have a complete state machine, allowing them to quickly join the cluster and participate in voting and consensus, thereby enhancing system availability in the presence of a faulty disk. It can address the problem of unavailability and data loss in the cluster caused by the failure of two replicas in a three-replica setup in a short period. The experimental results indicate that with the introduction of log replicas into the replica group, LR-Raft significantly reduces read and write latency and substantially improves throughput compared to the original Raft across various workloads.

Key words: distributed storage system; Raft consensus algorithm; fault handling; log replica; log compression optimization

① 基金项目: 河北省自然科学基金 (F2022105033)
收稿时间: 2023-12-29; 修改时间: 2024-01-29; 采用时间: 2024-02-26; csa 在线出版时间: 2024-04-30
CNKI 网络首发时间: 2024-05-07

分布式存储系统一般由若干存储节点 (storage node, SN) 构成, 通常采用多副本冗余策略来提高数据可靠性. 副本 (replica) 是指在不同的存储节点上持久化相同的数据, 并使用共识算法 (consensus algorithm) 为分布式存储系统提供一致性和容错服务. 近年来, Raft[1−3]逐渐取代 Paxos 成为首选的共识算法, Raft 相较于 Paxos 的优势在于其更为直观且易于理解, 设计上更具模块化, 并采用高效的领导者选举算法[4]. 在过去的几年内, 研究者们尝试从多个方向对 Raft 算法进行改进. ParallelRaft[5]提出了乱序确认、乱序提交和乱序应用的方案, 用于解决 Raft 算法顺序确认和提交在高并发环境下带来的性能问题. 谷晓松等人[6]和王进等人[7]同样对 Raft 算法进行了乱序提交的改进, Parallel-Raft 算法本质上是借鉴了 Multi-Paxos[8]允许日志空洞的特性. Guo 等人[9]在 Raft 日志的定义中引入了 Special Mark 的概念, 即在每条日志中嵌入 Special Mark, 解决了原始 Raft 算法中 Follower 副本日志修复算法浪费网络带宽和耗时较长的问题. Arora 等人[10]认为 Raft 算法的单 Leader 会成为读写性能的瓶颈, 提出了 Quorum Read 的方案, 即 Follower Read. Wang 等人 [11−13]提出了 Commit Return、Fastest Return 两种方式来加速写请求, 即 Leader 不需要 Apply 就返回客户端写成功, 但会增加客户端读请求的时延. 还提出了 Immediate Read、Optimal Read、Relaxed Read 这 3 种方式来加速读请求, 基本原理是同时读取状态机中已经 Apply 和内存中已提交但未 Apply 的数据. Pâris 等人[14]提出了 Witness 副本机制, Witness 副本在集群中拥有投票权, 但是没有任何实际数据, 也不能处理任何 IO 请求. 通过结合动态 Quorum 机制, 使得集群存在 3 个正常副本和 1 个 Witness 副本时, 即使在任意两个副本故障的情况下, 系统仍能保持可用状态, 同时也降低了数据存储的冗余度.

综上, 对 Raft 算法的改进主要集中在如何提升读写性能、日志同步效率和选举效率, 而对 Raft 算法做故障处理的优化研究较少.

在大规模分布式存储系统中, 硬盘故障是常见的故障类型, 特别是硬盘介质损坏可能导致持久性故障, 进而造成数据永久丢失. 通常, 系统在检测到硬盘故障后会等待一段预设的时间来容忍硬盘短时间离线, 超过该时间则判定故障硬盘无法恢复. 此时, 系统将移除故障硬盘中的数据副本, 并在其他可用的硬盘上创建新的数据副本以进行恢复.

然而, 这种故障处理方式存在问题. 以 Raft 三副本组为例, 当一个副本故障时, 恢复的时间较长, 如果此时另一个副本也故障, 整个 Raft 三副本组将失去可用性. 恢复时间通常受到故障副本的数据量影响, 难以有效优化. 从检测到副本发生故障到开始恢复存在一段时间, 用于容忍故障副本自动恢复. 如果在检测到故障副本后立即触发恢复操作, 而故障副本在一段时间后自动恢复, 那么这个恢复操作将导致不必要的资源浪费. 因此, 需要一种能够快速加入副本组参与共识, 同时不会造成资源浪费的机制. 本文引入了无状态机的日志副本 (log replica) 机制, 提出了一种基于日志副本的增强共识算法 LR-Raft (log replica based Raft). 在发生硬盘永久性故障时, 日志副本能快速加入集群并参与共识, 且能够长期存在于集群中. 进一步提出了日志压缩优化方案, 解决了引入日志副本后可能数据丢失的问题. 本文的主要研究工作如下.

(1) 分析了基于 Raft 的分布式存储系统的原始故障处理流程会导致系统可用性和读写性能下降的问题, 在 Raft 共识算法中引入日志副本, 优化了故障处理流程, 使系统故障处理过程中仍然具有良好的读写性能和可用性.

(2) 分析了引入日志副本可能出现丢失数据的场景, 提出了 Raft 日志压缩优化方案, 解决了丢失数据问题, 同时提升了日志复制的效率.

(3) 实现了 LR-Raft 共识算法, 并作为企业级分布式存储软件的共识模块, 利用 fio 完成了一系列的性能测试, 可以观察到 LR-Raft 在故障处理过程中表现出更好的系统吞吐量和读写性能.

1 相关背景

1.1 Raft 共识算法

Raft 是一种基于复制状态机 (replicated state machine) 模型的分布式共识算法. Raft 将复杂的共识过程简化为 Leader 选举和日志复制两个主要部分. Raft 中的副本有 3 种状态: 领导者 (Leader)、跟随者 (Follower) 和候选者 (Candidate). 在多副本集群中, 通过简单多数选举算法选出一个 Leader 副本 (简称为 Leader), 由 Leader 接收客户端数据写请求.

1.1.1 Raft 选举流程

副本组 (replica group, RG) 中的 Leader 在其任期

(Term) 内定期向所有 Follower 发送心跳, 以维持其 Leader 状态. 如果 Follower 在选举超时时间内没有收到 Leader 发出的心跳, 则认为当前任期 (current term) 的 Leader 已经下台. Follower 将自身状态转换为 Candidate, 并增加自己的任期, 向其他副本发送投票请求, 具有更高的 Term 和 Log Last Index 的 Candidate 在获得大多数选票后转换成 Leader, 提供读写服务.

1.1.2 Raft 三副本写流程

在采用 Raft 三副本策略的分布式存储系统中, Leader 接收客户端写请求, 将请求包装成日志条目 (log entry) 并分配任期和唯一索引, 按顺序持久化到本地非易失存储设备. 通过 AppendEntries RPC 将日志条目发送给 Follower. Follower 收到日志条目后进行安全性检查和持久化, 成功后通过 AppendEntriesResult RPC 告知 Leader 完成了 Append 操作. 当大多数副本完成了 Append 后, Leader 更新 Committed Index, 应用 (Apply) 日志条目到状态机, 并通过心跳携带最新的 Committed Index 通知其他副本完成 Apply. Leader 完成 Apply 后即可返回客户端写成功. Raft 三副本的写流程如图 1 所示.

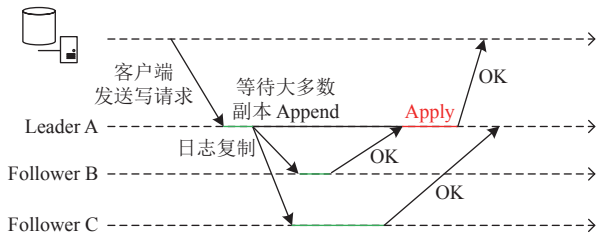


图 1 Raft 三副本写流程

1.1.3 Raft 日志压缩机制

随着客户端写请求的增加, Raft 的日志条目数量不断增长. 为提高日志复制效率, 通常将日志保存在存储节点内存中. 考虑到存储节点可能会发生故障, 导致内存中的日志失效, 因此需要将日志持久化到硬盘. 图 2 给出了内存中包含的不同状态的日志, 其中 Log First Index 表示内存中第 1 条应用到状态机的日志条目索引, Applied Index 表示最后一条应用到状态机的日志条目索引, Committed Index 表示已经提交的最大索引, Log Last Index 表示内存中最后一条日志条目索引. 为了维护性能, Raft 需要定期清理内存及硬盘中过量的已应用 (Applied) 的日志条目. 具体策略是, 如果内存中已应用的日志条目数量超过了阈值 (Threshold), 则可以删除内存和硬盘中最近应用的 Threshold 条日志条目, 并更新内存日志的 Log First Index. Raft 不会因为部分未同步完成的副本而延迟删除日志. 删除已应用的内存日志后, 如果落后的副本需要 Log First Index 之前的日志条目, 则无法通过日志复制的方式同步, 只能以发送 InstallSnapshot RPC 将状态机数据同步给落后副本.

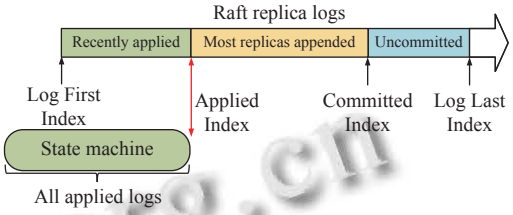


图 2 Raft 副本中不同状态的日志

1.2 故障处理流程

在分布式存储系统中, 通常由监视器 (Monitor) 组件负责检测存储节点及其硬盘的健康状态, 并维护集群状态的映射 (Map). 对于许多分布式存储系统 (例如 TiDB[15] 和 Ceph[16]), 处理硬盘故障的流程可以概括为以下 3 个阶段.

(1) 故障检测期: 从故障发生开始, 到监视器组件检测到故障为止. 故障可能包括硬盘故障、网络分区、存储节点故障、程序异常等, 这段时间的时长一般在秒级.

(2) 恢复等待期: 从监视器组件检测到故障开始, 到向相关副本发送新配置结束. 新配置包含了新的副本成员和要被踢出集群的故障副本. 这段时间的时长一般是分钟级. 监视器组件之所以在检测到故障后不立即发送新配置, 是为了容忍可以自动恢复的瞬时故障 (例如硬盘短暂离线、程序异常重启等).

(3) 数据重构期: 从监视器组件发送新配置开始, 到新的副本加入并完成数据重构结束. 这段时间的时长取决于副本恢复的数据量.

图 3 给出了采用三副本策略的分布式存储系统的故障处理流程. 副本 A、B、C 组成一个副本组, 假设某个时刻副本 B 发生故障, 监视器组件检测到副本 B 故障后, 在恢复等待期下发新配置, 系统移除副本 B, 加入新副本 D, 并对副本 D 进行数据重构.

2 LR-Raft 共识算法

2.1 故障处理流程

在分布式存储系统中, 所有存储节点上的多个硬盘被集中管理[16,17], 并抽象为存储池 (Pool). 客户端提

交的写请求数据被分割为固定大小的块 (Chunk), 并由放置组 (placement group, PG) 进行管理. 在使用 Raft 作为共识协议时, 一个放置组实际对应一个副本组. 图 4(a) 展示了一个正常状态下的分布式存储系统, 其中包含 3 个存储节点 (node1、node2 和 node3). 每个节点均配备 2 块硬盘 (SSD), 共 6 个硬盘 (从 SSD1 至 SSD6). 整个存储池中有 3 个副本组, 分别命名为 RG1、RG2 和 RG3. 每块硬盘上分别存储着 1–2 个副本的数据. 在某个时刻, 硬盘 SSD4 发生了永久性故障. 通常情况下, 可能无法立即获得现成的备用或热备硬盘进行替换, 那么在这段故障时间内, 副本组 RG2、RG3 中只有 2 个正常副本. 如果副本组 RG2、RG3 再有 1 个副本对应的硬盘发生故障, 这 2 个副本组将无法正常工作, 失去可用性. 因此, 当务之急是要想办法尽快让 RG2、RG3 恢复到 3 个正常副本. 如果 SSD3 上有一些剩余存储空间, 其大小无法容纳先前 SSD4 上副本 2、3 恢复后包含的完整状态机数据, 但可以容纳 SSD4 发生故障后副本 2、3 需要存储的日志数据, 那么就用 SSD3 上的这些剩余空间来存储副本 2、3 的日志数据. 像这样仅存储日志数据而没有状态机的副本, 被称为日志副本 (log replica). 日志副本具备接收 Leader 发送的日志、参与投票与共识的能力, 但不执行 Apply 操作. 当被选为 Leader 后, 无法处理客户端请求, 需要及时转移 Leader 权给其他副本. 此外, 日志副本能够接收 InstallSnapshot RPC, 但仅应用该 RPC 中的配置和元数据. 图 4(b) 展示了 SSD4 发生故障后, 在 SSD3 上建立日志副本 2、3 的情形.

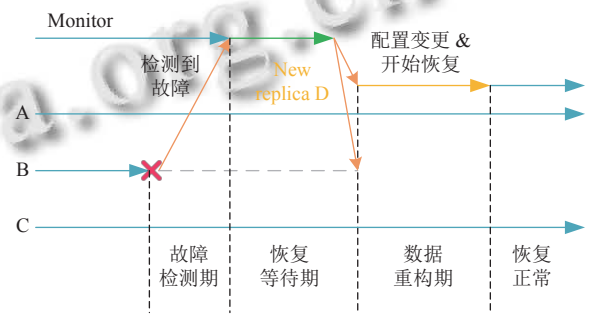
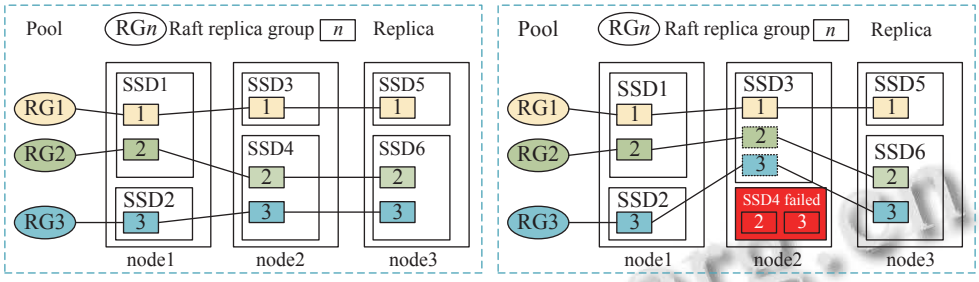


图 3 原始的故障处理流程



(a) 正常状态 (b) SSD4 故障后迁移数据到 SSD3

图 4 硬盘故障下的 Raft 副本数据迁移

由于引入了日志副本这种实体, 就需要对原有的 Raft 协议进行优化, 我们将优化后的 Raft 协议称为 LR-Raft.

引入日志副本后, 基于 Raft 的原始故障处理流程被优化为如图 5 所示的过程. 监视器组件检测到一个硬盘发生故障, 在该故障硬盘上可能保存了某些副本组的若干副本, 假设副本 B 就是其中之一, 并且 A、B、C 是一个副本组 (副本 A、B、C 一般分布在不同的硬盘上). 监视器组件知晓副本 B 发生故障后, 立即发起配置变更, 下发新的配置, 为故障涉及的副本组补充新的日志副本 D, 并移除故障副本 B. 在恢复等待期, 如果故障硬盘恢复 (对应瞬时故障) 正常, 则监视器组件发送新的配置, 将原始副本重新加入集群, 并移除日志副本. 如果故障硬盘无法恢复 (对应永久性故障), 则在恢复等待期结束后, 监视器组件通知日志副本进行数据重构, 重构完成后转成正常副本.

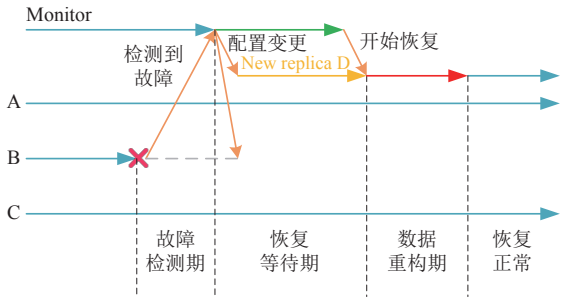


图 5 优化的故障处理流程

通过对比图 5 和图 3 所示的故障处理流程, 在原始故障处理流程中, 一个副本组在恢复等待期内仅有

2 个副本正常工作. 然而, 在优化的故障处理流程中, 一个副本组在恢复等待期内仍有 3 个副本处于正常工作状态. 这一改进显著增强了系统的可用性和容错能力.

硬盘故障导致需要恢复其上所有副本的数据, 这个时间通常以小时记, 如果在 1 h 内, 连续两块硬盘发生故障, 部分 Raft 三副本组就有可能失去可用, 而 LR-Raft 加入日志副本所需数据极少, 可以以秒记.

2.2 日志副本加入副本组

Raft 允许集群在正常运行期间进行配置变更, 例如向某个副本组中新增或删除副本. 这种配置变更也利用了 Raft 共识算法的逻辑. 由 Leader 接收监视器组件下发的配置变更, 将其包装为日志条目, 并标记为配置信息, 通过日志复制的方式同步给其他副本. 一旦大多数副本获取到最新的配置信息, 就能够了解新增或删除副本的配置变更, 从而替换掉旧的配置信息, 并将新的配置信息应用到状态机.

LR-Raft 算法中, Leader 更新配置时, 如果发现新加入的副本类型是日志副本, 虽然日志副本没有状态机, 但 Leader 仍需先发送 InstallSnapshot RPC 给日志副本. 此 RPC 中并不携带状态机数据, 其中 last Include Index 设置为 Log First Index–1, 并且携带最新的配置以及对应的配置索引 (Config Index). 日志副本接收 InstallSnapshot RPC 后, 应用其中最新配置, 如果后续接收了小于 Config Index 的配置, 则直接忽略. 特别地, 当日志副本第 1 次接收 InstallSnapshot RPC 时, last Include Index 将用于日志副本第 1 次接收日志时的日志匹配检查. 由于内存中的已应用的日志条目需要定期清理且通常较少, 因此日志副本能够快速赶上最新日志, 并参与共识.

如图 6(a) 所示, 假设副本 A、B、C 是一个副本组. 某个时刻, 监视器组件检测到副本 B 发生故障, 在其他硬盘上找到空闲存储空间补充日志副本 D. 监视器组件下发新的配置以加入日志副本 D, 副本 A (Leader) 将新配置信息包装为 Index 为 103 的日志条目. 在副本 A 和 C 完成应用 (Apply) 该日志条目后, 副本组中大多数副本 (A、C) 对加入日志副本 D 达成共识, 表示日志副本 D 正式加入该副本组, 如图 6(b) 所示. 然后, 副本 A 将最近的一些日志条目 (Index 为 101、102、103) 发送给日志副本 D, 而无需发送完整状态机数据, 如图 6(c) 所示. 最后, 按照同样的配置变更方式移除副本 B.

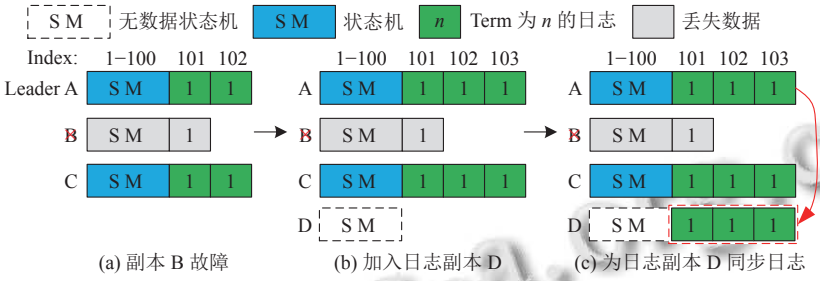


图 6 一个副本故障增加日志副本

在图 6 所描述的故障场景下, 日志副本 D 加入后, 该副本组由 2 个正常副本 (A、C) 和 1 个日志副本 (D) 组成. 如果后来副本 C 也发生故障, 监视器组件仍然能够成功下发新的配置并加入新的日志副本 E, 如图 7 所示. 此时, 这个副本组由 1 个正常副本 (A) 和 2 个日志副本 (D、E) 组成, 仍然保持可用状态.

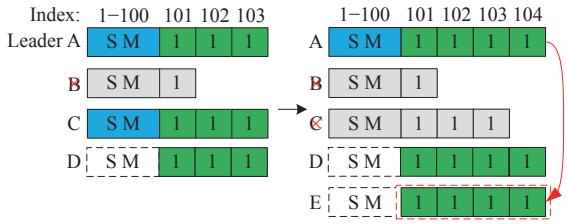


图 7 另一个副本故障下再次加入日志副本

LR-Raft 算法在图 6 和图 7 所示的情况下, 当三副本组中存在故障副本时, 通过日志副本迅速替换故障副本, 而无需经历漫长的状态机数据恢复时间, 即可加入副本组并参与共识. 在这种情况下, 三副本组仍能够容忍 1 个正常副本的故障. LR-Raft 算法通过提高系统在发生故障后一段时间内的容错能力.

2.3 日志副本当选 Leader 并让权

一个副本组中存在 2 个正常副本和 1 个日志副本的情况下, LR-Raft 算法为了容忍其中 1 个正常副本也发生故障, 不能限制日志副本当选 Leader 的能力. 考虑图 8(a) 所示的情况, 1 个副本组由 2 个正常副本 A、B (Leader) 和日志副本 C 组成. 当副本 B 发生故障后, 正

常副本 A 由于没有最新的日志, 不能从选举中获胜. 因此, 仅有日志副本 C 能当选 Leader, 如图 8(b) 所示. 日志副本 C 将其日志同步给正常副本 A, 如图 8(c) 所示. 由于日志副本 C 没有完整的状态机数据, 无法处理客户端的读写请求. 因此, 在同步完成后, 需要进行 Leader 权的转移.

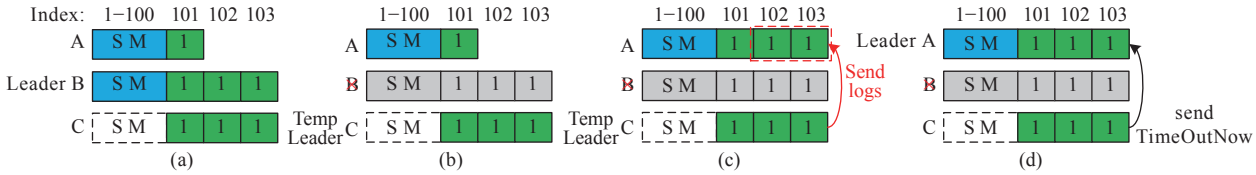


图 8 日志副本当选 Leader 及让权

当日志副本作为 Leader 接收到 Follower 回复的 AppendEntriesResult RPC, 若其中的 Last_index 等于自身的 Last_index 且该 Follower 为正常副本时, 将发送 TimeOutNow RPC 给该 Follower. Raft 算法[2]中的 TimeOutNow RPC 本质是通知接收到该 RPC 的 Follower 副本立即触发选举超时并发起选举. 在图 8(d) 所示场景中, 正常副本 A 收到日志副本的 TimeOutNow RPC 之后会立即发起选举, 并当选为 Leader.

2.4 Raft 日志压缩优化

在原始 Raft 算法中, 为了防止日志无限增加占用过多内存和存储空间, 除了利用状态机对日志进行压缩 (Compaction) 外, 还需要定期删除内存和硬盘中已应用的日志. 考虑如图 9 所示的情况, 一个副本组由两个正常副本 A、B (Leader) 和日志副本 C 组成, 它们的 Log First Index 都为 1. 由于日志副本能够参与共识, 副本 B、C 的 Committed Index 为 5, 而正常副本 A 暂时落后, 其 Committed Index 为 3, 如图 9(a) 所示. 假设副本 B 和 C 已应用的日志条目数达到了日志压缩的阈值 4, 各自删除了 Index 为 1–4 的日志条目, 如图 9(b) 所示. 后来, 如果副本 B 发生故障, 按照 LR-Raft 规定, 日志副本 C 被选为新的 Leader, 需要将 Index 为 4–5 的日志条目同步给正常副本 A, 然而日志副本 C 没有状态机, 也已经没有 Index 为 4 的日志条目, 此时便发生了丢失数据的问题 (如图 9(c) 所示), 即日志副本 (C) 和正常副本 (A) 无法通过 LR-Raft 算法恢复出完整的已提交的数据, 需要引入新的机制来解决这个问题.

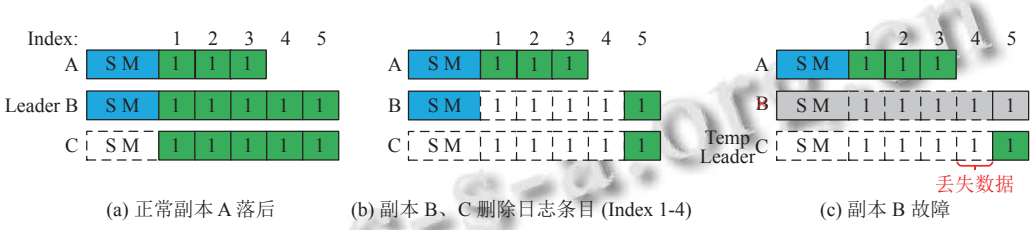


图 9 日志副本数据丢失的情况

分析图 9(a) 所示场景可知, 当副本 B 和 C 执行日志删除操作时, 应该只删除 Index 为 1–3 的日志条目, 保留 Index 为 4 的日志条目. 在一个副本组中, Leader 维护了所有副本的匹配索引 (Match Index), 用于记录每个副本最大被复制的日志条目的索引. 所有副本匹配索引的最小值被称为最慢匹配索引 (Min Match Index). 在图 9(a) 中, Leader 副本 (B) 维护了副本 A 和 C 的 Match Index 分别为 3 和 5, 则 Min Match Index 为 3. 因此, 在执行日志条目删除时, 应该只删除索引在 Log First Index 和 Min Match Index 之间的日志条目. 删除后, 副本 B 和 C 的 Log First Index 在原始值 1 的基础上加上阈值 (4), 更新为 5, Min Match Index 仍然为 3.

显然, 副本 B、C 的索引为 Min Match Index 和 Log First Index 之间的日志条目是为了避免丢失数据, 并且在内存和硬盘上都有保留, 浪费了存储空间. 一般而言, 每个日志条目中包含元数据和客户端数据, 元数据包含索引 (Index)、任期 (Term) 等, 通常占比较小, 而客户端数据占比较大. 为了防止日志条目占用过多内存, 对于副本 B、C 的索引为 Min Match Index 和 Log First Index 之间的日志条目, 在内存中可以删除其包含的客户端数据, 仅保留元数据, 但在硬盘中暂存完整的日志条目数据. 那么, 一个副本在内存中的日志状

态就应该进一步被细化, 即将已经应用的日志条目分解为仅包含元数据和包含完整数据的, 如图 10 所示. 其中索引为 Min Match Index 和 Log First Index 之间的日志条目仅包含元数据 (only metadata log). 此外 Leader 还需要将 Min Match Index 通过心跳同步给其他 Follower.

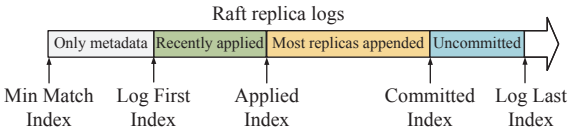


图 10 副本在内存中不同状态的日志

对于图 9(a) 所示的场景, 在执行优化后的日志压缩策略之后, 副本 B、C 的 Index 为 4 的日志条目在内存中仅包含元数据, 如图 11 所示. 后续, 当 Leader 需要发送元数据日志条目给某些落后副本时, 则应从硬盘中重新加载数据. 这种日志压缩优化方案解决了引入日志副本后可能丢失数据的问题, 且优势在于只对 Raft 的原有的日志复制逻辑做了最小的改动, 保证协议的一致性.

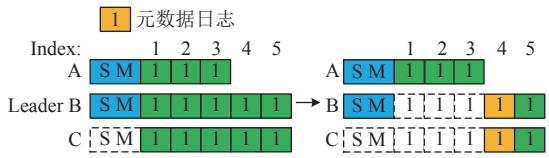


图 11 删除日志条目时保留日志条目中的元数据

3 实验结果与分析

本文使用异步 C 语言实现了 LR-Raft 和 Raft 共识算法, 并采用模块化设计使其作为分布式存储软件的共识模块, 主要对两种算法进行了比较. 实验集群由 4 个节点组成, 其中 3 个节点作为存储节点, 1 个作为测试工具运行节点. 存储节点的配置如表 1 所示. I/O 测试工具为 fio[18]. fio 是针对文件系统或磁盘进行压力测试的工具. 本实验使用的工作负载是混合随机 7:3 读写. 单次 I/O 的块文件大小为 4 KB, fio 的进程数配置为 36. 在接下来的实验中, 将采用系统吞吐量和平均、P90、P95、P99 时延作为性能指标, P99 时延代表性能测试的所有请求中时延最大的 1% 的时延, 是用来衡量分布式存储系统性能的重要指标. 吞吐量以 kops 为计量单位, 具体表示每秒执行的数千次操作数.

在总体实验中, 需要先将 50 GB 的数据写入集群, 以确保测试结果的准确性和可靠性. 实验中主要比较了 3 种模式下的性能指标, 即 Raft 的全副本写模式 (Raft Full) 和大多数副本写模式 (Raft Majority)、LR-Raft 的 2 个正常副本和 1 个日志副本组合的大多数写模式.

表 1 集群存储节点配置

配置项目	配置描述
CPU	Xeon(R) Silver 4314
操作系统	CentOS 7.0
内存	18 GB
硬盘	3.6 TB × 7
以太网卡	Broadcom BCM5720
RDMA网卡	Broadcom BCM57416

3.1 吞吐量测试实验

LR-Raft 在 4 KB 随机混合读写 (7:3) 的工作负载下, 其读写操作吞吐量较 Raft Full 和 Raft Majority 两种模式都更高. 实验结果如图 12 所示, LR-Raft 的读写操作总吞吐量相较于 Raft Majority 提升了 10.17%, 相较于 Raft Full 提升了 10.81%. 这是因为 LR-Raft 三副本中的日志副本并没有状态机, 该模式相较于其他模式少了 1/3 的 Raft Apply 操作, 降低了硬盘的工作负载, 其所在的节点的吞吐量也可以得到相应的提升.

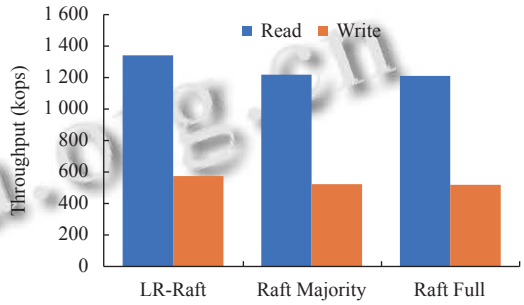


图 12 随机混合读写负载下 3 种模式的读写总吞吐量

如图 13 所示, 3 种模式在 4 KB 随机写的工作负载下, 分别设置 fio 的 nums_job 为 7, 6, 5, 4 这 4 种并发 I/O 请求数量 (io_depth 为 5) 进行性能测试. 结果表明, LR-Raft 的写操作吞吐量在 4 种并发 I/O 请求数量下比 Raft Majority 分别增加了 14.49%, 15.37%, 13.32% 和 10.68%, 比 Raft Full 分别增加了 21.46%, 19.84%, 20.52% 和 25.44%. 在 4 KB 随机写的工作负载下, LR-Raft 在吞吐量的表现上仍优于其他两种模式.

3.2 读写时延测试实验

实验分别评估了 3 种模式在 4 KB 随机混合读写

工作负载下的平均、P90、P95、P99 的读写时延. 图 14 显示了 3 种模式的写时延, LR-Raft 模式较 Raft Majority 和 Raft Full 模式, 在平均写时延下分别降低了 14.97% 和 29.02%, P99 写时延下分别降低了 30.32% 和 67.01%. LR-Raft 因其减少了 1/3 的 Raft Apply 操作, 且日志副本所在存储节点负载较低, 写入速度较快, 其相较于 Raft Majority 仍有不少的性能提升. Raft Full 在三副本冗余策略下, 需要等待最慢的副本写成功才能返回给客户端写操作完成, 故其写时延最大.

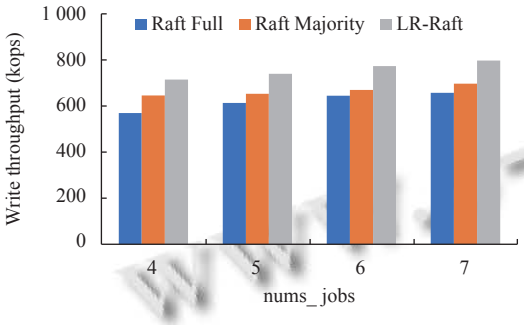


图 13 随机读负载下 3 种模式的写操作吞吐量

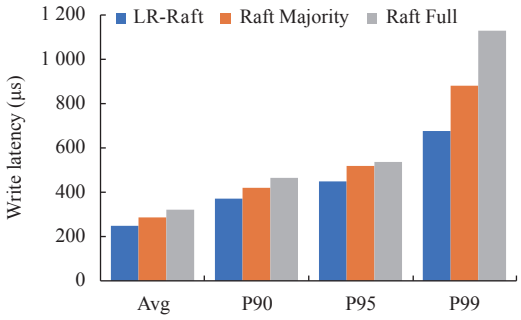


图 14 随机混合读写负载下 3 种模式的写时延

图 15 显示了 3 种模式的读时延, LR-Raft 模式较 Raft Majority 在平均、P90、P95 和 P99 时延分别降低了 6.65%、5.73%、11.21% 和 30.33%. 在 P99 时延上的降幅最为明显. 实验结果显示, LR-Raft 和 Raft Majority 比 Raft Full 模式在平均时延上分别降低了 7.98% 和 1.25%, 这得益于 Raft Full 模式的写模式处理, 不同于 LR-Raft 和 Raft Majority 模式, 其所有副本都可以处理读操作, 所以其读时延更低.

3.3 拔盘性能测试实验

在同一存储节点上分别拔出 1–4 块硬盘模拟硬盘故障. 每次拔出硬盘后, Raft 算法在该节点的其他硬盘上创建正常副本, LR-Raft 算法在该节点的其他硬盘上创建日志副本. 待到新创建的副本正常工作后, 开始进行 4 KB 随机写的性能测试. 如图 16 所示, 分别在拔出 1–4 块盘的情况下, LR-Raft 比 Raft 的写操作吞吐量分别增加了 1.31%, 1.77%, 16.43% 和 42.86%.

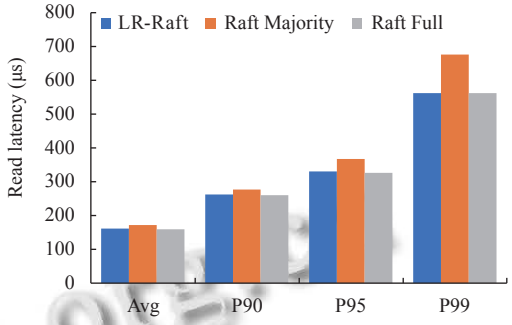


图 15 随机混合读写负载下 3 种模式的读时延

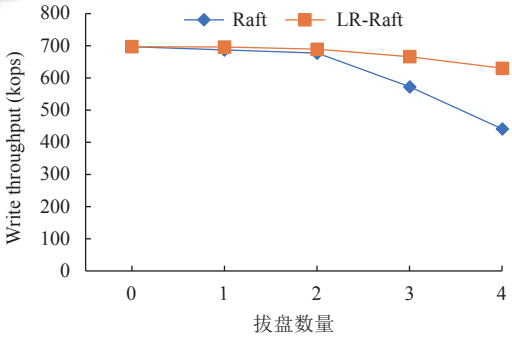


图 16 模拟拔盘随机写吞吐量

图 17 显示分别在拔出 1–4 块盘的情况下, LR-Raft 比 Raft 的写时延降低了 1.28%, 1.64%, 13.97% 和 30.07%. 实验结果显示, 在同一节点拔出多块盘后, LR-Raft 创建日志副本与 Raft 创建正常副本两种方案相比, 前者更加接近拔盘前的性能, 尤其是在存储节点近一半的硬盘被拔出的情况下.

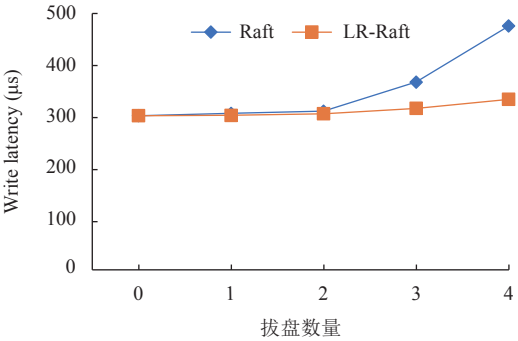


图 17 模拟拔盘随机写时延

3.4 总结

由于 LR-Raft 中日志副本没有状态机, 所以比其他副本少了一次 Apply 操作, 减少了硬盘的负载. 所以

无论是初始时就在副本组中加入日志副本, 还是在存在故障硬盘后添加日志副本, LR-Raft 在随机混合读写和随机写两种工作负载下都有不错的吞吐量提升, 平均时延和 P99 时延都有稳定的下降.

4 结束语

本文提出了一种增强的共识算法 LR-Raft. LR-Raft 打破了 Raft 对副本数据的限制, 引入了无状态机的日志副本的概念, 在存在故障副本时, 能快速加入集群参与共识, 增强了集群的容错能力, 提高了集群可用性. 在此基础上, 提出了日志压缩优化方案, 以解决引入日志副本后可能丢失数据的问题. 实验结果表明, 在发生故障的情况下, LR-Raft 较 Raft 算法有更低的读写时延和更高的吞吐量.

参考文献

1 Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC 14). Philadelphia: USENIX Association, 2014. 305–320.

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

3 Wang ZG, Zhao CG, Mu S, *et al*. On the parallels between Paxos and Raft, and how to port optimizations. Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. Toronto: ACM, 2019. 445–454.

4 Howard H, Mortier R. Paxos vs Raft: Have we reached consensus on distributed consensus? Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data. Heraklion: ACM, 2020. 8.

5 Cao W, Liu ZJ, Wang P, *et al*. PolarFS: An ultra-low latency and failure resilient distributed file system for shared storage cloud database. Proceedings of the VLDB Endowment, 2018, 11(12): 1849–1862. [doi: 10.14778/3229863.3229872]

6 谷晓松, 魏恒峰, 乔磊, 等. 支持乱序执行的 Raft 协议. 软件学报, 2021, 32(6): 1748–1778. [doi: 10.13328/j.cnki.jos.006248]

7 王进, 李博涵, 吴佳骏, 等. 支持日志乱序提交的分布式一致性协议. 浙江大学学报 (工学版), 2023, 57(2): 320–329.

8 Chandra TD, Griesemer R, Redstone J. Paxos made live: An engineering perspective. Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing. Portland: ACM, 2007. 398–407.

9 Guo JW, Cai P, Qian WN, *et al*. Accurate and efficient follower log repair for Raft-replicated database systems. Frontiers of Computer Science, 2021, 15(2): 152605. [doi: 10.1007/s11704-019-8349-0]

10 Arora V, Mittal T, Agrawal D, *et al*. Leader or majority: Why have one when you can have both? Improving read scalability in Raft-like consensus protocols. Proceedings of the 9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17). Santa Clara: USENIX Association, 2017. 14.

11 Wang YY, Wang ZK, Chai YP, *et al*. Rethink the linearizability constraints of Raft for distributed systems. IEEE Transactions on Knowledge and Data Engineering, 2023, 35(11): 11815–11829. [doi: 10.1109/TKDE.2023.3235399]

12 Wang YY, Wang ZK, Chai YP, *et al*. Rethink the linearizability constraints of Raft for distributed key-value stores. Proceedings of the 37th IEEE International Conference on Data Engineering (ICDE). Chania: IEEE, 2021. 1877–1882.

13 Wang YY, Chai YP. vRaft: Accelerating the distributed consensus under virtualized environments. Proceedings of the 26th International Conference on Database Systems for Advanced Applications. Taipei: Springer, 2021. 53–70.

14 Pâris JF, Long DDE. Pirogue, a lighter dynamic version of the Raft distributed consensus algorithm. Proceedings of the 34th IEEE International Performance Computing and Communications Conference (IPCCC). Nanjing: IEEE, 2015. 1–8.

15 Huang DX, Liu Q, Cui Q, *et al*. TiDB: A Raft-based HTAP database. Proceedings of the VLDB Endowment, 2020, 13(12): 3072–3084. [doi: 10.14778/3415478.3415535]

16 Weil SA, Brandt SA, Miller EL, *et al*. Ceph: A scalable, high-performance distributed file system. Proceedings of the 7th Symposium on Operating Systems Design and Implementation. Seattle: USENIX Association, 2006. 307–320.

17 Aghayev A, Weil S, Kuchnik M, *et al*. File systems unfit as distributed storage backends: Lessons from 10 years of Ceph evolution. Proceedings of the 27th ACM Symposium on Operating Systems Principles. Huntsville: ACM, 2019. 353–369.

18 fio. https://github.com/axboe/fio. [2023-10-20].

(校对责编: 孙君艳)