庖丁解 Ceph 之 Paxos - 简书

Ceph Monitor 作为 Ceph 服务中的元信息管理角色，肩负着提供高可用的集群配置的维护及提供责任。Ceph 选择了实现自己的 Multi-Paxos 版本来保证 Monitor 集群对外提供一致性的服务。Ceph Multi-Paxos 将上层的元数据修改当成一次提交扩散到整个集群，Ceph 中简单的用 Paxos 来指代 Multi-Paxos，我们也沿用这一指代。本文将介绍 Ceph Paxos 的算法细节，讨论其一致性选择，最后简略的介绍代码实现。本文的大部分信息来源于 Ceph Monitor 相关源码，若有偏颇或谬误，敬请指正。

算法介绍

概览

Paxos 节点与 Monitor 节点绑定，每个 Monitor 启动一个 Paxos。当有大多数的 Paxos 节点存活时集群可以运行，正常提供服务的过程中，一个节点做为 Leader 角色，其余为 Peon 角色。只有 Leader 可以发起提案，所有 Peon 角色根据本地历史选择接受或拒绝 Leader 的提案，并向 Leader 回复结果。Leader 统计并提交超过半数 Paxos 节点接受的提案。

Leader 发起提案及 Peon 接受提案时都会写入本地 Log，被提交的 Log 会最终写入 DB，写入 DB 的提案才最终可见。实现中用同一个 DB 实例承载 Log 和最终数据的存储，并用命名空间进行区分。

除去上面提到的 Leader 及 Peon 外，Paxos 节点还有可能处于 Probing、Synchronizing、Election 三种状态之一，如 Figure 1 所示。其中，Election 用来选举新的 Leader，Probing 用来发现并更新集群节点信息，同时发现 Paxos 节点之间的数据差异，并在 Synchronizing 状态中进行数据的追齐。当 Membership 发生变化，发生消息超时或 lease 超时后节点会通过 boostrap 进入 Probing 状态，并向其他节点广播 prob 消息，所有收到 prob 消息的非 prob 或 synchronizing 节点会同样回到 Probing 状态。Probing 状态收到过半数的对 Members 的认可后进入 Election 状态。同时 Probing 中发现数据差距过大的节点会进入 synchronizing 状态进行同步或部分同步。更多的内容会在稍后的 Recovery，Membership 及 Log Compaction 中介绍。

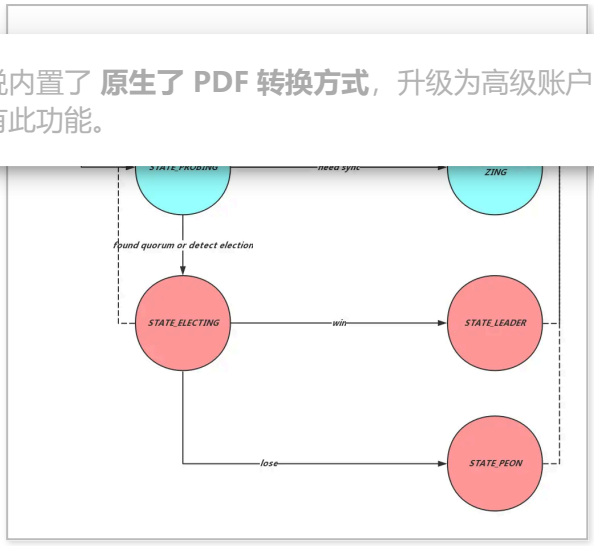

Figure 1

Leader 会向所有 Peon 发送 Lease 消息，收到 Lease 的 Peon 在租约时间内可以直接以本地数据提供 Paxos 读服务，来分担 Leader 的只读请求压力。Lease 过期的 Peon 会退回 Probing 状态，之后通过新一轮的选举产生新的 Leader。

Leader 会选择当前集群中最大且唯一的 Propose Num，简称 Pn，每次新 Leader 会首先将自己的 Pn 增加，并用来标记自己作为 Leader 的阶段，作为 Ceph Paxos 算法中的逻辑时钟（Logical Clock）。同时，每个提案会被指派一个全局唯一且单调递增的 version，实现中作为 Log 的索引位置。Pn 及 Version 会随着 Paxos 之间的消息通信进行传递，供对方判断消息及发起消息的 Leader 的新旧。Paxos 节点会将当前自己提交的最大提案的 version 号同 Log 一起持久化供之后的恢复使用。

常规过程（Normal Case）

常规服务状态下存在一个唯一的 Leader Quorum，Leader 将每个写请求被封装成一个新的提案发送给 Quorum 中的每个节点，其过程如下，注意这里的 Quorum 固定：

- Leader 将提案追加在本地 Log，并向 Quorum 中的所有节点发送 **begin** 消息，消息中携带提案值、Pn 及指向前一条提案 version 的 last_commit；
- Peon 收到 begin 消息，如果 accept 过更高的 pn 则忽略，否则提案写入本地 Log 并返回 **accept** 消息。同时 Peon 会将当前的 lease 过期掉，在下一次收到 lease 前不再提供服务；
- Leader 收到**全部** Quorum 的 accept 后进行 commit。将 Log 项在本地 DB 执行，返回调用方并向所有 Quorum 节点发送 **commit** 消息；
- Peon 收到 commit 消息同样在本地 DB 执行，完成 commit；
- Leader 追加 **lease** 消息将整个集群带入到 active 状态。

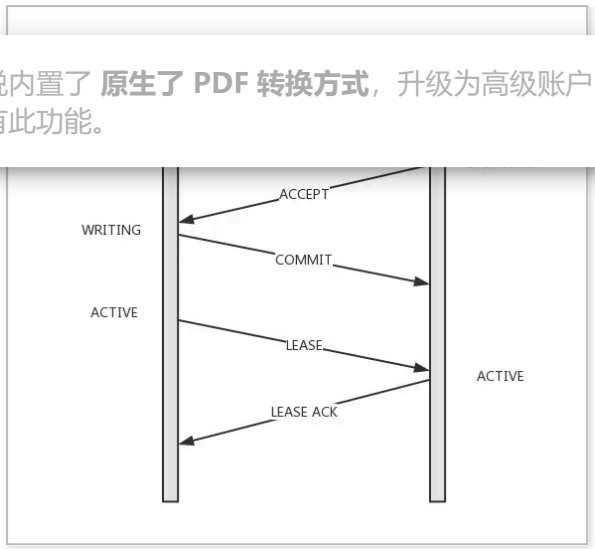

Figure 2

选主（Leader Election）

Peon 的 Lease 超时或 Leader 任何消息超时都会将整个集群带回到 Probing 状态，整个集群确定新的 Members 并最终进入 Election 状态进行选主。每个节点会在本地维护并在通信中交互选主轮次编号 election_epoch，election_epoch 单调递增，会在开始选主和选主结束时都加一，因此可以根据其奇偶来判断是否在选主轮次，选主过程如下：

- 将 election_epoch 加 1，向 Monmap 中的所有其他节点发送 **Propose** 消息；
- 收到 Propose 消息的节点进入 election 状态并仅对更新的 election_epoch 且 Rank 值大于自己的消息答复 **Ack**。这里的 Rank 简单的由 ip 大小决定。每个节点在每个 election_epoch 仅做一次 Ack，这就

但还是给的 Leader，它获得了大多数

- 发送 Propose 的节点统计收到的 Ack 数，超时时间内收到 Monmap 中大多数的 ack 后可进入 victory 过程，这些发送 ack 的节点形成 Quorum，election_epoch 加 1，结束 Election 阶段并向 Quorum 中所有节点发送 **Victory** 消息，并告知自己的 epoch 及当前 Quorum，之后进入 Leader 状态；

- 收到 VICTORY 消息的节点完成 Election，进入 Peon 状态；

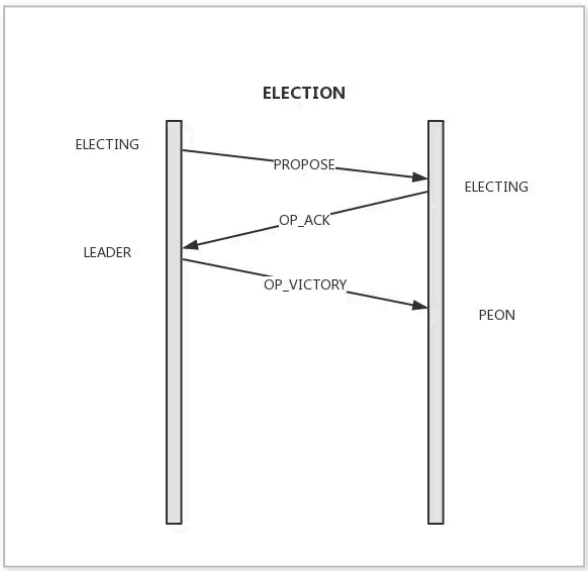

Figure 3

恢复（Recovery）

经过了上述的选主阶段，便确定了 Leader，Peon 角色以及 Quorum 成员。但由于 Election 阶段的选主策略，新的 Leader 并不一定掌握完整的 commited

数据，因此在真正的开始一致性请求之

息的是，Ceph Paxos 限制提案的发起按 version 顺序，前一条提案被 commit 后才能发起后一条，也就是说 Recovery 的时候最多只能有一条 uncommitted 数据，这种做法虽然牺牲了性能，但却很大程度的简化了 Recovery 阶段及整个一致性算法的实现，而这种性能的牺牲可以由 Ceph 层的聚合提交而弥补。

- Leader 生成新的更大的新的 Pn，并通过 **collect** 消息发送给所有的 Peon;
- Peon 收到 collect 消息，仅当 Pn 大于自己已经 accept 的最大 Pn 时才接受。Peon 通过 **last** 消息返回自己比 Leader 多 commit 的日志信息，以及 uncommitted 数据;
- Leader 收到 last 消息，更新自己的 commit 数据，并将新的 commit 日志信息通过 **commit** 消息发送给所有需要更新的 Peon;
- 当接收到所有 Peon accept 的 last 消息后，如果发现集群有 uncommitted 数据，则先对该提案重新进行提交，否则向 Peon 发送 **lease** 消息刷新其 Lease;

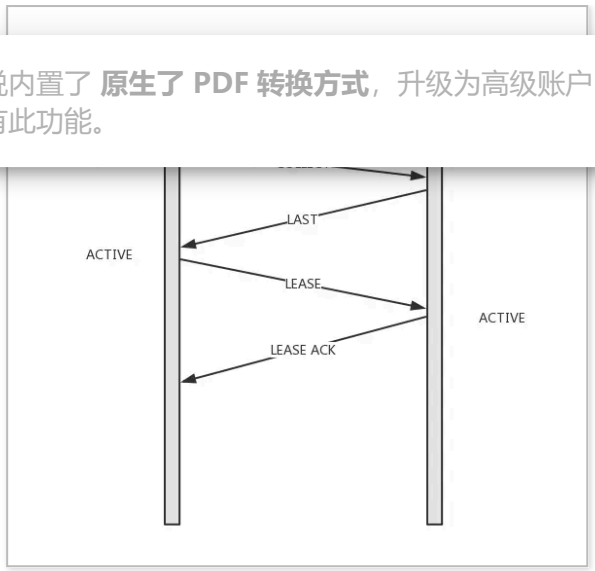

Figure 4

可以看出，当 Leader 和 Peon 之间的距离差距较大时，拉取并重放 Log 的时间会很长，因此在开始选主之前，Ceph Monitor 首先通过如 Figure 1 所示的 Synchronizing 来将所有参与 Paxos 节点的日志信息差距缩小到足够小的区间，这个长度由 paxos_max_join_drift 进行配置，默认为 10。Synchronizing 过程中 Monitor 节点会根据 Prob 过程中发现的 commit 位置之间的差异进行数据的请求和接收。

成员变化（Membership Change)

Ceph Paxos 的成员信息记录在 Monmap 中，Monmap 可以从配置文件中初始化，也可以在后期加入或删除。Ceph Monitor 中引入了 Probing 阶段来实现 Memebership 的变化，节点启动、新节点加入、Paxos 各个阶段发生超时、发现新的 prob 消息、Monmap 信息发生变化

时都会通过 bootstrap 进入到 Probing 状态，prob 消息来探测对方存在来生成并交互 Monmap 信息。而这个过程中整个 Paxos 集群是停止对外提供服务的。

日志截断（Log compaction）

通过上面的描述已经知道，Ceph Paxos 的 Log 中记录了每个提案的内容，这些内容本质是对节点状态机的一组原子操作。随着集群的正常提供服务，Log 数据会不断的增加，而过多的 Log 不仅会占用存储资源，同时会增加日志回放的时间。所以 Ceph 中引入了一套机制来删除旧的 Log 数据。每次提案 commit 成功后，Monitor 都会检查当前的 Log 数据量，超过某一配置值后便会进行截断（trim），这个保留的长度由 paxos_min 进行控制，默认是 500。Monitor 中用 first_commited 来标识当前保留的最早的 Log 的 version 号，trim 过程简单地删除一定量 Log 并修改 first_commited 内容，需要数据恢复时，如果需要小于 first_committed 的内容，则会在如 Figure 1 所示的 Synchronizing 过程中进行数据的全同步。

一致性选择

1，State Machine System

Ceph monitor 选择了 State Machine System。Log 中存储的内容以及 Paxos 节点之间的交互数据都是像 Put，Erase，Compacat 这样的幂等操作；而在 commit 后才会真正写入到状态机。

2，每次只能一条提案

Ceph Paxos 的提案发起严格有序，并且只有前一条 Log commit 后才会发起新的提案，这也就保证集群最多只能有一条 uncommitted 的提案，这也就简化了 Recovery 的实现逻辑。能这样做也是由于 Ceph Monitor 上层的聚合提交等减少对一致性协议执行的机制大大降低了 Ceph Paxos 对性能的要求。

3，固定的（Designated） Quorum

对 Paxos 算法来说，无论选主过程还是正常的访问过程，都需要保证有大多数节点（Quorum）的成功，通常这个 Quorum 每次是不固定的，而 Ceph Paxos 选择在选主成功后就确定的生成一个 Quorum 集合，之后的所有操作，都只向这节点发出，并等待集合内所有节点的答复，任何的超时都会重新通过 bootstrap 过程退回到 Probing 状态。猜测这里更多的是针对实现复杂度的考虑。

4，双向的 Recovery 方向

由于 Ceph Paxos 的选主策略仅根据节点 Leader 的节点可能并没有最新的数据，因此在提供服务前 Leader 需要先在 Recovery 阶段恢复自己和集群的数据，Recovery 的数据方向包括从 Peon 到 Leader 和 Leader 到 Peon 两个方向。

5，使用 Lease 优化只读请求

Ceph Paxos 引入了 Lease 机制来支持 Peon 分担 Leader 压力，在 Lease 有效的时间内，Peon 可以使用本地数据来处理只读请求；Peon 在接收到一个新的提案开始是会先取消本地的 Lease，提案 commit 后或 Leader 的 Lease 超时后 Leader 会刷新所有 Peon 的 Lease；

6，Leader Peon 同时检测发起新的 Election

Leader 和 Peon 之间的 Lease 消息同时承担了存活检测的任务，这个检测是双向的，Leader 长时间收不到某个 Peon 的 Lease Ack，或者 Peon Lease 超时后依然没有收到来自 Leader 的刷新，都会触发新一轮的 Election。

代码概述

Figure 5

Ceph Monitor 中 Paxos 相关的内容散布在不同的类型中，主要包括 Monitor，Election，Paxos 几个类：

Monitor 中维护了如 Figure 1 中的节点状态转换，并且在不同阶段调度 Election 及 Paxos 中的相关功能。同时 Monitor 也承担着为其他类提供全局数据的功能。Monitor 通过 boostrap 方法发起 Probing 生成或修改 Monmap，并发现节点之间的数据差异，当差异较大时会调用 start_sync 进入 Synchronizing 过程。

Election 主要负责选主过程，Monitor 会在 Probing 及 Synchronizing 过程结束后通过 call_election 开启选主逻辑。Election 选主结束后分别调用 Monitor 的 win_election 和 lose_election 将控制权交还给 Monitor。win_election 和 lose_election 中，Monitor 完成节点的状态变化，并分别调用 Paxos 的 leader_init 和 peon_init 方法开始 Paxos 作为 Leader 或者 Peon 的逻辑。Paxos 由 Leader 发起 Recovery 过程，之后进入 Active 状态准备提供服务。

上层的写操作会通过 **Paxos** 的

Paxos 会调用 Monitor 的 refresh_from_paxos 告知上层，同时，上层也可以向 Paxos 的不同阶段注册回调函数 finish_context 来完成上层逻辑，如 pending_finishers 或 committing_finishers 回调队列。

参考

RADOS: A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters

CEPH SOURCE CODE

choices in consensus algorithm

Vive La Diffe ŕence:Paxos vs. Viewstamped Replication vs. Zab

Paxos made simple

[Paxos made live] (https://static.googleusercontent.com/media/research.google.com/zh-CN//archive/)

全文完