CCeph分布式协议设计

# 1 概述

系统以对象为数据单位，支持对于对象的读、写和删除操作。其中读写操作可以指定偏移量和长度，即支持随机访问。暂时不考虑严格意义上的Append操作。

## 1.1 设计目标

系统的核心设计目标是：**所有IO请求不等待任何可用性设计范围内的超时。**

即假设数据采用三副本的容灾设计，可用性为允许一个副本宕机，则对于任何单点故障，请求的延迟（Latency）不受影响。

## 1.2 关键设计

系统的关键设计包括：

* 读写采用NWR一致性模型，其中N为存储池级别的参数，WR为客户端级别的参数。
* 为了保证数据一致性（不一定是强一致性），强制规定W必须大于N/2，即写操作必须完成半数以上的冗余，才能视为完成。
* 系统存在一个轻量级的中心系统，负责发布集群的状态（类似Ceph的mon或者Zookeeper），主要是集群硬件的拓扑结构（类似OSDMap）。任何硬件的变化通过集群状态发布机制实现在整个集群内部的传播。
* 集群状态（OSDMap）是数据分布的依据。每一个数据副本基于当前的状态，被一致性哈希（CRUSH）到一组osd上，这组osd是有序的。
* 集群内的每一个对象都包含了一个版本号version，每次对象的修改会使得版本号数量加一，设计中认为版本号是无限大（不会回绕）的。

## 1.3 一致性

当W+R > N时，系统表现为强一致性 ，即如果系统向客户端报告写成功，则该客户端（或者采用相同NWR规则的其它客户端）可以读到最新的数据。典型的情况是W=3，N=R=2的情况。

当W+R <= N时，系统表现为最终一致性，即最新写入的数据可能在一段时间内无法读取成功。考虑以下场景：W=2，N=2，R=1的规则下，假设三个数据副本分别是a，b，c，数据x初始版本为v1。客户端c1执行写操作将x更新为v2，假设写操作在a，b上完成，而尚未在c上完成，客户端c1认为x的当前版本已经v2。但是此时假设客户端c1，或者其他客户端c2执行R=1的读操作，而恰好读取的是c副本，则仍然可能读取到脏数据，即x的v1版本。

# 2 流程

## 2.1 写流程

写流程基本采用PAXOS，即各个副本之间通过投票达成一致，当节点拿到半数以上的投票时，说明该写请求已经生效。

任何Rep拿到半数以上请求，都可以认为该请求已经被所有Rep认可，即可向客户端返回Ack。所以客户端收到了任意Rep的请求成功的响应，则客户端都可以认为请求已完成。（通常状态）

**第一步：请求发送**



客户端向每一个副本发送完全相同的请求，包含以下内容：

* client\_id：每个客户端的唯一ID，客户端连接到集群时，从mon获取。
* request\_id：客户端每个请求的ID，由客户端编号，不重复。
* oid：要写入到对象ID
* offset、length、data、crc：请求内容。

每个请求的ClientID+RequestID可以唯一的区分这个请求。

**第二步：副本投票**



投票的目的是确认数据修改操作得到了半数以上的Rep的认可。具体的流程又可细分为：（1）Preapre、（3）Vote和（3）Apply

（1）Preapre

当每一个Rep收到请求时，都需要基于要修改的对象的版本，生成新的数据版本号。

假设，客户端c1的请求q1，要修改o1，该请求可以描述为：

(client\_id=c1, req\_id=q1, oid=o1)

则对于Rep，该请求还会有一个oid对应的新版本号，因此请求被标示为：

(client\_id=c1, req\_id=q1, oid=o1，v=1)

版本号是基于oid的，是一个递增的整数。实现中采用int64，视为无穷大，考虑到性能，所有对象的当前版本号全部需要缓存到内存中（占内存很小）。

当编号完成后，Rep会将请求，包含其数据版本号和请求的所有内容，记录到本地的日志中，这是一条Prepare日志。

此时数据并不能被访问到。

（2）Vote

当prepare日志被成功的记录到本地后，Rep会发起投票。即向其它Rep发送请求的情况，包括（ClientID、RequestID、oid、v）。注意Vote并不需要互相发送数据，最主要的信息是该Rep对某一个请求的版本号v，投票的主要目的是在Rep之间达成对数据变更的一致的认知。

当任意Rep收到半数（包含本身）以上的相同的Vote消息后，则可以确认请求已经的状态已经达成一致，则可以进入Apply过程。

如果某一个Rep发现o的版本号并不相同，怎么处理？

（3）Apply

Apply流程会向磁盘写入一条Apply的日志，然后修改内存缓存以反映数据的变化。

**第三步：回复响应**



当一个Rep收到半数以上的投票并完成本地的应用操作（Apply）后，即可向客户端返回请求完成。

对于客户端来说，写请求的返回值满足以下约束：

* 如果客户端收到了>=W个Rep的请求成功的响应，则客户端都可以认为请求已完成（通常状态）。
* 如果客户端收到了任意Rep的请求失败的响应，则该请求必然已经失败，请求的数据没有发生任何的修改。
* 如果客户端没有收到任何响应，则数据的状态未知。

客户端在收到失败请求时，需要重试。（完整的客户端逻辑是什么样子的？）

## 2.2 读流程

## 2.3 删除流程

删除被视作一条特殊的写操作，其流程与写操作完全相同。

# 3 故障

节点宕机

节点超时/网络抖动

网络闪断

包重复

并发请求/乱序

故障节点的识别与重新负载均衡

全节点宕机

# 4 性能

# 5 变更