参考资料：

https://zhuanlan.zhihu.com/p/143482529

Ceph分布式存储系统基于底层的RADOS分布式对象存储系统来完成数据的复制，数据恢复和数据的动态扩容和缩容。本文针对RADOS的数据复制算法和流行的Raft算法做对比分析，

试图分析其优劣缺点，提出相关的优化建议。

1、Rados的数据复制算法

1.1、数据写操作流程

Rados的基本概念是PG，PG是数据复制的基本单位。以三副本为例：一个PG分布在3个OSD上。一个PG里可以保存多个Object，每个Object最大为4M的文件。Rados的写操作如下：

1）客户端发送写请求到PG对应的主OSD

2）主OSD发送请求到2个Slave OSD

3）主OSD和Slave OSD同时写入底层的对象存储引擎ObjectStore（FileStore或者BlueStore存储引擎）。底层的ObjectStore完成日志的提交并完成数据的apply后返回。（Bluestore比Filestore有一些优化：Bluestore的new write不写日志；Bluestore提交后就返回，并没有等apply操作完成。）

4）主OSD等待所有的OSD写成功(日志提交并完成apply)后返回给客户端。

1.2、日志

Rados里有2种不同的日志：

1）Pg log日志：记录了该pg内所有对象的所有更新操作，只记录操作的元数据：操作类型，操作的对象ID和对象的版本号

2）底层对象存储引擎（object store）的日志：对象存储的日志作用是确保每次object store上写操作的原子性

1.3、故障处理

Rados的写操作，需要三副本都写成功，才能给客户端成功的应答。这个要求比较严格。两种故障的处理：

1）临时的故障：对于master osd的临时故障，客户端需要重试一定的次数。对于slave osd的临时故障，由于rados需要所有osd都应答成功：IO会卡主一定的时间，等待slave osd重试。

2）如果是永久故障：IO 都会卡主一定的时间：mon通过心跳消息来判断该osd永久故障，mon在osdmap中标记该osd为down状态，并且同步该osdmap信息该相关的osd，该pg开始重新peering过程。

1.4、数据恢复

Rados的数据恢复分成两个阶段：

1）pg的peering过程，使pg达到peered状态。 该过程只是使pg完成数据一致性协调过程，并没有完成数据的恢复。

2）pg的recovery（和 backfill）过程，使pg达到clean状态，完成数据的恢复过程。

标记着两个状态的值为：

last\_epoch\_started: pg 完成 peering的 epoch值。

ast\_epoch\_clean：pg完成 recover 和 backfill时的值

在rados里，当一个OSD失效后，pg 会发起Peering过程来完成数据的一致性协调（而不是完成恢复），其主要过程如下：

1）GetInfo阶段，获取各个副本OSD上该PG的信息，pg\_info\_t的信息：

日志的头部（最新更新的日志）：last update

日志的尾部（最旧的日志记录）：log\_tail

past\_interval阶段是指一个PG的列表没有发生变化的阶段。换句话说，就是一个past\_interval标记了一个PG的member列表变化了一次。从last\_epoch\_started阶段，也就是从上次peering完成的epoch阶段开始：检查每个 past interval阶段，是否有足够的osd存活。

2）GetLog阶段

1、选取权威日志：在getinfo阶段获取相关的日志信息后，需要选取一个拥有权威日志的OSD，

权威日志的定义：

1）日志最新更新的osd

2）优先选择日志最长的osd

3）优先选择当前主osd

2、主osd获取权威日志：当前主osd如果不是拥有权威日志的OSD，需要参照权威日志，拷贝自己缺失的日志，从而补全自己的日志，使自己的日志成为权威日志。（需要确保当前的主osd是可通过日志来恢复的，也就是当前主osd的日志和权威日志重叠。否则需要申请临时主osd）

3、在主osd对照权威日志补全日志的过程中，也同时计算出主osd上缺失的对象信息。

3）GetMissing阶段

1、此时主osd上已经拥有了权威日志

2、根据主osd上的权威日志，拉取各个slave osd上日志信息，并根据这些信息来计算该osd上缺失的对象。（当然这些osd是可以recovery的osd，而不是需要backfill的osd信息）。

至此，ceph的pg peering完成了整个过程:

1)主osd清楚了主osd和slave osd上所有缺失的和不一致的osd的信息。

2)对外可以提供读写请求。

3)在提供读写请求的过程中，如果有osd缺失，就需要阻塞，恢复流程优先恢复各个osd上缺失的对象，然后完成读写操作。

恢复流程是在后台，根据missing列表完成相应的缺失对象的修复过程。后台的修复过程不在IO的关键路径上，但可能会影响ceph正常业务io的读写性能。

首先：缺失的对象是临时阻塞的，优先完成恢复后再进行IO操作。

其次：后台修复的IO会占用整个系统的资源，影响业务IO的性能。

综上所述的ceph恢复的流程可以看到：

1）Ceph是基于日志来恢复的，日志是连续的记录。

2（Ceph的日志是基于pg的粗粒度的元数据日志记录。（基于4M大小的object，只记录对象的操作元数据，而没有操作的具体数据。）

3）Ceph的副本协商一致和数据一致（副本完成恢复，所有副本数据一致）是2个阶段分开完成。

2、Raft数据一致算法

Raft算法也是基于日志的数据一致。Raft的有raft group的概念，和rados的pg的概念，其意义是一样的，都是复制组的概念。每个Raft group里有日志和raft状态机。

2.1、数据写操作请求

Raft里有Leader和Slave的概念，对应rados里的master和slave。其读写流程具体如下：

1）客户端把请求发送给raft group中的Leader节点

2）Leader把请求以日志的形式发送（广播）给2个slave（以3副本为例），Leader和 slave都把日志持久化保存在自己的日志磁盘文件中，然后返回调用方。

3）Leader收到大多数（包括自己）的请求ack成功应答后：

1、Leader本地提交日志，并把日志apply到raft状态机中。

2、应答客户端写操作成功。

4）Leader给slave发日志commit的请求。（这一步一般和下次日志广播的请求合并发送）。

2.2、Raft的日志

Raft的日志记录该raft group中所有的更新操作的元数据和数据记录。

2.3、Raft的故障处理

如果raft的Leader节点出现了临时故障，客户端需要重试机制解决。如果raft的slave节点出现了临时故障，并不会影响写操作的性能，由于raft的写操作只要求大多数节点成功应答就可以。这一点上，

raft放松了限制，提高了性能，但是牺牲了可用性：3副本中只允许1个副本故障。

2.4、Raft的恢复流程

Raft在恢复的过程中，需要选举流程，在选举的流程中，选择Leader的标准是：

1）日志最新

2）日志最长

raft的leader选举的标准和rados的恢复选取权威日志的标准完全一致。raft在完成选举后，需要新的Leader发送一个空的更新操作。slave节点会检查本次操作和日志是否连续，如果不连续，就发起恢复

操作：对照Leader上的日志，完成slave节点日志的处理：缺失日志的补齐，多出日志的清理。当空操作完成后，raft开始对外提供服务。由以上分析可以知：

1）raft的恢复完成后才对外提供服务的，和rados的两个阶段的处理相比比较简单

2）raft的恢复是基于详细的日志，和rados相比比较高效。

3、总结

1）都是基于日志完成副本数据的一致性保障的，每个节点都需要一个完整，连续的日志记录来保证。

2）Rados不需要leader选举，其master是通过crush算法确定的。Raft需要一个Leader选举的过程，这个过程可能需要一定的时间（可能多次选举才能成功）。

3）rados由于记录的pg log日志粒度比较粗，并且只记录操作的元数据，导致数据完成一致性恢复的时间可能比较久。Rados采用了2个步骤：peering完成只是pg完成了数据一致性的协商，pg clean状态才是rados的pg真正完成数据一致性的时间点。这导致在数据恢复阶段的状态机异常复杂，实现难度较大。

4）rados其实在底层的对象存储（Filestore或者Bluestore）都有详细的日志（操作的元数据和数据），而该日志是基于object store存储引擎的，而不是基于pg的日志。也就是说object store的日志里存放的是该osd上的所有pg的日志，具体日志的truncate也是本地自己完成的。Rados并没有利用object store的日志来完成恢复，而是利用pg log来恢复。

总之，raft的实现比较ceph简单很多。ceph可以充分利用object store的日志记录来恢复，提供恢复的效率，ceph可以去掉peering和recovery两个阶段，实现可以简化很多。