Ceph原理介绍

1      整体架构介绍

1.1      总体介绍

Ceph是一种开源的软件定义存储系统，诞生于2004年，最早致力于开发下一代高性能分布式文件系统的项目。Ceph可以部署在任意x86服务器上工作，具有良好的扩展性、兼容性和可靠性。它能对外提供文件系统服务(cephfs)、块服务(rbd)和对象存储服务(rgw)，是一种统一存储系统。Ceph架构支持海量数据存储，集群可以扩展至PB容量，系统本身无热点数据，数据寻址依赖计算而不是查找，并能做到数据状态自维护，数据自修复，是一款优秀的分布式存储系统。

1.2      整体架构

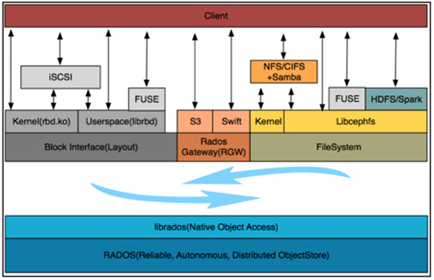


图 1 ceph全景

Ceph架构主要包含了：Rados集群，librados接口层，rgw、rbd和cephfs这三种存储服务。

Rados集群：Rados是Ceph系统的核心，包含了分布式集群管理和数据管理，集群的扩展性和高可用性是在这里体现的。主要的组件有monitor、osd和mds。Monitor是集群的关键服务，它保证了集群元数据的一致性和集群的可服务性。Osd是ceph的数据服务，它负责了业务数据的落盘，数据状态的监控，数据状态恢复，数据的迁移和恢复等流程。Mds是cephfs的元数据服务，维护文件系统的超级块信息，目录结构，文件信息等。一般如果不使用cephfs，是可以不用部署mds的。

librados接口层：统一封装的接口层，提供集群连接接口，pool创建接口，obj读写接口等，作为基础库提供给上层调用，比如librbd、libcephfs和librgw。第三方的应用可以直接调用librados对ceph做二次开发。

客户端：ceph客户端包括rbd、rgw、cephfs这三种类型，同时也包括librbd、libcephfs和librgw这些开发库。对外提供存储服务，比如rbd可以导出scsi块设备，cephfs可以mount到Linux主机上做为文件系统，也可以通过cifs/nfs导出为网络文件服务，rgw对外直接提供s3或者swift对象存储服务。

2      集群管理

2.1      Monitor

Monitor服务是ceph的核心服务之一，它主要作用是为整个集群提供全局的配置和系统信息。Ceph集群其实是体现在Monitor集群上的，Monitor是一个独立部署的服务，多个Monitor组成高可用集群。Monitor采用paxos算法保证集群数据的一致性，Monitor所管理的数据包括：

1、  Monitor Map：包括集群的fsid，所有的monitor的ip和端口，以及epoch

2、  OSD Map：包括集群的fsid，所有osd状态及监听地址，pool的信息及pg数等

3、  MDS Map：包括所有的mds服务的列表和状态，data pool和metadata pool

4、  PG Map：所有pg的信息，包括状态、version等

5、  CRUSH Map：一个树形结构，包括存储设备，故障域等

2.2      心跳管理

Monitor通过心跳这种方式来获得OSD服务的工作状态，根据这种状态的变化更新相应的位图。

首先是OSD之间会有心跳检查，OSD会检查osd\_heartbeat\_min\_peers个OSD服务，peer osd其实是指的osd相邻的osd服务，默认是10个。这些相邻的osd服务首先是包含同一个pg的osd列表，这个是从pg\_map上获取，如果得到的osd列表超过osd\_heartbeat\_min\_peers个就丢弃，不足的就补上当前osd周围状态up的osd进来。

检查过程是每osd\_heartbeat\_interval秒就检查一次peer端的osd，默认是6秒。如果peer端osd在osd\_heartbeat\_grace后没回复，默认20s，则标记这个osd状态为down。

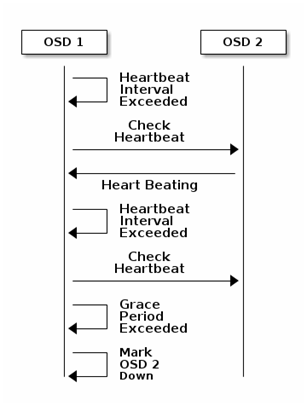


图 2 osd peer间心跳

在不同故障域之间osd的状态变化，monitor可能不会第一时间感知到，这里设置了不同故障域的reporter个数，默认是1

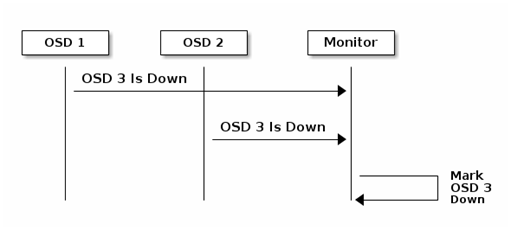


图 3 不同故障域的检查

图3所示，osd1和osd2分属于两个不同的故障域。

如果一个osd不能和peer内的所有osd做心跳了，则会向monitor申请最新的osd map。超时时间为osd\_mon\_heartbeat\_interval默认30s。

Monitor如果在mon\_osd\_report\_timeout内收不到osd的消息，则会设置osd为down，mon\_osd\_report\_timeout默认为900s。

正常情况下osd每隔osd\_mon\_report\_interval\_min，（最短间隔时间默认为5s）会向monitor报告一次事件，事件包括start刚启动、osd故障、up\_thru的变化等。osd向monitor报告的最长时间为osd\_mon\_report\_interval\_max，默认为120s。意思是说120s无论osd有没有状态变化都需要向monitor报告。

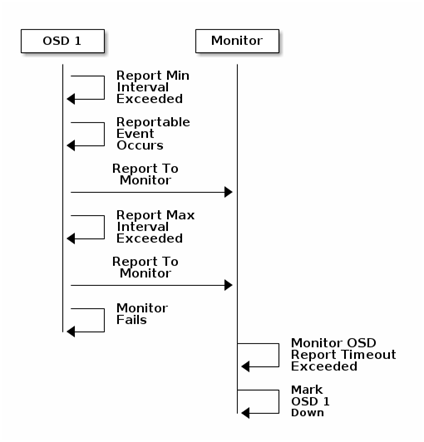


图 4 osd向monitor报告

3      数据读写

3.1      OSD

OSD（object storage daemon）负责Ceph集群的数据读写，同时也负责向Monitor报告监控的OSD的状态。管理数据的迁移，副本，数据平衡，数据恢复。Ceph集群数据管理的核心服务，通往磁盘的数据通道。

OSD主要包含了两个主要组成部分，一个PG，另一个是ObjectStore。PG（placement group）是Ceph的数据管理的基本逻辑单位，参与数据读写，数据迁移，数据恢复等和数据相关的全部流程。ObjectStore是负责向本地存储读写的模块，现在常用的是filestore和bluestore，当前onestor版本使用的是filestore。Filestore实际上是操作的本地文件系统，将业务最终写到本地磁盘的文件中，默认是使用xfs文件系统。Bluestore是直接操作的裸盘，数据直接通过BIO下到盘上，性能相较于filestore有较大提升，Ceph的最新版本默认采用是Bluestore。

3.2      读写流程

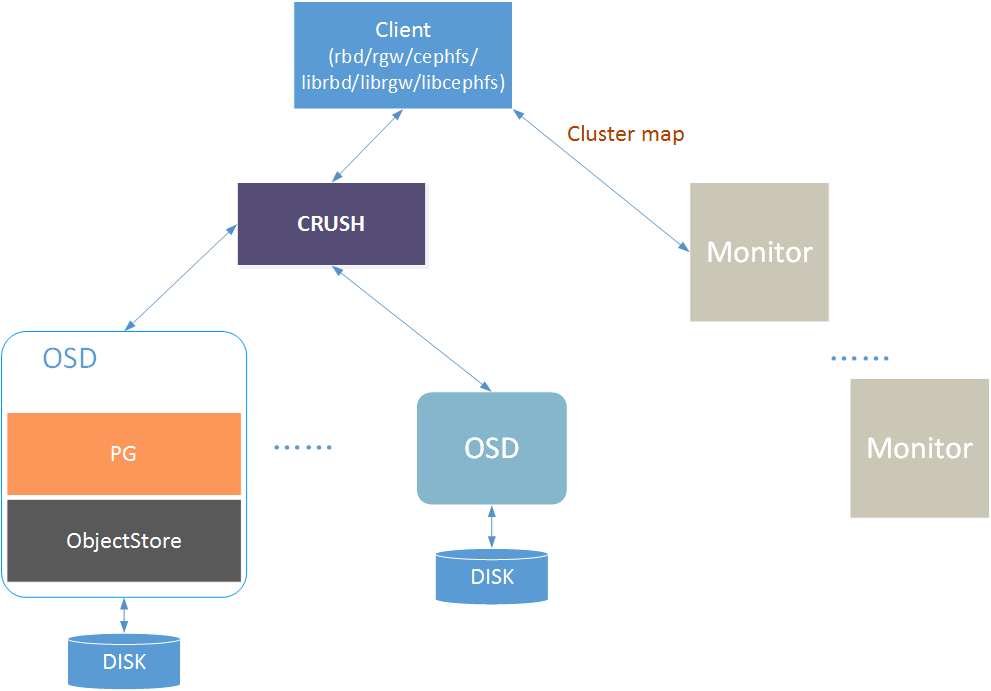


图 5 数据读写过程

数据要存储在Ceph上需要经历几个主要步骤：

1、  Client需要获得Monitor最新的cluster map，得到map后确定osd的信息

2、  对象到pg的hash，具体为对待写的object id做hash，得到的hash值，再对当前pool的pg\_num取模，最后得到该对象所在的pg id。

pg\_id = hash(object\_name) % pg\_num

然后再将pool id加到hash值之前，比如4.f1，4就是pool id，f1是上面算出的hash值，这两部分构成一个pg id。

3、  pg到OSD的映射，数据最终需落盘，所以pg得映射到一个OSD上。这个映射过程采用的Ceph自己独有的CRUSH算法，通过计算选择一个合适OSD。CRUSH本质是一种伪随机算法。找到OSD后，Client直接与OSD通信，建立网络连接，将数据发送到OSD服务处理。

4、  OSD最后把收到的数据投递给ObjectStore，由它完成向本地存储写入的过程。同时完成主从OSD的数据落盘。

OSD的数据写入过程满足数据的强一致性，待所有数据落盘才向上返回。

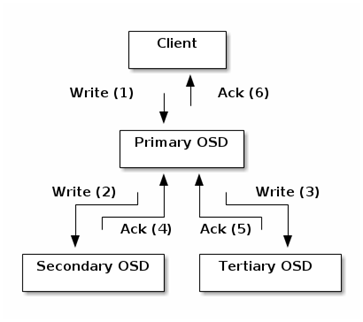


图 6 主从OSD写入过程

1、  Client先将写请求发到主OSD上。

2、  主OSD在收到Client请求后，立即向从OSD发送写请求，同时也写入主OSD的本地存储上。

3、  主OSD收到从OSD写入成功的ack，再确认自己处理正确，最后再返给Client写完成ack。写过程中必须等到所有的OSD的写成功返回才能向Client返回写操作成功的消息。

3.3      POOL和PG

Pool是一种逻辑上存储资源池，Ceph对外提供的存储服务，资源都是从Pool中划分提供的。Pool分两种一种是副本模式，一种是纠删码模式。一个Pool是由许多PG构成的。Pool的信息保存在osd map中。

PG是对象数据的集合，同一个集合内的对象应用相同的存储策略。比如对象的副本都会存放到相同的OSD列表中。一个对象只属于一个PG，一个PG包含多个对象，一个PG会存放到多个OSD上，一个OSD会承载多个PG。

图7表示了一个pool是双副本，对象t1是怎么分布到pg 1.3e的。

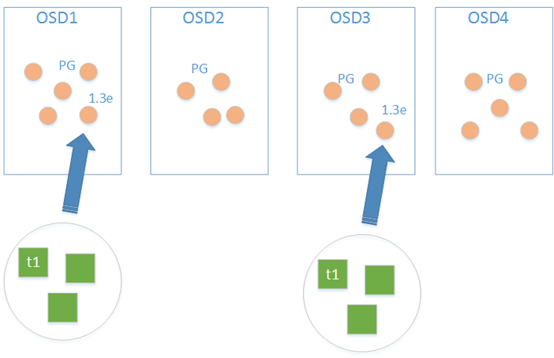


图 7 pg与osd对应关系

3.4      CRUSH算法

CRUSH（control replication under scalable hash）算法是Ceph内部中重要的寻址算法，它主要解决对象到盘的寻址过程。同时由于是通过计算得出，因此集群不需要查询地址，所以也就没有中心节点的产生，极大的减少了元数据的体量。CRUSH可以将数据尽量的分散到各个存储上。

CRUSH算法主要使用的场景：数据io的时候，创建pool的时候，osd状态up/down，osd的添加和删除。3.2节所描述的，CRUSH主要是解决PG是如何映射到OSD列表的问题。用函数表示就是：

crush(pg.x) -> (osd.1, osd.2, osd.3,….osdN)

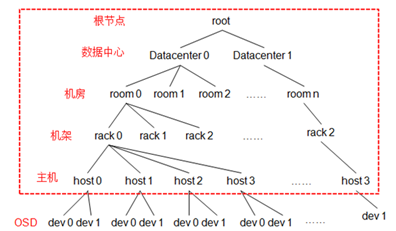


图 8 CRUSH Map

CRUSH map可以认为是数据中心的抽象，它的目的是寻找到合适的位置存放数据副本。CRUSH内容包括了组织结构Hierarchical CRUSH Map，副本选择规则Placement Rules以及容器Bucket。

层级化CRUSH Map，逻辑上看组织结构是个树形结构。包含device和bucket，device一般为osd服务作为叶子节点。bucket作为设备的容器，一般为CRUSH Map中的普通节点。bucket的类型有osd（device），host，chassis，rack，row，pdu，pod，room，datacenter，region，root，这些类型描述了CRUSH Map中的存储位置。每个bucket包含多个device。

Bucket Weight，权重是用来描述存储能力的指标。1T大小表示为1，采用双精度数表示。Bucket的权重大小是包含子bucket或device的权重大小。

Placement Rules决定了对象副本选择规则，它定义从哪些bucket或是device里去选择。这样可以定义不同的pool可以从不同的磁盘选择存放位置。

表格 1 bucket定义

|  |
| --- |
| # buckets  host cvknode146 {          id -2           # do not change unnecessarily          # weight 2.160          alg straw2          hash 0  # rjenkins1          item osd.1 weight 1.080          item osd.3 weight 1.080  }  host cvknode145 {          id -3           # do not change unnecessarily          # weight 2.160          alg straw2          hash 0  # rjenkins1          item osd.0 weight 1.080          item osd.4 weight 1.080  }  host cvknode144 {          id -4           # do not change unnecessarily          # weight 2.160          alg straw2          hash 0  # rjenkins1          item osd.2 weight 1.080          item osd.5 weight 1.080  }  rack rack0 {          id -7           # do not change unnecessarily          # weight 6.480          alg straw2          hash 0  # rjenkins1          item cvknode145 weight 2.160          item cvknode144 weight 2.160          item cvknode146 weight 2.160  }  root partition0 {          id -5           # do not change unnecessarily          # weight 6.480          alg straw2          hash 0  # rjenkins1          item rack0 weight 6.480  }  # rules  rule partition0\_rule {          ruleset 1          type replicated          min\_size 1          max\_size 10          step take partition0          step chooseleaf firstn 0 type host          step emit  }  rule partition0\_ec\_rule\_1 {          ruleset 2          type erasure          min\_size 3          max\_size 20          step set\_chooseleaf\_tries 5          step set\_choose\_tries 100          step take partition0          step chooseleaf indep 0 type host          step emit  } |

       表1中描述的是CRUSH Map的定义的Bucket，用树形图表示就是：

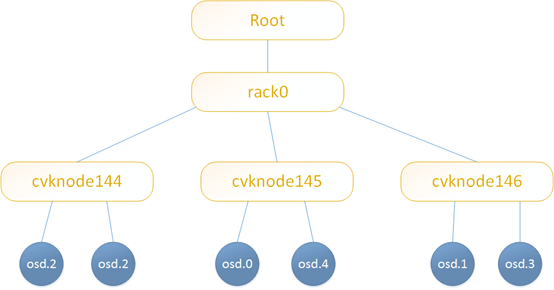


图 9 crush map图形化

图9所示，root作为crush map的入口，定义了bucket rack0，其中包括3个host，每个host包括了两个device osd。bucket的定义了随机选择算法为straw2，hash算法为rjenkins1。这里的hash算法是用来计算随机值，随机选择算法是根据随机值判定选择OSD。

表1中所定义的选择规则是partition0\_rule和partition0\_ec\_rule\_1。规则原型如下：

|  |
| --- |
| rule <rulename> {      ruleset <ruleset>      type [ replicated | erasure ]      min\_size <min-size>      max\_size <max-size>      step take <bucket-name> [class <device-class>]      step [choose|chooseleaf] [firstn|indep] <N> <bucket-type>      step emit  } |

ruleset：当前规则的id

type：pool的存储模式

min\_size：如果Pool的副本数小于min\_size则不使用这个规则

max\_size：如果Pool的副本数大于max\_size则不使用这个规则

step take <bucket-name> [class <device-class>]：

选择一个bucket，一般是root类型bucket，以这个为查询的输入，遍历这颗树。同时也可而已指定自定义的设备类型。

step choose firstn {num} type {bucket-type}：

深度优先选择num个bucket-type类型的子bucket。

l  If {num} == 0, choose pool-num-replicas buckets (all available).

l  If {num} > 0 && < pool-num-replicas, choose that many buckets.

l  If {num} < 0, it means pool-num-replicas - {num}.

如果num为0则选择和pool副本数一样的，num大于0小于pool的副本数，则返回num个副本数，如果num小于0，则返回pool副本数减num的个数。

step chooseleaf firstn {num} type {bucket-type}：

和上一条规则一样，区别在于chooseleaf是选择叶子节点，一般就是osd。

step emit：

输出选择结果。

3.4.1  Straw算法

现在有了crush的组织结构CRUSH Map和选择规则Placement Rules，从bucket中选出合适的设备是采用随机算法来做具体的选择。当前CRUSH所支持的随机算法有：

表格 2 随机选择算法

|  |  |
| --- | --- |
| Uniform | 适用于每个item权重相同，且很少添加或删除，item的数量比较确定。 |
| List | 以链表保存item，包含的item可以是任意权重，但是会造成某些节点被选中的概率变大。 |
| Tree | 采用2分查找树，包含大量item的情况下可以快速查找，但是会造成某些节点被选中的概率变大，而在节点删除添加移除和重新修改权重会引入额外组织变动开销，查找速度O(logn)。 |
| Straw | 相对List和Tree是比较公平的算法，每个节点都有公平竞争的机会，而且权重越大的节点被选中的机会越大。 |
| Straw2 | Straw的改进算法，减少了在节点删除移动的时候数据迁移量。 |

具体描述Straw算法，提供各个bucket尽量公平的选择机会，权重越大选中的概率越高。执行过程其实递归遍历bucket，找到合适device。如果权限大就尽量找权限大的，如果权限一样则随机返回。

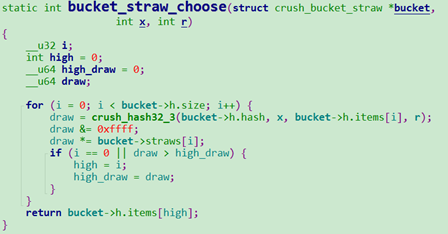


图 10 straw代码片段

       如代码所示：

1、  crush(pg\_id, osd\_id, r) => draw，r为常数，运算次数

2、  ( draw &0xffff ) \* osd\_weight => straw

3、  得到最大的high\_draw，返回该item

其中draw就是随机数，然后用draw乘以权重得到一个签值，选中最大签值的OSD。要保证不能每次都选中权重最大那个，随机数的产生就很重要。

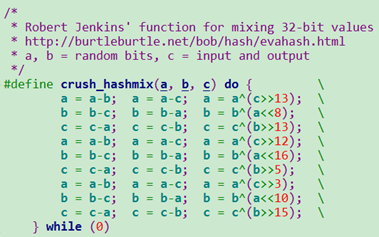


图 11 rjenkins hash算法

将3组数据传入进行混合搅拌，尽量得到一个随机值。可以看到只要传入的值是相同的，得到随机值也是相同的，所以CRUSH是一个伪随机的算法。

参考源码src/crush/mapper.c，入口函数crush\_do\_rule

3.5      ObjectStore模块

ObjectStore是Ceph系统落盘前的最后一个模块，它负责保证业务数据可以安全可靠高效的IO。ObjectStore定义事务操作，具体的本地存储必须实现这些操作。ObjectStore目前包括4种本地实现：

1、  FileStore：H、J版本使用较多的本地存储，采用文件系统做为对象读写的方式。

2、  BlueStore：最新L版本支持的本地存储，未来Ceph的默认方式，抛弃掉文件系统直接操作块设备，性能最好。

3、  KStore：使用KV存储系统作为本地存储。

4、  MemStore：数据和元数据都保存在内存中，主要用于测试和验证。

目前用于生产环境的是FileStore和BlueStore。

3.5.1  FileStore

FileStore默认使用xfs文件系统保存数据。利用文件系统POSIX接口实现ObjStore的接口，每个对象在底层是以文件存在。

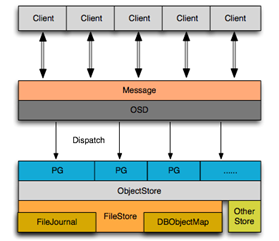


图 12 filestore结构

FileStore包含FileJournal和DBObjectMap两个模块，FileStore为了提高ObjectStore写事务处理能力和原子性引入了FileJournal。它相当于数据库的WAL(write ahead log)，为了保证每个写事务的完整性。它会使用direct io方式写到journal日志里，完成后再将事务提交到FileStore的队列中继续完成写操作，如果中途有发生crash，OSD在做recovery的时候会将日志恢复出来。

FileStore写数据顺序是，先写journal然后再写盘上。Journal写完后会返回给上层，但是要能read ready还是要等到数据落盘后才行，不过在大量小io随机写场景性能还是不错。FileStore由于先写日志再写盘，所以有个写放大的问题。

DBObjectMap是专门用来管理对象的属性的模块，有两种实现xattr和omap。xattr是用文件系统的扩展属性实现的，受限于文件系统扩展属性的长度限制，适合小量数据存放。omap是采用leveldb k-v键值存储实现，如果属性大小超过xattr的限制，则可以存放到omap中。

3.5.2  Bluestore

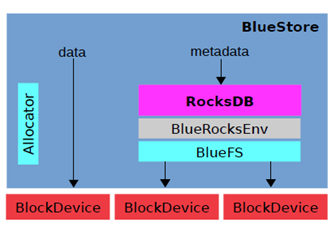


图 13 bluestore整体结构

Bluestore为解决filestore在写数据前先写journal产生的写放大，并针对ssd做了相应的优化。Bluestore直接操作裸盘，尽可能的避免文件系统（xfs、ext4）的开销。操作裸盘则盘空间管理由allocator（默认是BitmapAllocator）处理。Bluestore在io过程中产生的元数据，元数据则存放到rocksdb kv数据库中。rocksdb存数据依赖文件系统，但是rocksdb将底层系统相关的抽象为env，用户程序只需要实现相应的接口就可以提供底层的系统的封装。Bluestore实现了BlueRocksEnv，并实现了Bluefs支撑BlueRocksEnv。Bluefs的日志和数据文件都是保存在裸盘上，和用户数据共享裸盘设备，也可以分开指定不同设备。

https://static.oschina.net/uploads/space/2018/0223/092907_sb8u_2874076.png

图 14 osd设备数据分区

osd目录分区：默认占用100M，默认挂载xfs文件系统。提供基本的描述信息，包括whoami（osd编号），osd的type，osd的魔术字，block块设备入口等。

Block dev label分区：默认占用4K。存储bluestore\_bdev\_label\_t结构体信息，包含osd\_uuid，块设备size，设备描述信息，创建时间等。

Bluefs supper block分区：默认占用4K。存储bluefs\_super\_t结构体信息，包含osd\_uuid，version和block\_size等。

DB数据分区：默认占用MAX[1G，块设备总大小的4%]，保存DB数据和日志数据。

用户数据分区：默认占用剩余空间，存放用户业务数据。

元数据映射关系：

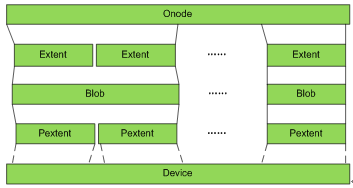


图 15 元数据映射关系

一个对象对应一个onode，一个onode包含多个extent，多个extent存放在一个或者多个blob里，每个blob又对应到多个pextent，就是具体的物理块，这样就完成了extent到pextent的映射。

BlueStore读过程比较简单，如果有预读标志则从cache里读，如果没命中则需要从盘上读出来并且添加到cache中。

Bluestore写过程比较复杂，大致分为大写和小写。根据写入数据长度来确定是小写还是大写，通过对min\_alloc\_size（blob的最小分配大写默认64K）取模，如果小于64K为小写流程，如果大于64K则为大写流程。

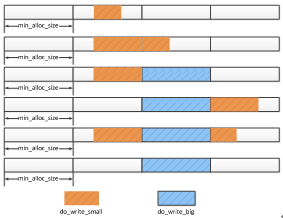


图 16 写场景

小写分为覆盖写和非覆盖写，其中非覆盖小写和大写没有写惩罚，先写数据然后改元数据。覆盖写场景会产生WAL日志，先要完成WAL日志处理落盘后，再写入业务数据。

整体来说性能比filestore提高不少。

4      数据peering和恢复

Ceph系统中承担数据IO和数据恢复的基本逻辑单元叫PG（Placement Group），PG是Ceph中数据载体。可以理解PG为一个目录或者集合，包含了多个对象。

PG有自己的状态，PG健康度出现异常，状态会发生变迁，数据也会发生迁移。

active+clean：PG的健康时的状态。

Degraded：降级状态，如果是双副本的配置，有个osd挂掉了，PG就会进入这种状态。这种情况下PG还是可以提供IO服务的。

Recovery：一个osd出现故障下线了，但这时pg还是可读写的，当故障osd启动后，它所保存的数据比其他osd版本要旧这个时候就会产生recovery，保证数据一致。

Remapped：如果osd下线是由于硬盘故障，这个时候就是永久性故障，需要再分配新osd做重新映射。选中新osd后，数据还是为空的，需要从正常的osd上将数据全量拷贝到新osd上，所以状态一般是remapped+backfilling。

Peered：出现这种状态，是由于故障的osd数量过多，造成了pg不能读写了。这个状态可以理解为正常的osd在等待其他副本osd上线。

4.1      数据peering机制

Peering过程是一个PG对应的一组OSD的状态达到一致的过程，PG达到active时，peering过程就完成了，但是PG对应的OSD上的数据并不一定完全一致。

PG发生的场景：

n  系统初始化OSD启动重新加载PG或者创建PG，会触发PG发起一次Peering过程。

n  OSD故障或OSD增加减少，会导致PG的acting set发生变化，PG也会发起一次Peering过程。

peering概念介绍

1、  acting set和up set

acting set是一个PG活动的一组OSD，该列表是有序的，下标第一个OSD为prime OSD。up set一般与acting set相同。不相同的情况是OSD故障导致，生成临时PG。这个时候acting set和up set不一致。

2、  临时pg

原OSDacting set列表中[0,1,2]的主OSD osd.0故障了，crush重新选出的acting set为[3,1,2]，但是这个时候osd3并没有该PG的数据，需要做backfill，这个时候osd3是不能读的。所以会产生一个临时PG做为prime，比如通知monitor让osd1做临时prime，这个时候up set就为[1,3,2]，acting set还是[3,1,2]，backfill做完，临时pg取消，两个列表就想同了。

3、  Authoritative History权威日志

指的是记录pg完整顺序的连续的操作日志记录，做为数据恢复的依据。

Peering的过程，基本分为三个步骤

l  GetInfo : pg的主osd通过发送消息获取各个从OSD的pg\_info信息。

l  GetLog：根据pg\_info的比较，选择一个拥有权威日志的osd（auth\_log\_shard） , 如果主osd不是拥有权威日志的osd，就去拥有权威日志的osd获取，获取后主osd也拥有权威日志。

l  GetMissing：拉取其它从OSD 的pg log（或者部分获取，或者全部获取FULL\_LOG) , 通过本地的auth log对比，来判别从OSD 上缺失的object 信息。以用于后续recovery过程的依据。

l  Active: 激活主osd，并发想通知notify消息，激活相应的从osd。

简单来说peering并不恢复数据，只是将各个osd的状态统一起来，明确哪些对象需要恢复，为下一步做准备。

4.2      数据恢复

Peering完成后，就可以知道是否有数据需要恢复了。恢复的方式有两种recovery和backfill。

Recovery是就根据PG日志中缺失的记录来修复不一致的对象。

Backfill是PG通过重新扫描所有对象，对比发现缺失的对象，通过整体拷贝的方式修复。当OSD失效时间过长导致无法根据PG日志记录来修复，或者新OSD加入导致的数据迁移，这些都会导致Backfill流程。

5      数据端到端一致性

存储系统中io路径复杂，传统存储系统一般包括从应用层到内核文件系统、块设备、SCSI层再到HBA和磁盘控制器，每层都有出错的可能。因此传统的端到端解决方案会以数据块校验为主来解决。

 因为 Ceph 作为一个应用层的应用，这时候如果封装固定数据块并且加入校验数据会导致较严重的性能问题，因此 Ceph 在这方面只是引入 Scrub 机制(Read Verify)来保证数据的正确性。

简单来说，Ceph 的 OSD 会定时启动 Scrub 线程来扫描部分对象，通过与其他副本进行对比来发现是否一致，如果存在不一致的情况，Ceph 会抛出这个异常交给用户去解决。

Scrub按照扫描内容不同分为两种方式：

n  一种叫srub，它只是比较各个对象的副本元数据，来检查数据一致性。只是检查元数据，所以读取和计算的量都比较小，一种轻量的检查。

n  一种叫deep-srub，它要进一步检查对象的数据内容是否一致，实现深度扫描，几乎要扫描磁盘上的所有数据并计算crc32校验，所有开销很大，占用系统资源很多。

Scrub按照扫描的方式分为两种：

n  在线扫描：不影响系统正常业务。

n  离线扫描：需要系统停或冻结业务。

Ceph的Scrub是实现了在线扫描，可不中断系统业务的前提下执行scrub，但是针对具体某个对象做Scrub，是会锁定对象的，阻止客户端对它的访问，一直到Scrub执行完成才会释放。