参考资料：

<https://blog.csdn.net/majianting/article/details/105520674>

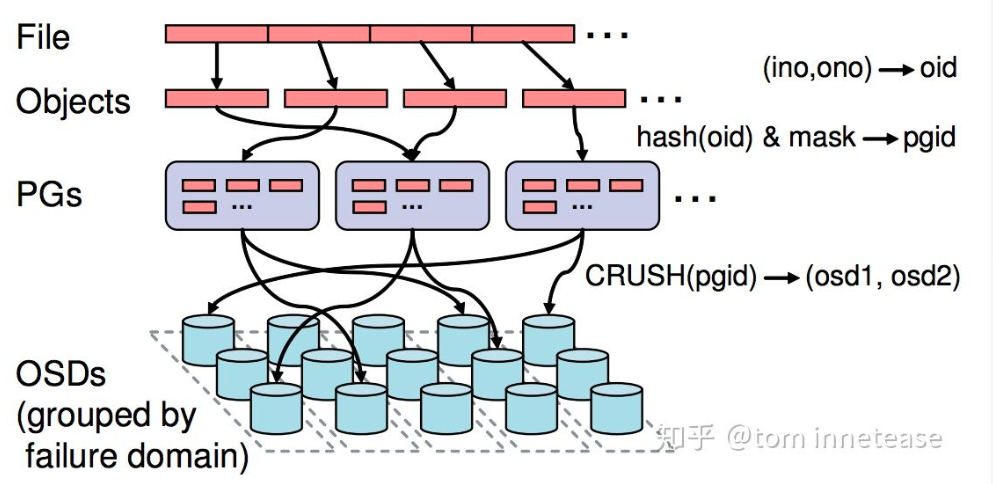
1、背景

在分布式存储系统中，数据的位置存放规则一直是研究的热门话题之一。一般来说，系统中所有角色（Clients、Servers）需要有一个统一的数据寻址算法Locator，满足：

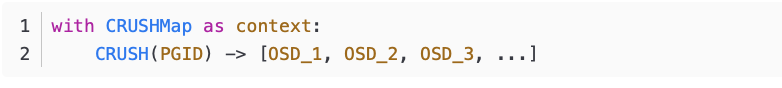
Locator(ID) -> [Device\_1, Device\_2, Device\_3, ...]

其中输入ID是数据的唯一标识符，输出Device列表是一系列存储设备（多设备冗余以达到多份数据保护或切分提高并发等效果）。早期的直观方案是维护一张全局的Key-Value表，任何角色操作数据时查询该表即可。显然，随着数据量的增多和集群规模的扩大，要在整个系统中维护这么一张不断扩大的表变得越来越困难。CRUSH(Controlled Replication Under Scalable Hashing)即为解决此问题而生，她仅需要一份描述集群物理架构的信息和预定义的规则（均包含在CRUSH map中），便可实现确定数据存储位置的功能。

在Ceph的RADOS中，还引入了PG的概念用以更好地管理数据，如下图所示：

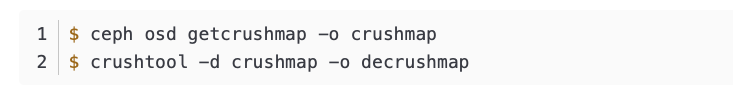


RADOS提供的是基于object的存储功能，每个object会先通过简单的Hash算法归到一个PG中，PGID再作为入参通过CRUSH计算置入到多个OSD中（这里object可以看作是文件，PG即是一个目录，OSD是一个数据根目录下仅有一级子目录的简易文件系统）。因此，CRUSH提供的功能可以描述为：



2、CRUSH Map

CRUSH map包含了集群正常情况下完成PG定位所需要的绝大部分信息，它可以通过以下命令获取：



直接获取到的crushmap是编码过的，需要通过crushtool转换才能变成人可读的文本文件。decrushmap一般包含以下几个部分：

1）Tunables: 可调整的参数列表（仅一部分，非完整列表）

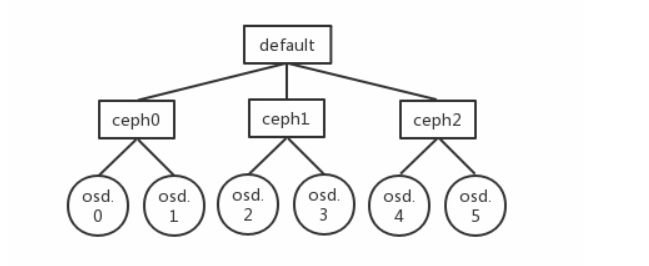
2）Devices: 存储设备列表，列举了集群中所有的OSD

3）Types: 类型定义，一般0为OSD，其它正整数代表host、chassis、rack等

4）Buckets: 容器列表，指明了每个bucket下直接包含的children项及其权重值（非OSD的items统称为bucket）

5）Rules: 规则列表，每个规则定义了一种选取OSD的方式

下面的案例是一个非常简单的版本，描述了如下图所示的物理架构：



*# begin crush map*

*tunable choose\_local\_tries 0*

*tunable choose\_local\_fallback\_tries 0*

*tunable choose\_total\_tries 50*

*tunable chooseleaf\_descend\_once 1*

*tunable straw\_calc\_version 1*

*# devices*

*device 0 osd.0*

*device 1 osd.1*

*device 2 osd.2*

*device 3 osd.3*

*device 4 osd.4*

*device 5 osd.5*

*# types*

*type 0 osd*

*type 1 host*

*type 2 chassis*

*type 3 rack*

*type 4 row*

*type 5 pdu*

*type 6 pod*

*type 7 room*

*type 8 datacenter*

*type 9 region*

*type 10 root*

*# buckets*

*host ceph0 {*

*id -2 # do not change unnecessarily*

*# weight 2.000*

*alg straw2*

*hash 0 # rjenkins1*

*item osd.0 weight 1.000*

*item osd.1 weight 1.000*

*}*

*host ceph1 {*

*id -3 # do not change unnecessarily*

*# weight 2.000*

*alg straw2*

*hash 0 # rjenkins1*

*item osd.2 weight 1.000*

*item osd.3 weight 1.000*

*}*

*host ceph2 {*

*id -4 # do not change unnecessarily*

*# weight 2.000*

*alg straw2*

*hash 0 # rjenkins1*

*item osd.4 weight 1.000*

*item osd.5 weight 1.000*

*}*

*root default {*

*id -1 # do not change unnecessarily*

*# weight 6.000*

*alg straw2*

*hash 0 # rjenkins1*

*item ceph0 weight 2.000*

*item ceph1 weight 2.000*

*item ceph2 weight 2.000*

*}*

*# rules*

*rule replicated\_ruleset {*

*ruleset 0*

*type replicated*

*min\_size 1*

*max\_size 10*

*step take default*

*step chooseleaf firstn 0 type host*

*step emit*

*}*

*# end crush map*

3. CRUSH Rule

当集群系统变得复杂时，我们往往需要多种策略来存放不同类型的数据；例如，将热数据存放于SSD中，而将冷数据放在HDD中。CRUSH rule就是定义了一种选择策略，Ceph中每个逻辑池都必须对应一条合法的rule才能正常工作。还是以上面链接中的crushmap为例，分析一下rule的具体含义。



Line 1-2: rule的名称和编号

Line 3: rule类型；replicated代表适用于副本池，erasure代表适用于EC池

Line 4-5: rule适用的池size大小；本例表示1副本到10副本的池均可采用此条rule

Line 6-9: 具体的选择步骤

take: 直接选中一项item，一般用于指定选择算法的起点；本例中名为default的root类型bucket即为起点

chooseleaf firstn 0 type host: 选择策略，这个是重点，后面细述

emit: 返回结果

4、选择策略

上述rule中Line 7包含了4部分信息，可以分解为: step <1> <2> <3> type <4>

<1>: choose/chooseleaf

choose表示选择结果类型为故障域（由<4>指定）

chooseleaf表示在确定故障域后，还必须选出该域下面的OSD节点（即leaf）

<2>: firstn/indep

firstn: 适用于副本池，选择结果中rep（replica，指一份副本或者EC中的一个分块，下同）位置无明显意义

indep: 适用于EC池，选择结果中rep位置不可随意变动

举例来说，副本池中每份副本保存的是完全相同的数据，因此选择结果为[0, 1, 2]（数字代表OSD编号）与[0, 2, 1]并无大的不同。但是EC池不一样，在2+1的配比下前两份是数据块，最后一份是校验块，后两份rep位置一交换就会导致数据出错。

<3>: num\_reps

这个整数值指定需要选择的rep数目，可以是正值负值或0。正整数值即代表要选择的副本数，非常直观

0表示的是与实际逻辑池的size相等；也就是说，如果2副本池用了这个rule，0就代表了2；如果3副本池用了此rule，0就相当于3

负整数值代表与实际逻辑池size的差值；如果3副本池使用此rule将该值设为了-1，那边该策略只会选择出2个reps

<4>: failure domain

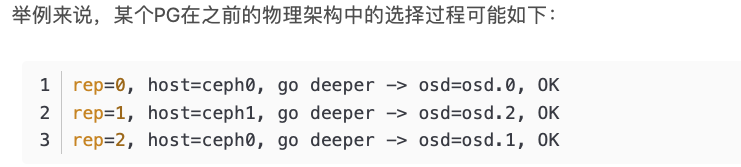
指定故障域类型；CRUSH确保同一故障域最多只会被选中一次。

4、Rule执行流程

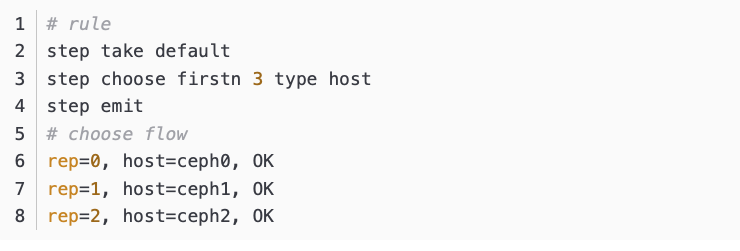
接下来介绍选择策略的核心实现流程。为简单起见，以下面的rule step为例：







可见，最后我们选中了同一个host ceph0下的两个OSD。在实际应用中，通常不会以OSD为故障域，而是使用高级的bucket（如host，rack）等作为故障域，那么结果会怎么样呢？以改用host为例：



显然，最后只选到了host而没有OSD，那么数据的最终存放位置并没有确定，这个rule的设置不合理。这里有两个解决方案

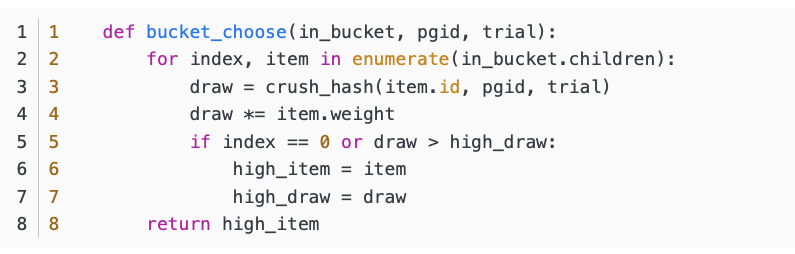




这样，在选中一个failure\_domain type的bucket后，会递归调用一次choose函数来选择一个该bucket下的OSD。到此，CRUSH rule实现的整体逻辑就清楚了，接下来再进一步分析里面的细节。

4.1、random\_pick\_from\_list

这个函数是CRUSH里面非常重要的一部分，代码中一般调用的函数为crush\_bucket\_choose。每个bucket有一项属性alg(algorithm)，用来确定在该bucket的children列表中如何选出一个合适的item。目前的实践环境中一般使用straw或者straw2算法（早期的alg还有uniform、list、tree等，但均因各种缺陷而被逐渐遗忘；straw2是straw算法的修正改进版本，也是最新的默认选项），这里仅介绍straw2的实现思路。顾名思义，straw类算法可以看作是让bucket下的每个item随机抽一根签（本质上就是一个数值），然后互相比较选出签长最长的那个item；流程如下：



其中crush\_hash可以简单地看成是一个伪随机的hash函数：它接收3个整数作入参，返回一个固定范围内的随机值。同样的输入下其返回值是确定的，但是任何一个参数的改变都会导致其返回值发生变化。Weight是每个item的权重值（对于OSD来说，weight值与硬盘容量成正比；bucket的weight值即其下children weight值的总和），显然line 4可以使得weight值大的item被选中的几率升高。bucket\_choose函数有以下几个特点：

1）对于确定的bucket，不同的pgid能返回不同的结果

2）对于确定的bucket和pgid，调整trial值可以获得不同的结果

3）对于确定的pgid和trial值，如果bucket内item增加或删除或调整weight，返回结果要么不变，要么就变更到这个发生变化的item上

这里特点1和2是比较直观的，简单讨论下第三点。假设新增了一个new\_item，比对流程可以发现，只需比较原来的high\_item和new\_item的draw即可，因此返回结果要么仍然是原来的high\_item，要么就是new\_item，不会出现变为另一个旧的item的情况；同样，假设删除了一个old\_item，如果原来它就是high\_item，那么会有一个新的item被选择出来，如果它不是high\_item，那么返回结果依然是旧的high\_item，不会发生变化。

这些特性在一定程度上保证了CRUSH算法的稳定性，即我们期望集群设备的增删仅影响到必要的节点，不要在正常运行的OSD上做无意义的数据迁移。

4.2、is\_available

这个函数主要是判断指定的item是否可用，如果不可用，则应将其抛弃而重新选择。此函数的判断分为两部分：

1）item被选取过，则返回False；这就确保了同一故障域最多被选中一次

2）如果item是OSD且OSD状态不正常，则返回False。这里第一次考虑到了OSD的状态，其实已经超出了CRUSH map的范围，而引入了osdmap的信息，在此仅作简要说明：查看OSD的reweight值（参考命令ceph osd tree），若其为1.0，则返回True；若其为0（即通常所说out状态）则返回False；若其为中间值，则概率性选择返回值。

4.3、while True

在一些特殊情况（如大量OSD处于out状态）下，CRUSH算法不保证一定可以选出要求的所有replicas，因此其设定了一些重试参数，而不会真正陷入死循环。当达到重试次数限制时，会放弃该replica选择。综上所述，CRUSH rule的实现流程可以描述为：



*备注说明*

*1）本节所列的所有伪代码仅用作原理说明，实际的代码流程需要更多的细节考虑，会有较大出入，具体请参考Ceph源码*

*2）Ceph系统中PG的实际操作并不直接以CRUSH的输出为结果，还需考虑其他因素如OSD down、PG temp、primary affinity等*

5. Rule流程举例

以下通过几个简化的例子阐述CRUSH算法在实际应用中遇到的一些问题及可能的解决方案。所写流程示例仅针对某个特定的PG，用r代表trial的值，使用的选择策略为：chooseleaf firstn 0 type host。



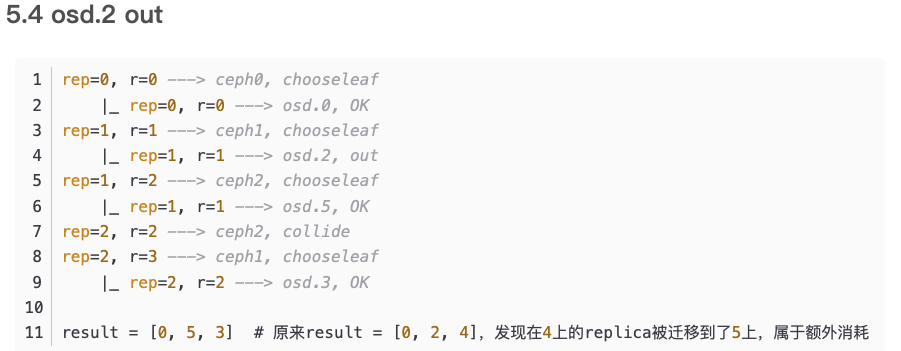


这里的主要原因在于每次递归调用时，r取值为2，且MAX\_TRIES被设置成了1。因此，每次在ceph2下都会选中osd.4，而此osd已经out，只能退回上层重试。而上层只有3个host的情况下，没有其余选择，导致最终失败；这也警示我们在故障域个数与逻辑池size相等时，应慎重考虑。此问题的解决方案也很多，如：

1）增加新的host

2）将chooseleaf\_vary\_r参数设置成大于0，可以在递归调用中传入不同的r值，选出ceph2下的其它OSD

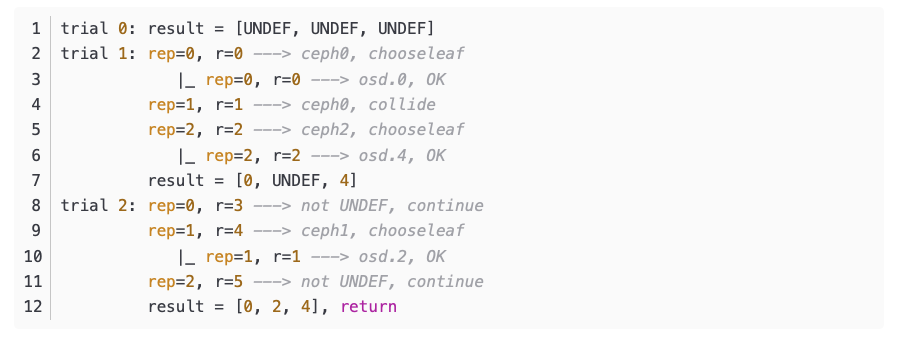
3）设置chooseleaf\_tries或将chooseleaf\_descend\_once改为false等，会调整递归调用中的MAX\_TRIES值，达到ceph2下多次重试的效果



这里的主要原因在于递归调用时r取值等于rep值，在replica发生位置变化时返回了不同的结果。要想结果稳定可以将每次递归调用的r设成一个固定值，如0；这点可以通过将chooseleaf\_stable参数设为true实现。

5.5 EC indep

适用于EC的indep类rule的流程大体上与firstn类似，只是需要为没选出的item留好位置，并且在每一次trial中，都遍历一轮所有的replicas。以下以chooseleaf indep 0 type host为例简要说明：



可见，trial 1中rep 1选择失败，其位置被留了出来；osd.4直接放在了最后一个replica的位置。在trial 2中，其实仅选择了一个replica，rep=0和rep=2会直接跳过。

6、CRUSH特点

最后就笔者个人理解分析一下CRUSH算法的特点，一言以蔽之，依赖于Hash实现的纯伪随机算法。具体来说：

1）计算独立性：每次计算完全独立，不依赖于集群分配情况或已选择结果（先计算再判断冲突，而不是将冲突项从备选项中移除）；仅依靠多次重试解决选择失败问题

2）稳定性：只有OSD的增删或者weight/reweight变化才会影响到计算结果；正常运行时结果不变

3）可预测性：通过对指定的CRUSH map进行离线计算即可预测出PG的分布情形，且与集群内实际使用完全一致

虽然CRUSH算法为Ceph数据定位提供了有力的技术支持，但也依然存在一些缺陷，如：

1）假失败：因为计算的独立性CRUSH很难处理权重失衡（weight skew）的情形。例如，假设3个hosts的weight值分别为10，10，1，MAX\_TRIES为50，现已经选中了前两个hosts，那么第三个replica有大约(20/21)^50=8.72%的概率选择失败，即使低weight的那个host其实是可用的。因此，在实践中应尽量避免权重失衡的情形出现。

2）故障额外迁移：上节5.4仅解决了OSD状态由in到out的额外迁移，实际环境中还会因为OSD的增删产生一定量的数据额外迁移，对集群造成影响。

3）使用率不均衡：这也是CRUSH被诟病最多的缺陷，即完全依赖Hash的随机导致集群中OSD的容量使用率出现明显失衡（实践中遇到过差40%以上），造成空间浪费。因此，自Luminous版本起，Ceph提供了被称为upmap的新机制（可以看成记录了一张特例表），用以手动指定PG的分布位置，来达到均衡数据的效果。

不管怎样，相信随着社区的发展和新思路的不断涌现，CRUSH算法的这些缺陷最终都能被一一解决，或是出现更优秀的算法来替代之，让我们一起参与，一起期待。